User Guide

AWS HealthOmics

Version latest

AWS HealthOmics: User Guide

Table of Contents

What is AWS HealthOmics? ... 1
Important notice ... 1
HealthOmics features ... 1
Concepts ... 2
Workflows ... 3
Storage ... 3
Analytics ... 4
Related services ... 4
How to access HealthOmics ... 4
Regions and endpoints for AWS HealthOmics ... 5
Learn more ... 5
AWS HealthOmics variant store and annotation store availability change ... 6
Overview of migration options ... 6
Migration options for ETL logic ... 7
Migration options for storage ... 7
Analytics ... 7
AWS Partners ... 7
Examples ... 8
Athena DDL ... 8
Create tables using Python (without Athena) ... 8
Setting up HealthOmics ... 12
Sign up for an AWS account ... 12
Create a user with administrative access ... 13
Create IAM permissions for HealthOmics ... 14
Connect with external code repositories ... 14
Using Amazon Q CLI with HealthOmics ... 15
Getting started ... 16
Using a Ready2Run workflow in the HealthOmics console ... 16
Example prompts for Amazon Q CLI ... 16
Private workflows ... 18
Creating workflows ... 19
Git repository integration ... 20
Workflow definition files ... 23
Parameter template files ... 77

Container images ... 90
Workflow README files ... 102
Optional: Sentieon licenses ... 105
Workflow linters ... 107
Workflow operations ... 107
Workflow versioning ... 124
Default version ... 125
Create a version ... 126
Update a version ... 133
Delete a version ... 135
HealthOmics runs ... 136
Run storage types ... 137
Run retention modes ... 140
Run inputs ... 141
Run lifecycle ... 146
Run outputs ... 150
Run failure reasons ... 152
Task lifecycle ... 156
Run optimization ... 159
Run operations ... 166
Run groups ... 178
Run priority ... 179
Create a run group using the console ... 179
Create a run group using the CLI ... 180
Delete a run group using the console ... 181
Delete a run group using the CLI ... 181
Call caching ... 182
How call caching works ... 182
Creating a run cache ... 188
Updating a run cache ... 190
Deleting a run cache ... 190
Contents of a run cache ... 191
Engine-specific caching features ... 192
Using the run cache ... 193
Sharing workflows ... 197
Subscribing to a shared workflow ... 198

Monitoring status of a workflow share ... 198
Sharing a private workflow using the console ... 199
Sharing a private workflow using the CLI ... 199
Accepting a shared workflow using the console ... 200
Running a shared workflow using the console ... 200
Running a shared workflow using the API ... 201
Ready2Run workflows ... 202
Available workflows ... 203
Subscribing to Sentieon workflows ... 209
Starting Ready2Run workflows (console) ... 209
Starting Ready2Run workflows (API) ... 210
HealthOmics storage ... 212
HealthOmics ETags ... 213
Amazon S3 ETags ... 213
How HealthOmics calculates ETags ... 214
Creating a reference store ... 215
Creating a reference store using the console ... 215
Creating a reference store using the CLI ... 216
Creating a sequence store ... 221
Creating a sequence store using the console ... 221
Creating a sequence store using the CLI ... 222
Updating a sequence store ... 224
Updating read set tags for a sequence store ... 225
Importing genomic files ... 225
Deleting stores ... 226
Importing read sets into a sequence store ... 227
Upload files to Amazon S3 ... 227
Creating a manifest file ... 228
Starting the import job ... 231
Monitor the import job ... 231
Find the imported sequence files ... 233
Get details about a read set ... 236
Download the read set data files ... 237
Direct upload to a sequence store ... 238
Direct upload to a sequence store using the AWS CLI ... 238
Configure a fallback location ... 244

Exporting read sets ... 245
Accessing read sets with Amazon S3 URIs ... 247
Amazon S3 URI structure in HealthOmics storage ... 249
Using Hosted or Local IGV to access read sets ... 249
Using Samtools or HTSlib in HealthOmics ... 250
Using Mountpoint HealthOmics ... 250
Using CloudFront with HealthOmics ... 251
Activating read sets ... 251
HealthOmics analytics ... 255
Creating variant stores ... 256
Creating a variant store using the console ... 256
Creating a variant store using the API ... 257
Creating variant store import jobs ... 259
Creating annotation stores ... 263
Creating an annotation store using the console ... 263
Creating an annotation store using the API ... 264
Creating annotation store import jobs ... 266
Creating an annotation import job using the API ... 266
Additional parameters for TSV and VCF formats ... 268
Creating TSV formatted annotation stores ... 269
Starting VCF formatted import jobs ... 272
Creating annotation store versions ... 273
Deleting analytics stores ... 276
Querying analytics data ... 277
Configuring Lake Formation ... 278
Configuring Athena for queries ... 281
Runnning queries ... 282
Sharing analytics stores ... 284
Creating a store share ... 284
Resource sharing ... 286
Creating a share ... 286
Retrieve information about a share ... 287
View the shares that you own ... 288
View accepted shares from other accounts ... 288
Delete a share ... 288
Tagging resources in HealthOmics ... 289

Important notice ... 289
Tagging HealthOmics resources ... 289
Best practices ... 291
Tagging requirements ... 291
Sequence store read set tags ... 291
Adding a tag ... 292
Listing tags ... 293
Removing tags ... 293
Permissions ... 295
User policies ... 295
Define custom IAM permissions for runs ... 297
Service roles ... 298
Example IAM service policies ... 299
Example CloudFormation template ... 302
Amazon ECR permissions ... 303
Create a resource policy for the Amazon ECR repository ... 304
Running workflows with cross-account containers ... 305
Amazon ECR policies for shared workflows ... 307
Policies for Amazon ECR pull through cache ... 309
Resource permissions ... 314
Lake Formation permissions ... 314
Amazon S3 URI Permissions ... 315
Policy based sharing ... 316
Example Restriction ... 320
Security ... 323
Data protection ... 323
Encryption at rest ... 324
Encryption in transit ... 335
Identity and access management ... 335
Audience ... 335
Authenticating with identities ... 336
Managing access using policies ... 337
How AWS HealthOmics works with IAM ... 339
Identity-based policy examples ... 345
AWS managed policies ... 348
Troubleshooting ... 351

Compliance validation ... 353
Resilience ... 355
VPC endpoints (AWS PrivateLink) ... 356
Considerations for HealthOmics VPC endpoints ... 356
Creating an interface VPC endpoint for HealthOmics ... 356
Creating a VPC endpoint policy for HealthOmics ... 357
Special considerations for accessing read sets using Amazon S3 URIs ... 358
Monitoring AWS HealthOmics ... 359
S3 access logging ... 360
CloudWatch metrics ... 360
Viewing AWS HealthOmics metrics ... 361
Creating an alarm ... 361
CloudWatch Logs ... 362
Log types for HealthOmics workflows ... 363
Logs in CloudWatch ... 364
Logs in Amazon S3 ... 365
Interactive CloudWatch Logs in the CLI ... 365
Accessing CloudWatch Logs from the console ... 366
CloudTrail logs ... 366
HealthOmics information in CloudTrail ... 367
Understanding HealthOmics log file entries ... 368
EventBridge ... 369
Set up EventBridge for HealthOmics ... 370
EventBridge events in HealthOmics ... 371
Event message structure ... 373
Event message examples ... 373
Troubleshooting ... 377
Troubleshooting workflows ... 377
How do I troubleshoot a failed run? ... 377
How do I troubleshoot a failed task? ... 377
Where do I find the engine logs for successfully completed runs? ... 378
How can I reduce the input parameter size for a workflow? ... 378
Why is my run not completing? ... 378
Troubleshooting call caching issues ... 378
Why isn't my run saving to the cache? ... 378
Why isn't a task using the cache entry? ... 378

Why is the call caching for a task disabled? ... 379
Troubleshooting data stores ... 379
Why is S3 GetObject failing on my read set? ... 380
Why can't I see my annotation store or variant store in Athena? ... 380
Why can't I access my data store in Athena? ... 380
Troubleshooting with Amazon Q CLI ... 381
Quotas ... 382
Service quotas ... 382
Fixed size quotas ... 387
Analytics file size quotas ... 387
Storage file size quotas ... 388
Workflow fixed size quotas ... 389
Ready2Run workflow fixed size quotas ... 392
API quotas ... 395
General API quotas ... 395
Storage API quotas ... 396
Workflow API quotas ... 397
Analytics API quotas ... 398
Document history ... 400

What is AWS HealthOmics?

AWS HealthOmics is a HIPAA-eligible service that accelerates clinical diagnostic testing, drug discovery, and agriculture research by fully managing the complex infrastructure behind your bioinformatics workflows. HealthOmics supports industry-standard workflow languages (WDL, Nextflow, CWL) and seamlessly scales bioinformatics infrastructure to support data from tens of thousands of tests per day—all with predictable cost per-sample. HealthOmics handles the technical complexities like managing compute resources and maintaining workflow engines so you can focus entirely on scientific breakthroughs.

Topics

- Important notice
- HealthOmics features
- HealthOmics concepts
- Related services
- How to access HealthOmics
- Regions and endpoints for AWS HealthOmics
- Learn more

Important notice

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. 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 features

Primary use cases for HealthOmics:

- *Clinical diagnostics* – Build and scale diagnostic testing workflows with predictable costs and fully managed infrastructure that grows with your testing volume.

- *Drug discovery* – Accelerate therapeutic research by orchestrating biological foundation models at scale, enabling rapid iteration across millions of potential candidates.
- *Agricultural research* – Enhance crop traits like drought tolerance and pest resistance through AI-powered workflows that improve food security and agricultural productivity.

Key benefits of HealthOmics:

- *Scalability* – Scale workflows across 100,000+ concurrent vCPUs to support tens of thousands of tests daily with zero infrastructure management and predictable cost-per sample.
- *Focus on science, not infrastructure* – Use familiar workflow languages and APIs while AWS automatically handles infrastructure orchestration and data management behind the scenes.
- *Maintain compliance* – Comprehensive audit trails, data provenance tracking, and HIPAA-eligible infrastructure designed for clinical workflows—all out-of-the-box—support development of solutions that meet regulatory requirements.

HealthOmics consists of three main components:

- HealthOmics workflows — Run bioinformatics computations on automatically provisioned and scaled infrastructure.
- HealthOmics storage — Store and share petabytes of genomics data efficiently at a low cost per gigabase.
- HealthOmics analytics — Prepare genomics data for multiomics and multimodal analyses.

Use these components independently or combine them for an end-to-end solution.

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 this guide.

Topics

- Workflows
- Storage
- Analytics

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* – 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 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.

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.

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 AI – Use SageMaker AI to run HealthOmics tasks using Jupyter notebooks.
- GitHub connections – Use connections to connect your external code respoitories to your HealthOmics workflows.

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.

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.

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).

AWS HealthOmics variant store and annotation store availability change

After careful consideration, we decided to close AWS HealthOmics variant stores and annotation stores to new customers starting November 7th, 2025. Existing customers can continue to use the service as normal.

The following section describes migration options to help you move your variant stores and analytics stores to new solutions. For any questions or concerns, create a support case at support.console.aws.amazon.com.

Topics

- Overview of migration options
- Migration options for ETL logic
- Migration options for storage
- Analytics
- AWS Partners
- Examples

Overview of migration options

The following migration options provide an alternative to using variant stores and annotation stores:

1. Use the HealthOmics-provided reference implementation of ETL logic.

   Use S3 table buckets for storage and continue to use existing AWS analytics services.
2. Create a solution using a combination of existing AWS services.

   For ETL, you can write custom Glue ETL jobs, or use open-source HAIL or GLOW code on EMR, to transform variant data.

   Use S3 table buckets for storage and continue to use existing AWS analytics services
3. Select an AWS partner that offers a variant and annotation store alternative.

Migration options for ETL logic

Consider the following migration options for ETL logic:

1. HealthOmics provides the current variant store ETL logic as a reference HealthOmics workflow. You can use this workflow's engine to power exactly the same variant data ETL process as the variant store, but with full control over the ETL logic.

   This reference workflow is available by request. To request access, create a support case at support.console.aws.amazon.com.
2. To transform variant data, you can write custom Glue ETL jobs, or use open-source HAIL or GLOW code on EMR.

Migration options for storage

As a replacement for service-hosted data store, you can use Amazon S3 table buckets to define a custom table schema.For more information about table buckets, see Table buckets in the *Amazon S3 User Guide*.

You can use table buckets for fully managed Iceberg tables in Amazon S3.

You can raise a support case to request the HealthOmics team to migrate the data from your variant or annotation store to the Amazon S3 table bucket that you configured.

After your data is populated in the Amazon S3 table bucket, you can delete your variant stores and annotation stores. For more information, see Deleting HealthOmics analytics stores.

Analytics

For data analytics, continue to use AWS analytics services, such as Amazon Athena, Amazon EMR, Amazon Redshift, or Amazon Quick.

AWS Partners

You can work with an AWS partner that provides customizable ETL, table schemas, built-in query and analysis tools, and user interfaces for interacting with data.

Examples

The following examples show how to create tables suitable for storing VCF and GVCF data.

Athena DDL

You can use the following DDL example in Athena to create a table suitable for storing VCF and GVCF data in a single table. This example isn't the exact equivalent of the variant store structure, but it works well for a generic use case.

Create your own values for DATABASE_NAME and TABLE_NAME when you create the table.

```
 CREATE TABLE <DATABASE_NAME>. <TABLE_NAME> (
  sample_name string,
  variant_name string COMMENT 'The ID field in VCF files, '.' indicates no name',
  chrom string,
  pos bigint,
  ref string,
  alt array <string>,
  qual double,
  filter string,
  genotype string,
  info map <string, string>,
  attributes map <string, string>,
  is_reference_block boolean COMMENT 'Used in GVCF for non-variant sites')
PARTITIONED BY (bucket(128, sample_name), chrom)
LOCATION '{URL}/'
TBLPROPERTIES (
  'table_type'='iceberg',
  'write_compression'='zstd'
);
```

Create tables using Python (without Athena)

The following Python code example shows how to create the tables without using Athena.

```
 import boto3
from pyiceberg.catalog import Catalog, load_catalog
from pyiceberg.schema import Schema
from pyiceberg.table import Table
from pyiceberg.table.sorting import SortOrder, SortField, SortDirection, NullOrder
```

```
from pyiceberg.partitioning import PartitionSpec, PartitionField
from pyiceberg.transforms import IdentityTransform, BucketTransform
from pyiceberg.types import (
    NestedField,
    StringType,
    LongType,
    DoubleType,
    MapType,
    BooleanType,
    ListType
)


def load_s3_tables_catalog(bucket_arn: str) -> Catalog:
    session = boto3.session.Session()
    region = session.region_name or 'us-east-1'

    catalog_config = {
        "type": "rest",
        "warehouse": bucket_arn,
        "uri": f"https://s3tables.{region}.amazonaws.com/iceberg",
        "rest.sigv4-enabled": "true",
        "rest.signing-name": "s3tables",
        "rest.signing-region": region
    }

    return load_catalog("s3tables", **catalog_config)


def create_namespace(catalog: Catalog, namespace: str) -> None:
    try:
        catalog.create_namespace(namespace)
        print(f"Created namespace: {namespace}")
    except Exception as e:
        if "already exists" in str(e):
            print(f"Namespace {namespace} already exists.")
        else:
            raise e


def create_table(catalog: Catalog, namespace: str, table_name: str, schema: Schema,
                 partition_spec: PartitionSpec = None, sort_order: SortOrder = None) ->
 Table:
    if catalog.table_exists(f"{namespace}.{table_name}"):
```

```
        print(f"Table {namespace}.{table_name} already exists.")
        return catalog.load_table(f"{namespace}.{table_name}")

    create_table_args = {
        "identifier": f"{namespace}.{table_name}",
        "schema": schema,
        "properties": {"format-version": "2"}
    }

    if partition_spec is not None:
        create_table_args["partition_spec"] = partition_spec
    if sort_order is not None:
        create_table_args["sort_order"] = sort_order

    table = catalog.create_table(**create_table_args)
    print(f"Created table: {namespace}.{table_name}")
    return table


def main(bucket_arn: str, namespace: str, table_name: str):
    # Schema definition
    genomic_variants_schema = Schema(
        NestedField(1, "sample_name", StringType(), required=True),
        NestedField(2, "variant_name", StringType(), required=True),
        NestedField(3, "chrom", StringType(), required=True),
        NestedField(4, "pos", LongType(), required=True),
        NestedField(5, "ref", StringType(), required=True),
        NestedField(6, "alt", ListType(element_id=1000, element_type=StringType(),
 element_required=True), required=True),
        NestedField(7, "qual", DoubleType()),
        NestedField(8, "filter", StringType()),
        NestedField(9, "genotype", StringType()),
        NestedField(10, "info", MapType(key_type=StringType(), key_id=1001,
 value_type=StringType(), value_id=1002)),
        NestedField(11, "attributes", MapType(key_type=StringType(), key_id=2001,
 value_type=StringType(), value_id=2002)),
        NestedField(12, "is_reference_block", BooleanType()),
        identifier_field_ids=[1, 2, 3, 4]
    )

    # Partition and sort specifications
    partition_spec = PartitionSpec(
        PartitionField(source_id=1, field_id=1001, transform=BucketTransform(128),
 name="sample_bucket"),
```

```
        PartitionField(source_id=3, field_id=1002, transform=IdentityTransform(),
 name="chrom")
    )

    sort_order = SortOrder(
        SortField(source_id=3, transform=IdentityTransform(),
 direction=SortDirection.ASC, null_order=NullOrder.NULLS_LAST),
        SortField(source_id=4, transform=IdentityTransform(),
 direction=SortDirection.ASC, null_order=NullOrder.NULLS_LAST)
    )

    # Connect to catalog and create table
    catalog = load_s3_tables_catalog(bucket_arn)
    create_namespace(catalog, namespace)
    table = create_table(catalog, namespace, table_name, genomic_variants_schema,
 partition_spec, sort_order)

    return table


if __name__ == "__main__":
    bucket_arn = 'arn:aws:s3tables:<REGION>:<ACCOUNT_ID>:bucket/<TABLE_BUCKET_NAME'
    namespace = "variant_db"
    table_name = "genomic_variants"

    main(bucket_arn, namespace, table_name)
```

Setting up HealthOmics

To set up AWS HealthOmics, sign up for an AWS account, create an administrative user, and securely manage access for additional users.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Create IAM permissions for HealthOmics
- Connect with external code repositories
- Using Amazon Q CLI with 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 or text message 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*.

Create IAM permissions for HealthOmics

To use HealthOmics, configure the following IAM permissions:

- IAM identity-based policies for users in your account to access HealthOmics.
- An IAM service role for HealthOmics to access resources on your behalf.
- Permissions in other services (such as Lake Formation and Amazon ECR) for your users and the HealthOmics service to access resources.

For more information about configuring IAM permissions for HealthOmics, see IAM permissions for HealthOmics.

Connect with external code repositories

With AWS HealthOmics, you can manage your workflows using Git-based repositories through AWS CodeConnections. HealthOmics uses this connection to access your source code repositories.

Before working with external code repositories, follow the Setting up connections guide to start working with AWS CodeConnections. Verify that you have created the proper IAM policies and permissions for your AWS account. For a list of supported Git providers and more information, see What third-party providers can I create connections for?.

Create a connection

To create a connection with your preferred repository provider, follow the Create a connection tutorial.

Using Amazon Q CLI with HealthOmics

Amazon Q CLI provides natural language interactions with AWS HealthOmics, allowing you to perform complex genomic workflows and analysis tasks using conversational commands. To use Amazon Q CLI, be sure to configure IAM permissions for HealthOmics and other services (such as CloudWatch, Amazon ECR, or Amazon S3) for Amazon Q to access their resources.

The HealthOmics Agentic generative AI tutorial provides a step-by-step guidance for configuring context files and enabling Amazon Q CLI to create, run, and optimize your AWS HealthOmics workflows.

Getting started with HealthOmics

To get started with HealthOmics, ensure that you have properly set up your IAM permissions and roles for HealthOmics.

Using a Ready2Run workflow in the HealthOmics console

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 you do not need to create your own data.

1. Open the HealthOmics console.
2. Select the navigation pane (≡) in the top left, and select **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, in the top right of the page select **Start run**.
5. In the **Specify run details** page, enter a run name.
6. Enter or select an Amazon S3 location for the run output.
7. For **Run metadata retention mode**, choose whether to retain or remove runmeta data.
8. In the **Service role** panel, choose **Create and use a new service role**.
9. Choose **Next**.
10. On the **Add parameter values** page, choose **Run workflow with Ready2Run test data**.
11. Choose **Next**.
12. Review your inputs, then choose **Start run**.

Example prompts for Amazon Q CLI

Amazon Q CLI can run genomic workflows and analysis tasks in AWS HealthOmics using natural language commands. The following example prompts allow you to create workflows, manage runs, and analyze genomic data. For more information and example prompts for HealthOmics, see the HealthOmics Agentic generative AI tutorial on GitHub.

- "Create a WDL 1.1 workflow file as `main.wdl` that will run on HealthOmics. The workflow will take a reference genome as an input and pairs of fastq files. It will index the reference genome using BWA and then map each pair of fastq files to the reference. Finally merge each mapped BAM to a single BAM file and output this file and it's bai index."
- "Package the workflow and create it in HealthOmics"
- "Update the inputs.json file to use real files from my Amazon S3 bucket `omics-my-bucket-with-genome-data`" (Provide a specific Amazon S3 bucket location, or let Amazon Q explore)
- "Find suitable containers in my Amazon ECR repositories and update inputs.json to use these"
- "Find or create a suitable IAM role to use when running the workflow"
- "Create a run cache for my workflow"
- "Run the workflow in HealthOmics"
- "Check the status of the run"

Warning

When working with Amazon Q CLI, review all generated content and proposed actions before proceeding. Provide feedback to improve response quality and to match your workflow's requirements. For more information, see Security considerations and best practices for Amazon Q.

Private workflows in HealthOmics

Use *Private workflows* when you want to create your own workflow definition. The workflow definition specifies information about the workflow and defines the workflow tasks. A *run* is a single invocation of a workflow, and a *task* is a single process within the run.

HealthOmics supports workflow definitions that you create in Workflow Description Language (WDL), Common Workflow Language (CWL), or Nextflow.

HealthOmics workflows provide the following optional features:

- **Run groups** – You can add private workflows to a run group to control compute usage. A *run group* is a collection of 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.
- **Call caching** – You can use a call cache to save and reuse task outputs, which results in shorter run durations and compute cost savings.
- **Sharing workflows** – You can share your private workflows with other AWS accounts in the same Region.
- **Workflow versions** – You can create versions of a private workflow. Workflow versioning provides the ability for users to choose when to start using updated functionality. Workflow versions are immutable and provide the same level of data provenance as workflows.

For information about configuring IAM permissions for workflows, see IAM permissions for HealthOmics.

For full examples of how to use HealthOmics private workflows, see HealthOmics Github tutorials or the AWS workshop end to end tutorial for HealthOmics.

Topics

- Creating private workflows in HealthOmics
- Workflow versioning in HealthOmics
- Using HealthOmics runs
- Using HealthOmics run groups
- Call caching for HealthOmics runs

- Sharing HealthOmics workflows

Creating private workflows in HealthOmics

Private workflows depend on a variety of resources that you create and configure before creating the workflow:

- **Workflow definition file:** A workflow definition file written in WDL, Nextflow, or CWL. The workflow definition specifies the inputs and outputs for runs that use the workflow. It also includes specifications for the runs and run tasks for your workflow, including compute and memory requirements. The workflow definition file must be in `.zip` format. For more information, see Workflow definition files.
  - You can use Amazon Q CLI to build and validate your workflow definition files in WDL, Nextflow, and CWL. For more information, see Example prompts for Amazon Q CLI and the HealthOmics Agentic generative AI tutorial on GitHub.
- **(Optional) Parameter template file:** A parameter template file written in JSON. Create the file to define the run parameters, or HealthOmics generates the parameter template for you. For more information, see Parameter template files for HealthOmics workflows.
- **Amazon ECR container images:** Create a private Amazon ECR repository for the workflow. Create container images in the private repository, or synchronize the contents of a supported upstream registry with your Amazon ECR private repository.
- **(Optional) Sentieon licenses:** Request a Sentieon license to use the Sentieon software in private workflows.

Optionally, you can run a linter on the workflow definition before or after you create the workflow. The **linter** topic describes the linters available in HealthOmics.

Topics

- HealthOmics workflow integration with Git-based repositories
- Workflow definition files in HealthOmics
- Parameter template files for HealthOmics workflows
- Container images for private workflows
- HealthOmics Workflow README files
- Requesting Sentieon licenses for private workflows

- Workflow linters in HealthOmics
- HealthOmics workflow operations

HealthOmics workflow integration with Git-based repositories

When you create a workflow (or a workflow version), you provide a workflow definition to specify information about the workflow, runs, and tasks. HealthOmics can retrieve the workflow definition as a .zip archive (stored locally or in an Amazon S3 bucket), or from a supported Git-based repository.

The HealthOmics integration with Git-based repositories enables the following capabilities:

- Direct workflow creation from public, private, and self-managed instances.
- Integration of workflow README files and parameter templates from repositories.
- Support for GitHub, GitLab, and Bitbucket repositories.

By using a Git-based repository, you avoid the manual steps of downloading workflow definition files and input parameter template files, creating a .zip archive, and then staging the archive to S3. This simplifies workflow creation for scenarios such as the following examples:

1. You want to get started quickly using a common open source workflow, such as nf-core. HealthOmics automatically retrieves all workflow definition and input parameter template files from the nf-core repository on GitHub and uses these files to create your new workflow.
2. You are using a public workflow from GitHub, and some new updates become available. You can easily create a new HealthOmics workflow version using the updated workflow definition on GitHub as the source. Users of your workflow can choose between the original workflow or the new workflow version that you created.
3. Your team is building a proprietary pipeline that is not public. You keep your code on a private git repository and use this workflow definition for your HealthOmics workflows. The team updates the workflow definition frequently as part of an iterative workflow development lifecycle. You can easily create new workflow versions as required from your private repository.

Topics

- Supported Git-based repositories
- Configure connections to external code repositories

- Accessing self managed repositories
- Quotas related to external code repositories
- Required IAM permissions

Supported Git-based repositories

HealthOmics supports public and private repositories for the following Git-based providers:

- GitHub
- GitLab
- Bitbucket

HealthOmics supports self-managed repositories for the following Git-based providers:

- GitHubEnterpriseServer
- GitLabSelfManaged

HealthOmics supports use of cross-account connections for GitHub, GitLab, and Bitbucket. Set up shared permissions through the AWS Resource Access Manager. For an example, see Shared connections in the *CodePipeline user guide*.

Configure connections to external code repositories

Connect your workflows to Git-based repositories using AWS CodeConnection. HealthOmics uses this connection to access your source code repositories.

> **Note**
>
> The AWS CodeConnections service is not available in the il-central-1 region. For this region, configure service us-east-1 to create workflows or workflow versions from a repository.

Create a connection

Before you can create connections, follow the instructions in Setting up connections in the *Developer Console Tools User Guide*.

To create a connection, follow the instructions in Create a connection in the *Developer Console Tools User Guide*.

Configure authorization for the connection

You must authorize the connection using the provider's OAuth flow. Make sure that the connection status is `AVAILABLE` before you use it.

For examples, see the blog post How To Create an AWS HealthOmics Workflows from Content in Git.

Accessing self managed repositories

To set up connections to a GitLab self-managed repository, use an admin Personal Access Token when creating a host. The subsequent connection creation accesses Oauth with the customer's account.

The following example sets up a connection to a GitLab self-managed repository:

1. Set up access to the Personal Access Token of an admin user.

   To set up a PAT in a GitLab self managed repository, see Personal access tokens in *GitLab Docs*.
2. Create a host

   a. Navigate to **CodePipeline>Settings>Connections**.
   b. Choose the **Hosts** tab and then choose **Create Host**.
   c. Configure the following fields:

      - Enter a name of the host
      - For provider type, choose **GitLab Self Managed**
      - Enter the **Host URL**
      - Enter the VPC information if the host is defined in a VPC

   d. Choose **Create Host**, which creates the host in PENDING state.
   e. To complete the set up, choose **Set up Host**.
   f. Enter the Personal Access Token (PAT) of an Admin user, then choose **Continue**.
3. Create the connection

   a. Choose **Create Connections** on the **Connections** tab.
   b. For provider type, select **GitLab self-managed**.

c. Under **Connection Settings>Enter Connection Name**, enter the Host URL that you previously created.

d. If your GitLab self-managed instance is only accessible via a VPC, configure the VPC details.

e. Choose **Update Pending Connection**. The modal window re-directs you to the GitLab login page.

f. Enter the username and password for the customer account and complete the authorization process.

g. For first time setup, choose **Authorize AWS Connector for Gitlab Self Managed**.

Quotas related to external code repositories

For HealthOmics integration with external code repositories, there is a maximum size for a repository, each repository file, and each README file. For details, see HealthOmics workflow fixed size quotas.

Required IAM permissions

Add the following actions to your identity-based IAM policy:

```
"codeconnections:CreateConnection",
"codeconnections:GetConnection",
"codeconnections:GetHost",
"codeconnections:ListConnections",
"codeconnections:UseConnection"
```

Workflow definition files in HealthOmics

You use a workflow definition to specify information about the workflow, runs, and the tasks in the runs. You create workflow definitions in one or more files using a workflow definition language. HealthOmics supports workflow definitions written in WDL, Nextflow, or CWL.

HealthOmics supports the following choices for WDL workflow definitions:

- WDL – Provides a spec-conformant WDL engine.
- WDL lenient – Designed to handle workflows migrated from Cromwell. It supports customer Cromwell directives and some non-conformant logic. For details, see Implicit type conversion in WDL lenient.

For information about each of the workflow languages, see the language-specific detailed sections below.

You specify the following types of information in the workflow definition:

- **Language version** – The language and version of the workflow definition.
- **Compute and memory** – The compute and memory requirements for tasks in the workflow.
- **Inputs** – Location of the inputs to the workflow tasks. For more information, see HealthOmics run inputs.
- **Outputs** – Location to save the outputs that the tasks generate.
- **Task resources** – Compute and memory requirements for each task.
- **Accelerators** – other resources that the tasks require, such as accelerators.

Topics

- HealthOmics workflow definition requirements
- Version support for HealthOmics workflow definition languages
- Compute and memory requirements for HealthOmics tasks
- Task outputs in a HealthOmics workflow definition
- Task resources in a HealthOmics workflow definition
- Task accelerators in a HealthOmics workflow definition
- WDL workflow definition specifics
- Nextflow workflow definition specifics
- CWL workflow definition specifics
- Example workflow definitions

HealthOmics workflow definition requirements

The HealthOmics workflow definition files must meet the following requirements:

- Tasks must define input/output parameters, Amazon ECR container repositories, and runtime specifications such as memory or CPU allocation.
- Verify that your IAM roles have the required permissions.
  - Your workflow has access to input data from AWS resources, such as Amazon S3.

  - Your workflow has access to external repository services when needed.
- Declare the output files in the workflow definition. 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.
- HealthOmics storage workflow inputs must be in `ACTIVE` status. HealthOmics won't import inputs with an `ARCHIVED` status, causing the workflow to fail. For information about Amazon S3 object inputs, see HealthOmics run inputs.
- A **main** location of the workflow is optional if your ZIP archive contains either a single workflow definition or a file named 'main'.
  - Example path: `workflow-definition/main-file.wdl`
- Before you create a workflow from Amazon S3 or your local drive, create a zip archive of the workflow definition files and any dependencies, such as subworkflows.
- We recommend that you declare Amazon ECR containers in the workflow as input parameters for validation of the Amazon ECR permissions.

Additional Nextflow considerations:

- **/bin**

  Nextflow workflow definitions may include a /bin folder with executable scripts. This path has read-only plus executable access to tasks. Tasks that rely on these scripts should use a container built with the appropriate script interpreters. Best practice is to call the interpreter directly. For example:

```
process my_bin_task {
   ...
   script:
      """
      python3 my_python_script.py
      """
}
```

- **includeConfig**

  Nextflow-based workflow definitions can include nextflow.config files that help to abstract parameter definitions or process resource profiles. To support development and execution of

Nextflow pipelines on multiple environments, use a HealthOmics-specific configuration that you add to the global config using the includeConfig directive. To maintain portability, configure the workflow to include the file only when running on HealthOmics by using the following code:

```
// at the end of the nextflow.config file
if ("$AWS_WORKFLOW_RUN") {
    includeConfig 'conf/omics.config'
}
```

- **Reports**

  HealthOmics doesn't support engine-generated dag, trace, and execution reports. You can generate alternatives to the trace and execution reports using a combination of GetRun and GetRunTask API calls.

Additional CWL considerations:

- **Container image uri interpolation**

  HealthOmics allows the dockerPull property of the DockerRequirement to be an inline javascript expression. For example:

```
requirements:
  DockerRequirement:
    dockerPull: "$(inputs.container_image)"
```

  This allows you to specifying container image URIs as input parameters to the workflow.

- **Javascript expressions**

  Javascript expressions must be `strict mode` compliant.

- **Operation process**

  HealthOmics doesn't support CWL Operation processes.

Version support for HealthOmics workflow definition languages

HealthOmics supports workflow definition files written in Nextflow, WDL, or CWL. The following sections provide information about HealthOmics version support for these languages.

Topics

- WDL version support
- CWL version support
- Nextflow version support

WDL version support

HealthOmics supports versions 1.0, 1.1, and the development version of the WDL specification.

Every WDL document must include a version statement to specify which version (major and minor) of the specification it adheres to. For more information about versions, see WDL versioning

Versions 1.0 and 1.1 of the WDL specification do not support the `Directory` type. To use the `Directory` type for inputs or outputs, set the version to **development** in the first line of the file:

```
version development  # first line of .wdl file
     ... remainder of the file ...
```

CWL version support

HealthOmics supports versions 1.0, 1.1, and 1.2 of the CWL language.

You can specify the language version in the CWL workflow definition file. For more information about CWL, see the CWL user guide

Nextflow version support

HealthOmics supports three Nextflow stable versions. Nextflow typically releases a stable version every six months. HealthOmics doesn't support the monthly “edge” releases.

HealthOmics supports released features in each version, but not preview features.

Supported versions

HealthOmics supports the following Nextflow versions:

- Nextflow v22.04.01 DSL 1 and DSL 2
- Nextflow v23.10.0 DSL 2 (default)

- Nextflow v24.10.8 DSL 2

To migrate your workflow to the latest supported version (v24.10.8), follow the Nextflow upgrade guide.

There are some breaking changes when migrating from Nextflow v23 to v24, as described in the following sections of the Nextflow migration guide:

- Breaking changes in 24.04
- Breaking changes in 24.10

Detect and process Nextflow versions

HealthOmics detects the DSL version and Nextflow version that you specify. It automatically determines the best Nextflow version to run based on these inputs.

DSL version

HealthOmics detects the requested DSL version in your workflow definition file. For example, you can specify: `nextflow.enable.dsl=2`.

HealthOmics supports DSL 2 by default. It provides backwards compatibility with DSL 1, if specified in your workflow definition file.

- If you specify DSL 2, HealthOmics runs Nextflow v23.10.0, unless you specify Nextflow v22.04.0 or v24.10.8.
- If you specify DSL 1, HealthOmics runs Nextflow v22.04 DSL1 (the only supported version that runs DSL 1).
- If you don't specify a DSL version, or if HealthOmics can’t parse the DSL information for any reason (such as syntax errors in your workflow definition file), HealthOmics defaults to DSL 2 and runs Nextflow v23.10.0.
- To upgrade your workflow from DSL 1 to DSL 2 to take advantage of the latest Nextflow versions and software features, see Migrating from DSL 1.

Nextflow versions

HealthOmics detects the requested Nextflow version in the Nextflow configuration file (nextflow.config), if you provide this file. We recommend that you add the `nextflowVersion`

clause at the end of the file to avoid any unexpected overrides from included configs. For more information, see Nextflow configuration.

You can specify a Nextflow version or a range of versions using the following syntax:

```
// exact match
manifest.nextflowVersion = '1.2.3'

// 1.2 or later (excluding 2 and later)
manifest.nextflowVersion = '1.2+'

// 1.2 or later
manifest.nextflowVersion = '>=1.2'

// any version in the range 1.2 to 1.5
manifest.nextflowVersion = '>=1.2, <=1.5'

// use the "!" prefix to stop execution if the current version
// doesn't match the required version.
manifest.nextflowVersion = '!>=1.2'
```

HealthOmics processes the Nextflow version information as follows:

- If you use **=** to specify an exact version that HealthOmics supports, HealthOmics uses that version.
- If you use **!** to specify an exact version or a range of versions that are not supported, HealthOmics raises an exception and fails the run. Consider using this option if you want to be strict with version requests and fail quickly if the request includes unsupported versions.
- If you specify a range of versions, HealthOmics uses the latest supported version in that range, unless the range includes v24.10.8. In this case, HealthOmics gives preference to an earlier version. For example, if the range covers both v23.10.0 and v24.10.8, HealthOmics chooses v23.10.0.
- If there is no requested version, or if the requested versions aren't valid or can’t be parsed for any reason:
  - If you specified DSL 1, HealthOmics runs Nextflow v22.04.
  - Otherwise, HealthOmics runs Nextflow v23.10.0.

You can retrieve the following information about the Nextflow version that HealthOmics used for each run:

- The run logs contain information about the actual Nextflow version that HealthOmics used for the run.
- HealthOmics adds warnings in the run logs if there isn't a direct match with your requested version or if it needed to use a different version than you specified.
- The response to the **GetRun** API operation includes a field (`engineVersion`) with the actual Nextflow version that HealthOmics used for the run. For example:

```
"engineVersion":"22.04.0"
```

Compute and memory requirements for HealthOmics tasks

HealthOmics runs your private workflow tasks in an omics instance. HealthOmics provides a variety of instance types to accommodate different types of tasks. Each instance type has a fixed memory and vCPU configuration (and fixed GPU configuration for accelerated computing instance types). The cost of using an omics instance varies depending on the instance type. For details, see the HealthOmics Pricing page.

For tasks in a workflow, you specify the required memory and vCPUs in the workflow definition file. When a workflow task runs, HealthOmics allocates the smallest omics instance that accommodates the requested memory and vCPUs. For example, if a task needs 64 GiB of memory and 8 vCPUs, HealthOmics selects `omics.r.2xlarge`.

We recommend that you review the instance types and set your requested vCPUs and memory size to match the instance that best meets your needs. The task container uses the number of vCPUs and the memory size that you specify in your workflow definition file, even if the instance type has additional vCPUs and memory.

The following list contains additional information about vCPU and memory allocation:

- Container resource allocations are hard limits. If a task runs out of memory or attempts to use additional vCPUs , the task generates an error log and exits.
- If you don’t specify any compute or memory requirements, HealthOmics selects **omics.c.large** and defaults to a configuration with 1 vCPU and 1 GiB of memory.
- The minimum configuration that you can request is 1 vCPU and 1 GiB of memory.
- If you specify vCPUs, memory, or GPUs that exceeds the supported instance types, HealthOmics throws an error message and the workflow fails validations
- If you specify fractional units, HealthOmics rounds up to the nearest integer.

- HealthOmics reserves a small amount of memory (5%) for management and logging agents, so the full memory allocation might not always be available to the application in the task.
- HealthOmics matches instance types to fit the compute and memory requirements that you specify, and may use a mix of hardware generations. For this reason, there can be some minor variances in task run times for the same task.

These topics provide details about the instance types that HealthOmics supports.

Topics

- Standard instance types
- Compute-optimized instances
- Memory-optimized instances
- Accelerated-computing instances

> **Note**
>
> For standard, compute, and memory optimized instances, increase the instance bandwidth size if the instance requires a higher throughput. Amazon EC2 instances with fewer than 16 vCPUs (size 4xl and smaller) can experience throughput bursting. For more information on Amazon EC2 instance throughput, see Amazon EC2 available instance bandwidth.

Standard instance types

For standard instance types, the configurations aim for a balance of compute power and memory.

HealthOmics supports the 32xlarge and 48xlarge instances in these regions: US West (Oregon) and US East (N. Virginia).

Instance	Number of vCPUs	Memory
omics.m.large	2	8 GiB
omics.m.xlarge	4	16 GiB
omics.m.2xlarge	8	32 GiB

Instance	Number of vCPUs	Memory
omics.m.4xlarge	16	64 GiB
omics.m.8xlarge	32	128 GiB
omics.m.12xlarge	48	192 GiB
omics.m.16xlarge	64	256 GiB
omics.m.24xlarge	96	384 GiB
omics.m.32xlarge	128	512 GiB
omics.m.48xlarge	192	768 GiB

Compute-optimized instances

For compute-optimized instance types, the configurations have more compute power and less memory.

HealthOmics supports the 32xlarge and 48xlarge instances in these regions: US West (Oregon) and US East (N. Virginia).

Instance	Number of vCPUs	Memory
omics.c.large	2	4 GiB
omics.c.xlarge	4	8 GiB
omics.c.2xlarge	8	16 GiB
omics.c.4xlarge	16	32 GiB
omics.c.8xlarge	32	64 GiB
omics.c.12xlarge	48	96 GiB
omics.c.16xlarge	64	128 GiB

Instance	Number of vCPUs	Memory
omics.c.24xlarge	96	192 GiB
omics.c.32xlarge	128	256 GiB
omics.c.48xlarge	192	384 GiB

Memory-optimized instances

For memory-optimized instance types, the configurations have less compute power and more memory.

HealthOmics supports the 32xlarge and 48xlarge instances in these regions: US West (Oregon) and US East (N. Virginia).

Instance	Number of vCPUs	Memory
omics.r.large	2	16 GiB
omics.r.xlarge	4	32 GiB
omics.r.2xlarge	8	64 GiB
omics.r.4xlarge	16	128 GiB
omics.r.8xlarge	32	256 GiB
omics.r.12xlarge	48	384 GiB
omics.r.16xlarge	64	512 GiB
omics.r.24xlarge	96	768 GiB
omics.r.32xlarge	128	1024 GiB
omics.r.48xlarge	192	1536 GiB

Accelerated-computing instances

You can optionally specify GPU resources for each task in a workflow, so that HealthOmics allocates an accelerated-computing instance for the task. For information on how to specify the GPU information in the workflow definition file, see Task accelerators in a HealthOmics workflow definition.

If you specify a GPU that supports multiple instance types, HealthOmics selects the instance type based on availability. If both instance types are available, HealthOmics gives preference to the lower cost instance.

G4 instances aren't supported in the Israel (Tel Aviv) Region. G5 instances aren't support in the Asia Pacific (Singapore) Region.

Topics

- G6 and G6e instance types
- G4 and G5 instances

G6 and G6e instance types

HealthOmics supports the following G6 accelerated-computing instance configurations. All omics.g6 instances use Nvidia L4 or Nvidia L4 A10G GPUs.

HealthOmics supports the G6 and G6e instances in these regions: US West (Oregon) and US East (N. Virginia).

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory	
omics.g6.xlarge	4	16 GiB	1	24 GiB	
omics.g6.2xlarge	8	32 GiB	1	24 GiB	
omics.g6.4xlarge	16	64 GiB	1	24 GiB	

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory
omics.g6.8xlarge	32	128 GiB	1	24 GiB
omics.g6.12xlarge	48	192 GiB	4	96 GiB
omics.g6.16xlarge	64	256 GiB	1	24 GiB
omics.g6.24xlarge	96	384 GiB	4	96 GiB

All omics.g6e instances use Nvidia L40s GPUs.

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory
omics.g6e.xlarge	4	32 GiB	1	48 GiB
omics.g6e.2xlarge	8	64 GiB	1	48 GiB
omics.g6e.4xlarge	16	128 GiB	1	48 GiB
omics.g6e.8xlarge	32	256 GiB	1	48 GiB
omics.g6e.12xlarge	48	384 GiB	4	192 GiB
omics.g6e.16xlarge	64	512 GiB	1	48 GiB

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory	
omics.g6e .24xlarge	96	768 GiB	4	192 GiB	

G4 and G5 instances

HealthOmics supports the following G4 and G5 accelerated-computing instance configurations.

All omics.g5 instances use Nvidia L4 A10G, Nvidia Tesla A10G, or Nvidia Tesla T4 A10G GPUs.

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory	
omics.g5. xlarge	4	16 GiB	1	24 GiB	
omics.g5. 2xlarge	8	32 GiB	1	24 GiB	
omics.g5. 4xlarge	16	64 GiB	1	24 GiB	
omics.g5. 8xlarge	32	128 GiB	1	24 GiB	
omics.g5. 12xlarge	48	192 GiB	4	96 GiB	
omics.g5. 16xlarge	64	256 GiB	1	24 GiB	
omics.g5. 24xlarge	96	384 GiB	4	96 GiB	

All omics.g4dn instances use Nvidia Tesla T4 or Nvidia Tesla T4 A10G GPUs.

Instance	Number of vCPUs	Memory	Number of GPUs	GPU memory
omics.g4dn.xlarge	4	16 GiB	1	16 GiB
omics.g4dn.2xlarge	8	32 GiB	1	16 GiB
omics.g4dn.4xlarge	16	64 GiB	1	16 GiB
omics.g4dn.8xlarge	32	128 GiB	1	16 GiB
omics.g4dn.12xlarge	48	192 GiB	4	64 GiB
omics.g4dn.16xlarge	64	256 GiB	1	24 GiB

Task outputs in a HealthOmics workflow definition

You specify task outputs in the workflow definition. By default, HealthOmics discards all intermediate task files when the workflow completes. To export an intermediate file, you define it as an output.

If you use call caching, HealthOmics saves task outputs to the cache, including any intermediate files that you define as outputs.

The following topics include task definition examples for each of the workflow definition languages.

Topics

- Task outputs for WDL
- Task outputs for Nextflow
- Task outputs for CWL

Task outputs for WDL

For workflow definitions written in WDL, define your outputs in the top level workflow **outputs** section.

HealthOmics

Topics

- Task output for STDOUT
- Task output for STDERR
- Task output to a file
- Task output to an array of files

Task output for STDOUT

This example creates a task named `SayHello` that echoes the STDOUT content to the task output file. The WDL **stdout** function captures the STDOUT content (in this example, the input string **Hello World!**) in file **stdout_file**.

Because HealthOmics creates logs for all STDOUT content, the output also appears in CloudWatch Logs, along with other STDERR logging information for the task.

```
version 1.0
 workflow HelloWorld {
    input {
        String message = "Hello, World!"
        String ubuntu_container = "123456789012.dkr.ecr.us-east-1.amazonaws.com/
dockerhub/library/ubuntu:20.04"
    }

    call SayHello {
        input:
            message = message,
            container = ubuntu_container
    }

    output {
        File stdout_file = SayHello.stdout_file
    }
}
```

```
task SayHello {
    input {
        String message
        String container
    }

    command <<<
        echo "~{message}"
        echo "Current date: $(date)"
        echo "This message was printed to STDOUT"
    >>>

    runtime {
        docker: container
        cpu: 1
        memory: "2 GB"
    }

    output {
        File stdout_file = stdout()
    }
}
```

Task output for STDERR

This example creates a task named `SayHello` that echoes the STDERR content to the task output file. The WDL **stderr** function captures the STDERR content (in this example, the input string **Hello World!**) in file **stderr_file**.

Because HealthOmics creates logs for all STDERR content, the output will appear in CloudWatch Logs, along with other STDERR logging information for the task.

```
version 1.0
 workflow HelloWorld {
    input {
        String message = "Hello, World!"
        String ubuntu_container = "123456789012.dkr.ecr.us-east-1.amazonaws.com/
dockerhub/library/ubuntu:20.04"
    }

    call SayHello {
        input:
            message = message,
```

```
            container = ubuntu_container
    }

    output {
        File stderr_file = SayHello.stderr_file
    }
}

task SayHello {
    input {
        String message
        String container
    }

    command <<<
        echo "~{message}" >&2
        echo "Current date: $(date)" >&2
        echo "This message was printed to STDERR" >&2
    >>>

    runtime {
        docker: container
        cpu: 1
        memory: "2 GB"
    }

    output {
        File stderr_file = stderr()
    }
}
```

Task output to a file

In this example, the SayHello task creates two files (message.txt and info.txt) and explicitly declares these files as the named outputs (message_file and info_file).

```
version 1.0
workflow HelloWorld {
    input {
        String message = "Hello, World!"
        String ubuntu_container = "123456789012.dkr.ecr.us-east-1.amazonaws.com/
dockerhub/library/ubuntu:20.04"
    }
```

```
    call SayHello {
        input:
            message = message,
            container = ubuntu_container
    }

    output {
        File message_file = SayHello.message_file
        File info_file = SayHello.info_file
    }
}

task SayHello {
    input {
        String message
        String container
    }

    command <<<
        # Create message file
        echo "~{message}" > message.txt

        # Create info file with date and additional information
        echo "Current date: $(date)" > info.txt
        echo "This message was saved to a file" >> info.txt
    >>>

    runtime {
        docker: container
        cpu: 1
        memory: "2 GB"
    }

    output {
        File message_file = "message.txt"
        File info_file = "info.txt"
    }
}
```

Task output to an array of files

In this example, the `GenerateGreetings` task generates an array of files as the task output. The task dynamically generates one greeting file for each member of the input array `names`. Because the file names are not known until runtime, the output definition uses the WDL glob() function to output all files that match the pattern `*_greeting.txt`.

```
version 1.0
 workflow HelloArray {
    input {
        Array[String] names = ["World", "Friend", "Developer"]
        String ubuntu_container = "123456789012.dkr.ecr.us-east-1.amazonaws.com/
dockerhub/library/ubuntu:20.04"
    }

    call GenerateGreetings {
        input:
            names = names,
            container = ubuntu_container
    }

    output {
        Array[File] greeting_files = GenerateGreetings.greeting_files
    }
}

task GenerateGreetings {
    input {
        Array[String] names
        String container
    }

    command  <<<
        # Create a greeting file for each name
        for name in ~{sep=" " names}; do
            echo "Hello, $name!" > ${name}_greeting.txt
        done
    >>>

    runtime {
        docker: container
        cpu: 1
        memory: "2 GB"
```

```
    }

    output {
        Array[File] greeting_files = glob("*_greeting.txt")
    }
}
```

Task outputs for Nextflow

For workflow definitions written in Nextflow, define a **publishDir** directive to export task content to your output Amazon S3 bucket. Set the **publishDir** value to `/mnt/workflow/pubdir`.

For HealthOmics to export files to Amazon S3, the files must be in this directory.

If a task produces a group of output files for use as inputs to a subsequent task, we recommend that you group these files in a directory and emit the directory as a task output. Enumerating each individual file can result in an I/O bottleneck in the underlying file system. For example:

```
process my_task {
      ...
      // recommended
      output "output-folder/", emit: output

      // not recommended
      // output "output-folder/**", emit: output
      ...
  }
```

Task outputs for CWL

For workflow definitions written in CWL, you can specify the task outputs using `CommandLineTool` tasks. The following sections show examples of `CommandLineTool` tasks that define different types of outputs.

Topics

- Task output for STDOUT
- Task output for STDERR
- Task output to a file
- Task output to an array of files

Task output for STDOUT

This example creates a `CommandLineTool` task that echoes the STDOUT content to a text output file named **output.txt**. For example, if you provide the following input, the resulting task output is **Hello World!** in the **output.txt** file.

```
{
    "message": "Hello World!"
}
```

The `outputs` directive specifies that the output name is **example_out** and it's type is `stdout`. For a downstream task to consume the output of this task, it would refer to the output as `example_out`.

Because HealthOmics creates logs for all STDERR and STDOUT content, the output also appears in CloudWatch Logs, along with other STDERR logging information for the task.

```
cwlVersion: v1.2
class: CommandLineTool
baseCommand: echo
stdout: output.txt
inputs:
  message:
    type: string
    inputBinding:
      position: 1
outputs:
  example_out:
    type: stdout

requirements:
    DockerRequirement:
        dockerPull: 123456789012.dkr.ecr.us-east-1.amazonaws.com/dockerhub/library/
ubuntu:20.04
    ResourceRequirement:
        ramMin: 2048
        coresMin: 1
```

Task output for STDERR

This example creates a `CommandLineTool` task that echoes the STDERR content to a text output file named **stderr.txt**. The task modifies the `baseCommand` so that `echo` writes to STDERR (instead of STDOUT).

The `outputs` directive specifies that the output name is **stderr_out** and it's type is `stderr`.

Because HealthOmics creates logs for all STDERR and STDOUT content, the output will appear in CloudWatch Logs, along with other STDERR logging information for the task.

```
cwlVersion: v1.2
class: CommandLineTool
baseCommand: [bash, -c]
stderr: stderr.txt
inputs:
  message:
    type: string
    inputBinding:
      position: 1
      shellQuote: true
      valueFrom: "echo $(self) >&2"
outputs:
  stderr_out:
    type: stderr

requirements:
    DockerRequirement:
        dockerPull: 123456789012.dkr.ecr.us-east-1.amazonaws.com/dockerhub/library/
ubuntu:20.04
    ResourceRequirement:
        ramMin: 2048
        coresMin: 1
```

Task output to a file

This example creates a `CommandLineTool` task that creates a compressed tar archive from the input files. You provide the name of the archive as an input parameter (archive_name).

The **outputs** directive specifies that the `archive_file` output type is `File`, and it uses a reference to the input parameter `archive_name` to bind to the output file.

```
cwlVersion: v1.2
```

```
class: CommandLineTool
baseCommand: [tar, cfz]
inputs:
  archive_name:
    type: string
    inputBinding:
      position: 1
  input_files:
    type: File[]
    inputBinding:
      position: 2

outputs:
  archive_file:
    type: File
    outputBinding:
      glob: "$(inputs.archive_name)"

requirements:
    DockerRequirement:
        dockerPull: 123456789012.dkr.ecr.us-east-1.amazonaws.com/dockerhub/library/
ubuntu:20.04
    ResourceRequirement:
        ramMin: 2048
        coresMin: 1
```

Task output to an array of files

In this example, the `CommandLineTool` task creates an array of files using the `touch` command. The command uses the strings in the `files-to-create` input parameter to name the files. The command outputs an array of files. The array includes any files in the working directory that match the `glob` pattern. This example uses a wildcard pattern ("*") that matches all files.

```
cwlVersion: v1.2
class: CommandLineTool
baseCommand: touch
inputs:
  files-to-create:
    type:
      type: array
      items: string
    inputBinding:
      position: 1
```

```
outputs:
  output-files:
    type:
      type: array
      items: File
    outputBinding:
      glob: "*"

requirements:
    DockerRequirement:
        dockerPull: 123456789012.dkr.ecr.us-east-1.amazonaws.com/dockerhub/library/
ubuntu:20.04
    ResourceRequirement:
        ramMin: 2048
        coresMin: 1
```

Task resources in a HealthOmics workflow definition

In the workflow definition, define the following for each task:

- The container image for task. For more information, see Container images for private workflows.
- The number of CPUs and memory required for the task. For more information, see Compute and memory requirements for HealthOmics tasks.

HealthOmics ignores any per-task storage specifications. HealthOmics provides run storage that all tasks in the run can access. For more information, see Run storage types in HealthOmics workflows.

WDL

```
task my_task {
   runtime {
      container: "<aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/<image-name>"
      cpu: 2
      memory: "4 GB"
   }
   ...
}
```

For a WDL workflow, HealthOmics attempts up to two retries for a task that fails because of service errors (API request returns a 5XX HTTP status code). For more information about task retries, see Task Retries.

You can opt out of the retry behavior by specifying the following configuration for the task in the WDL definition file:

```
runtime {
   preemptible: 0
}
```

NextFlow

```
process my_task {
   container "<aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/<image-name>"
   cpus 2
   memory "4 GiB"
   ...
}
```

CWL

```
cwlVersion: v1.2
class: CommandLineTool
requirements:
    DockerRequirement:
        dockerPull: "<aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/<image-
name>"
    ResourceRequirement:
        coresMax: 2
        ramMax: 4000 # specified in mebibytes
```

Task accelerators in a HealthOmics workflow definition

In the workflow definition, you can optionally specify the GPU accelerator-spec for a task. HealthOmics supports the following accelerator-spec values, along with the supported instance types:

Accelerator spec	Healthomics instance types				
nvidia-tesla-t4	G4				
nvidia-tesla-t4-a10g	G4 and G5				
nvidia-tesla-a10g	G5				
nvidia-l4-a10g	G5 and G6				
nvidia-l4	G6				
nvidia-l40s	G6e				

If you specify an accelerator type that supports multiple instance types, HealthOmics selects the instance type based on available capacity. If both instance types are available, HealthOmics gives preference to the lower cost instance.

For details about the instance types, see Accelerated-computing instances.

In the following example, the workflow definition specifies `nvidia-l4` as the accelarator:

WDL

```
task my_task {
 runtime {
    ...
    acceleratorCount: 1
    acceleratorType: "nvidia-l4"
 }
 ...
}
```

NextFlow

```
process my_task {
 ...
 accelerator 1, type: "nvidia-l4"
 ...
}
```

CWL

```
cwlVersion: v1.2
class: CommandLineTool
requirements:
  ...
  cwltool:CUDARequirement:
      cudaDeviceCountMin: 1
      cudaComputeCapability: "nvidia-l4"
      cudaVersionMin: "1.0"
```

WDL workflow definition specifics

The following topics provide details about types and directives available for WDL workflow definitions in HealthOmics.

Topics

- Implicit type conversion in WDL lenient
- Namespace definition in input.json
- Primitive types in WDL
- Complex types in WDL
- Directives in WDL
- Task metadata in WDL
- WDL workflow definition example

Implicit type conversion in WDL lenient

HealthOmics supports implicit type conversion in the input.json file and the workflow definition. To use implicit type casting, specify the workflow engine as WDL lenient when you create the

workflow. WDL lenient is designed to handle workflows migrated from Cromwell. It supports customer Cromwell directives and some non-conformant logic.

WDL lenient supports type conversion for the following items in the list of WDL's limited exceptions:

- Float to Int, where the coercion results in no loss of precision (such as 1.0 maps to 1).
- String to Int/Float, where the coercion results in no loss of precision.
- Map[W, X] to Array[Pair[Y, Z]], in the case where W is coercible to Y and X is coercible to Z.
- Array[Pair[W, X]] to Map[Y, Z], in the case where W is coercible to Y and X is coercible to Z (such as 1.0 maps to 1).

To use implicit type casting, specify the workflow engine as WDL_LENIENT when you create the workflow or workflow version.

In the console, the workflow engine parameter is named **Language**. In the API, the workflow engine parameter is named **engine**. For more information, see Create a private workflow or Create a workflow version.

Namespace definition in input.json

HealthOmics supports fully qualified variables in input.json. For example, if you declare two input variables named number1 and number2 in workflow **SumWorkflow**:

```
workflow SumWorkflow {
  input {
    Int number1
    Int number2
  }
}
```

You can use them as fully qualified variables in input.json:

```
{
    "SumWorkflow.number1": 15,
    "SumWorkflow.number2": 27
}
```

Primitive types in WDL

The following table shows how inputs in WDL map to the matching primitive types. HealthOmics provides limited support for type coercion, so we recommend that you set explicit types.

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.
`File`	`string`	`File f`	`"f": "s3://amzn-s3-demo-bucket1/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://`,

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
				`https://`, and `ftp://`). The URI must specify an object. It cannot be a directory meaning it can not end with a `/`.
`Directory`	`string`	`Directory d`	`"d": "s3://bucket/path/"`	The `Directory` type isn't included in WDL 1.0 or 1.1, so you will need to add `version development` 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 `Directory` should only contain files related to the workflow.

Complex types in WDL

The following table show how inputs in WDL map to the matching complex JSON types. 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": "0", "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 SampleBamAndIndex { String sample_name File bam File bam_index`	`"b_and_i": { "sample_name": "NA12878", "bam": "s3://amzn-s3-demo-bucket1/`	The names of the struct members must exactly match the names of the JSON object keys. Each value must use the JSON format of

WDL type	JSON type	Example WDL	Example JSON key and value	Notes
		} SampleBamAndIndex b_and_i	NA12878.bam", "bam_index": "s3://amzn-s3-demo-bucket1/NA12878.bam.bai" }	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.

Directives in WDL

HealthOmics supports the following directives in all WDL versions that HealthOmics supports.

Configure GPU resources

HealthOmics supports runtime attributes **acceleratorType** and **acceleratorCount** with all supported GPU instances. HealthOmics also supports aliases named **gpuType** and **gpuCount**, which have the same functionality as their accelerator counterparts. If the WDL definition contains both directives, HealthOmics uses the accelerator values.

The following example shows how to use these directives:

```
runtime {
    gpuCount: 2
    gpuType: "nvidia-tesla-t4"
}
```

Configure task retry for service errors

HealthOmics supports up to two retries for a task that failed because of service errors (5XX HTTP status codes). You can configure the maximum number of retries (1 or 2) and you can opt out of retries for service errors. By default, HealthOmics attempts a maximum of two retries.

The following example sets `preemptible` to opt out of retries for service errors:

```
{
  preemptible: 0
}
```

For more information about task retries in HealthOmics, see Task Retries.

Configure task retry for out of memory

HealthOmics supports retries for a task that failed because it ran out of memory (container exit code 137, 4XX HTTP status code). HealthOmics doubles the amount of memory for each retry attempt.

By default, HealthOmics doesn't retry for this type of failure. Use the `maxRetries` directive to specify the maximum number of retries.

The following example sets `maxRetries` to 3, so that HealthOmics attempts a maximum of four attempts to complete the task (the initial attempt plus three retries):

```
runtime {
    maxRetries: 3
}
```

> **Note**
>
> Task retry for out of memory requires GNU findutils 4.2.3+. The default HealthOmics image container includes this package. If you specify a custom image in your WDL definition, make sure that the image includes GNU findutils 4.2.3+.

Configure return codes

The **returnCodes** attribute provides a mechanism to specify a return code, or a set of return codes, that indicates a successful execution of a task. The WDL engine honors the return codes that you specify in the **runtime** section of the WDL definition, and sets the tasks status accordingly.

```
runtime {
    returnCodes: 1
}
```

HealthOmics also supports an alias named **continueOnReturnCode**, which has the same capabilities as **returnCodes**. If you specify both attributes, HealthOmics uses the **returnCodes** value.

Task metadata in WDL

HealthOmics supports the following metadata options for WDL tasks.

Disable task-level caching with the volatile attribute

The **volatile** attribute allows you to disable call caching for specific tasks in your WDL workflow. When a task is marked as volatile, it will always execute and never use cached results, even when caching is enabled for the run.

Add the **volatile** attribute to the **meta** section of your task definition:

```
task my_volatile_task {
    meta {
        volatile: true
    }

    input {
        String input_file
    }

    command {
        echo "Processing ${input_file}" > output.txt
    }

    output {
        File result = "output.txt"
    }
```

```
}
```

WDL workflow definition example

The following examples show 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.

The following example shows how to include the Amazon ECR container as a parameter. This allows HealthOmics to verify the access permissions to your container before it starts the run the run.

```
{
   ...
   "gotc_docker":"<account_id>.dkr.ecr.<region>.amazonaws.com/genomes-in-the-
cloud:2.4.7-1603303710"
}
```

The following example shows how to specify which files to use in your run, when the files are in an Amazon S3 bucket.

```
{
    "input_cram": "s3://amzn-s3-demo-bucket1/inputs/NA12878.cram",
    "ref_dict": "s3://amzn-s3-demo-bucket1/inputs/Homo_sapiens_assembly38.dict",
    "ref_fasta": "s3://amzn-s3-demo-bucket1/inputs/Homo_sapiens_assembly38.fasta",
    "ref_fasta_index": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket1/inputs/Homo_sapiens_assembly38.dict",
    "ref_fasta": "s3://amzn-s3-demo-bucket1/inputs/Homo_sapiens_assembly38.fasta",
    "ref_fasta_index": "s3://amzn-s3-demo-bucket1/inputs/
Homo_sapiens_assembly38.fasta.fai",
    "sample_name": "NA12878"
```

```
}
```

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

```
 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}"
    }
}
```

Nextflow workflow definition specifics

HealthOmics suppports Nextflow DSL1 and DSL2. For details, see Nextflow version support.

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.

Topics

- Use nf-schema and nf-validation plugins
- Specify storage URIs
- Nextflow directives
- Export task content

Use nf-schema and nf-validation plugins

> **Note**
>
> Summary of HealthOmics support for plugins:
>
> - v22.04 – no support for plugins
> - v23.10 – supports `nf-schema` and `nf-validation`
> - v24.10 – supports `nf-schema`

HealthOmics provides the following support for Nextflow plugins:

- For Nextflow v23.10, HealthOmics pre-installs the nf-validation@1.1.1 plugin.

- For Nextflow v23.10 and later, HealthOmics pre-installs the nf-schema@2.3.0 plugin.
- You cannot retrieve additional plugins during a workflow run. HealthOmics ignores any other plugin versions that you specify in the `nextflow.config` file.
- For Nextflow v24 and higher, `nf-schema` is the new version of the deprecated `nf-validation` plugin. For more information, see nf-schema in the Nextflow GitHub repository.

Specify storage URIs

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. For examples, see Amazon S3 input parameter formats.

HealthOmics partially supports the use of glob patterns in Amazon S3 URIs or HealthOmics Storage URIs. Use Glob patterns in the workflow definition for the creation of `path` or `file` channels. For the expected behavior and exact cases, see Nextflow Handling of Glob pattern in Amazon S3 inputs.

Nextflow directives

You configure Nextflow directives in the Nextflow config file or workflow definition. The following list shows the order of precedence that HealthOmics uses to apply configuration settings, from lowest to highest priority:

1. Global configuration in the config file.
2. Task section of the workflow definition.
3. Task-specific selectors in the config file.

Topics

- Task retry strategy using errorStrategy
- Task retry attempts using maxRetries
- Opt out of task retry using omicsRetryOn5xx
- Task duration using the time directive

Task retry strategy using `errorStrategy`

Use the `errorStrategy` directive to define the strategy for task errors. By default, when a task returns with an error indication (a non-zero exit status), the task stops and HealthOmics terminates the entire run. If you set `errorStrategy` to `retry`, HealthOmics attempts one retry of the failed task. To increase the number of retries, see Task retry attempts using `maxRetries`.

```
process {
    label 'my_label'
    errorStrategy 'retry'

    script:
    """
    your-command-here
    """
}
```

For information about how HealthOmics handles task retries during a run, see Task Retries.

Task retry attempts using `maxRetries`

By default, HealthOmics doesn't attempt any retries of a failed task, or attempts one retry if you configure `errorStrategy`. To increase the maximum number of retries, set `errorStrategy` to `retry` and configure the maximum number of retries using the `maxRetries` directive.

The following example sets the maximum number of retries to 3 in the global configuration.

```
process {
    errorStrategy = 'retry'
    maxRetries = 3
}
```

The following example shows how to set `maxRetries` in the task section of the workflow definition.

```
process myTask {
    label 'my_label'
    errorStrategy 'retry'
    maxRetries 3

    script:
```

```
    """
    your-command-here
    """
}
```

The following example shows how to specify task-specific configuration in the Nextflow config file, based on the name or label selectors.

```
process {
    withLabel: 'my_label' {
        errorStrategy = 'retry'
        maxRetries = 3
    }

    withName: 'myTask' {
        errorStrategy = 'retry'
        maxRetries = 3
    }
}
```

Opt out of task retry using `omicsRetryOn5xx`

For Nextflow v23 and v24, HealthOmics supports task retries if the task failed because of service errors (5XX HTTP status codes). By default, HealthOmics attempts up to two retries of a failed task.

You can configure `omicsRetryOn5xx` to opt out of task retry for service errors. For more information about task retry in HealthOmics, see Task Retries.

The following example configures `omicsRetryOn5xx` in the global configuration to opt out of task retry.

```
process {
    omicsRetryOn5xx = false
}
```

The following example shows how to configure `omicsRetryOn5xx` in the task section of the workflow definition.

```
process myTask {
    label 'my_label'
```

```
    omicsRetryOn5xx = false

    script:
    """
    your-command-here
    """
}
```

The following example shows how to set `omicsRetryOn5xx` as task-specific configuration in the Nextflow config file, based on the name or label selectors.

```
process {
    withLabel: 'my_label' {
        omicsRetryOn5xx = false
    }

    withName: 'myTask' {
        omicsRetryOn5xx = false
    }
}
```

Task duration using the `time` directive

HealthOmics provides an adjustable quota (see HealthOmics service quotas) to specify the maximum duration for a run. For Nextflow v23 and v24 workflows, you can also specify maximum task durations using the Nextflow `time` directive.

During new workflow development, setting maximum task duration helps you catch runaway tasks and long-running tasks.

For more information about the Nextflow time directive, see time directive in the Nextflow reference.

HealthOmics provides the following support for the Nextflow time directive:

1. HealthOmics supports 1 minute granularity for the time directive. You can specify a value between 60 seconds and the maximum run duration value.
2. If you enter a value less than 60, HealthOmics rounds it up to 60 seconds. For values above 60, HealthOmics rounds down to the nearest minute.
3. If the workflow supports retries for a task, HealthOmics retries the task if it times out.

4. If a task times out (or the last retry times out), HealthOmics cancels the task. This operation can have a duration of one to two minutes.
5. On task timeout, HealthOmics sets the run and task status to failed, and it cancels the other tasks in the run (for tasks in Starting, Pending, or Running status). HealthOmics exports the outputs from tasks that it completed before the timeout to your designated S3 output location.
6. Time that a task spends in pending status does not count toward the task duration.
7. If the run is part of a run group and the run group times out sooner than the task timer, the run and task transition to failed status.

Specify the timeout duration using one or more of the following units: `ms`, `s`, `m`,`h`, or `d`.

The following example shows how to specify global configuration in the Nextflow config file. It sets a global timeout of 1 hour and 30 minutes.

```
process {
    time = '1h30m'
}
```

The following example shows how to specify a time directive in the task section of the workflow definition. This example sets a timeout of 3 days, 5 hours, and 4 minutes. This value takes precedence over the global value in the config file, but doesn't take precedence over a task-specific time directive for `my_label` in the config file.

```
process myTask {
    label 'my_label'
    time '3d5h4m'

    script:
    """
    your-command-here
    """
}
```

The following example shows how to specify task-specific time directives in the Nextflow config file, based on the name or label selectors. This example sets a global task timeout value of 30 minutes. It sets a value of 2 hours for task `myTask` and sets a value of 3 hours for tasks with label `my_label`. For tasks that match the selector, these values take precedence over the global value and the value in the workflow definition.

```
process {
    time = '30m'

    withLabel: 'my_label' {
        time = '3h'
    }

    withName: 'myTask' {
        time = '2h'
    }
}
```

Export task content

For workflows written in Nextflow, 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
    """
}
```

CWL workflow definition specifics

Workflows written in Common Workflow Language, or CWL, offer similar functionality to workflows written in WDL and Nextflow. You can use Amazon S3 or HealthOmics storage URIs as input parameters.

If you define input in a secondaryFile in a sub workflow, add the same definition in the main workflow.

HealthOmics workflows don't support operation processes. To learn more about operations processes in CWL workflows, see the CWL documentation.

Best practice is to define a separate CWL workflow for each container that you use. We recommend that you don't hardcode the dockerPull entry with a fixed Amazon ECR URI.

Topics

- Convert CWL workflows to use HealthOmics
- Opt out of task retry using omicsRetryOn5xx
- Loop a workflow step
- Retry tasks with increased memory
- Examples

Convert CWL workflows to use HealthOmics

To convert an existing CWL workflow definition to use HealthOmics, make the following changes:

- Replace all Docker container URIs with Amazon ECR URIs.
- Make sure that all the workflow files are declared in the main workflow as input, and all variables are explicitly defined.
- Make sure that all JavaScript code is strict-mode complaint.

Opt out of task retry using `omicsRetryOn5xx`

HealthOmics supports task retries if the task failed because of service errors (5XX HTTP status codes). By default, HealthOmics attempts up to two retries of a failed task. For more information about task retry in HealthOmics, see Task Retries.

To opt out of task retry for service errors, configure the `omicsRetryOn5xx` directive in the workflow definition. You can define this directive under requirements or hints. We recommend adding the directive as a hint for portability.

```
requirements:
  ResourceRequirement:
    omicsRetryOn5xx: false

hints:
  ResourceRequirement:
    omicsRetryOn5xx: false
```

Requirements override hints. If a task implementation provides a resource requirement in hints that is also provided by requirements in an enclosing workflow, the enclosing requirements takes precedence.

If the same task requirement appears at different levels of the workflow, HealthOmics uses the most specific entry from `requirements` (or `hints`, if there are no entries in `requirements`). The following list shows the order of precedence that HealthOmics uses to apply configuration settings, from lowest to highest priority:

- Workflow level
- Step level
- Task section of the workflow definition

The following example shows how to configure the `omicsRetryOn5xx` directive at different levels of the workflow. In this example, the workflow-level requirement overrides the workflow level hints. The requirements configurations at the task and step levels override the hints configurations.

```
class: Workflow
# Workflow-level requirement and hint
requirements:
  ResourceRequirement:
    omicsRetryOn5xx: false

hints:
  ResourceRequirement:
    omicsRetryOn5xx: false  # The value in requirements overrides this value

steps:
  task_step:
    # Step-level requirement
    requirements:
      ResourceRequirement:
        omicsRetryOn5xx: false
    # Step-level hint
    hints:
      ResourceRequirement:
        omicsRetryOn5xx: false
    run:
      class: CommandLineTool
      # Task-level requirement
      requirements:
```

```
        ResourceRequirement:
          omicsRetryOn5xx: false
      # Task-level hint
      hints:
        ResourceRequirement:
          omicsRetryOn5xx: false
```

Loop a workflow step

HealthOmics supports looping a workflow step. You can use loops to run workflow steps repeatedly until a specified condition is met. This is useful for iterative processes where you need to repeat a task multiple times or until a certain result is achieved.

Note: Loop functionality requires CWL version 1.2 or later. Workflows using CWL versions earlier than 1.2 do not support loop operations.

To use loops in your CWL workflow, define a Loop requirement. The following example shows the loop requirement configuration:

```
requirements:
  - class: "http://commonwl.org/cwltool#Loop"
    loopWhen: $(inputs.counter < inputs.max)
    loop:
      counter:
        loopSource: result
        valueFrom: $(self)
    outputMethod: last
```

The `loopWhen` field controls when the loop terminates. In this example, the loop continues as long as the counter is less than the maximum value. The `loop` field defines how input parameters are updated between iterations. The `loopSource` specifies which output from the previous iteration feeds into the next iteration. The `outputMethod` field set to `last` returns only the final iteration's output.

Retry tasks with increased memory

HealthOmics supports automatic retry of out-of-memory task failures. When a task exits with code 137 (out-of-memory), HealthOmics creates a new task with increased memory allocation based on the specified multiplier.

> **Note**
>
> HealthOmics retries out-of-memory failures up to 3 times or until the memory allocation reaches 1536 GiB, whichever limit is reached first.

The following example shows how to configure out-of-memory retry:

```
hints:
  ResourceRequirement:
    ramMin: 4096
  http://arvados.org/cwl#OutOfMemoryRetry:
    memoryRetryMultiplier: 2.5
```

When a task fails due to out-of-memory, HealthOmics calculates the retry memory allocation using the formula: `previous_run_memory × memoryRetryMultiplier`. In the example above, if the task with 4096 MB of memory fails, the retry attempt uses 4096 × 2.5 = 10,240 MB of memory.

The `memoryRetryMultiplier` parameter controls how much additional memory to allocate for retry attempts:

- **Default value:** If you don't specify a value, it defaults to 2 (doubles the memory)
- **Valid range:** Must be a positive number greater than 1. Invalid values result in a 4XX validation error
- **Minimum effective value:** Values between 1 and `1.5` are automatically increased to `1.5` to ensure meaningful memory increases and prevent excessive retry attempts

Examples

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

inputs: []
outputs: []
```

Example workflow definitions

The following example shows the same workflow definition in WDL, Nextflow, and CWL.

WDL

```
version 1.1

task my_task {
   runtime { ... }
   inputs {
       File input_file
```

```
        String name
        Int threshold
    }

    command <<<
    my_tool --name ~{name} --threshold ~{threshold} ~{input_file}
    >>>

    output {
        File results = "results.txt"
    }
}

workflow my_workflow {
    inputs {
        File input_file
        String name
        Int threshold = 50
    }

    call my_task {
        input:
           input_file = input_file,
           name = name,
           threshold = threshold
    }
    outputs {
        File results = my_task.results
    }
}
```

Nextflow

```
nextflow.enable.dsl = 2

params.input_file = null
params.name = null
params.threshold = 50

process my_task {
    // <directives>

    input:
```

```
     path input_file
     val name
     val threshold

   output:
     path 'results.txt', emit: results

   script:
     """
     my_tool --name ${name} --threshold ${threshold} ${input_file}
     """


}

workflow MY_WORKFLOW {
   my_task(
       params.input_file,
       params.name,
       params.threshold
   )
}

workflow {
   MY_WORKFLOW()
}
```

CWL

```
cwlVersion: v1.2
class: Workflow

requirements:
    InlineJavascriptRequirement: {}

inputs:
   input_file: File
   name: string
   threshold: int

outputs:
```

```
    result:
        type: ...
        outputSource: ...

steps:
    my_task:
        run:
            class: CommandLineTool
            baseCommand: my_tool
            requirements:
                ...
            inputs:
                name:
                    type: string
                    inputBinding:
                        prefix: "--name"
                threshold:
                    type: int
                    inputBinding:
                        prefix: "--threshold"
                input_file:
                    type: File
                    inputBinding: {}
            outputs:
                results:
                    type: File
                    outputBinding:
                        glob: results.txt
```

Parameter template files for HealthOmics workflows

Parameter templates define the input parameters for a workflow. You can define input parameters to make your workflow more flexible and versatile. For example, you can define a parameter for the Amazon S3 location of the reference genome files. Parameter templates can be provided through a Git-based repository service or your local drive. Users can then run the workflow using various data sets.

You can create the parameter template for your workflow, or HealthOmics can generate the parameter template for you.

The parameter template is a JSON file. In the file, each input parameter is a named object that must match the name of the workflow input. When you start a run, if you don't provide values for all the required parameters, the run fails.

The input parameter object includes the following attributes:

- **description** – This required attribute is a string that the console displays in the **Start run** page. This description is also retained as run metadata.
- **optional** – This optional attribute indicates whether the input parameter is optional. If you don't specify the **optional** field, the input parameter is required.

The following example parameter template shows how to specify the input parameters.

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

Generating parameter templates

HealthOmics generates the parameter template by parsing the workflow definition to detect input parameters. If you provide a parameter template file for a workflow, the parameters in your file override the parameters detected in the workflow definition.

There are slight differences between the parsing logic of the CWL, WDL, and Nextflow engines, as described in the following sections.

Topics

- Parameter detection for CWL
- Parameter detection for WDL
- Parameter detection for Nextflow

Parameter detection for CWL

In the CWL workflow engine, the parsing logic makes the following assumptions:

- Any nullable supported types are marked as optional input parameters.
- Any non-null supported types are marked as required input parameters.
- Any parameters with default values are marked as optional input parameters.
- Descriptions are extracted from the `label` section from the `main` workflow definition. If `label` is not specified, the description will be blank (an empty string).

The following tables show CWL interpolation examples. For each example, the parameter name is `x`. If the parameter is required, you must provide a value for the parameter. If the parameter is optional, you don't need to provide a value.

This table shows CWL interpolation examples for primitive types.

Input	Example input/output	Required
`x:` `  type: int`	1 or 2 or ...	Yes

`x:`
`  type: int`
`  default: 2`
	Default value is 2. Valid input is 1 or 2 or ...	No

`x:` `  type: int?`	Valid input is None or 1 or 2 or ...	No

`x:`
`  type: int?`
`  default: 2`
	Default value is 2. Valid input is None or 1 or 2 or ...	No

The following table shows CWL interpolation examples for complex types. A complex type is a collection of primitive types.

Input	Example input/output	Required

`x:`
`  type: array`
`  items: int`
	[] or [1,2,3]	Yes

`x:`
`  type: array?`
`  items: int`
	None or [] or [1,2,3]	No

`x:`
`  type: array`
`  items: int?`
	[] or [None, 3, None]	Yes

`x:`
`  type: array?`
`  items: int?`
	[None] or None or [1,2,3] or [None, 3] but not []	No

Parameter detection for WDL

In the WDL workflow engine, the parsing logic makes the following assumptions:

- Any nullable supported types are marked as optional input parameters.
- For non-nullable supported types:
  - Any input variable with assignment of literals or expression are marked as optional parameters. For example:

    ```
     Int x = 2
    Float f0 = 1.0 + f1
    ```

  - If no values or expressions have been been assigned to the input parameters, they will be marked as required parameters.
- Descriptions are extracted from `parameter_meta` in the `main` workflow definition. If `parameter_meta` is not specified, the description will be blank (an empty string). For more information, see the WDL specification for Parameter metadata.

The following tables show WDL interpolation examples. For each example, the parameter name is x. If the parameter is required, you must provide a value for the parameter. If the parameter is optional, you don't need to provide a value.

This table shows WDL interpolation examples for primitive types.

Input	Example input/output	Required
Int x	1 or 2 or ...	Yes
Int x = 2	2	No
Int x = 1+2	3	No
Int x = y+z	y+z	No
Int? x	None or 1 or 2 or ...	Yes
Int? x = 2	None or 2	No
Int? x = 1+2	None or 3	No
Int? x = y+z	None or y+z	No

The following table shows WDL interpolation examples for complex types. A complex type is a collection of primitive types.

Input	Example input/output	Required		
Array[Int] x	[1,2,3] or []	Yes		
Array[Int]+ x	[1], but not []	Yes		
Array[Int]? x	None or [] or [1,2,3]	No		
Array[Int?] x	[] or [None, 3, None]	Yes		

Input	Example input/ output	Required		
Array[Int?]=? x	[None] or None or [1,2,3] or [None, 3] but not []	No		
Struct sample {String a, Int y} later in inputs: Sample mySample	`String a = mySample.a Int y = mySample.y`	Yes		
Struct sample {String a, Int y} later in inputs: Sample? mySample	`if (defined( mySample)) { String a = mySample.a Int y = mySample.y }`	No		

Parameter detection for Nextflow

For Nextflow, HealthOmics generates the parameter template by parsing the `nextflow_schema.json` file. If the workflow definition doesn't include a schema file, HealthOmics parses the main workflow definition file.

Topics

- Parsing the schema file
- Parsing the main file
- Nested parameters
- Examples of Nextflow interpolation

Parsing the schema file

For parsing to work correctly, make sure the schema file meets the following requirements:

- The schema file is named `nextflow_schema.json` and is located in the same directory as the main workflow file.
- The schema file is valid JSON as defined in either of the following schemas:
  - json-schema.org/draft/2020-12/schema.
  - json-schema.org/draft-07/schema.

HealthOmics parses the `nextflow_schema.json` file to generate the parameter template:

- Extracts all **properties** that are defined in the schema.
- Includes the property **description** if available for the property.
- Identifies whether each parameter is optional or required, based on the **required** field of the property.

The following example shows a definition file and the generated parameter file.

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "object",
    "$defs": {
        "input_options": {
            "title": "Input options",
            "type": "object",
            "required": ["input_file"],
            "properties": {
                "input_file": {
                    "type": "string",
                    "format": "file-path",
                    "pattern": "^s3://[a-z0-9.-]{3,63}(?:/\\S*)?$",
                    "description": "description for input_file"
                },
                "input_num": {
                    "type": "integer",
                    "default": 42,
                    "description": "description for input_num"
                }
```

```
            }
        },
        "output_options": {
            "title": "Output options",
            "type": "object",
            "required": ["output_dir"],
            "properties": {
                "output_dir": {
                    "type": "string",
                    "format": "file-path",
                    "description": "description for output_dir",
                }
            }
        }
    },
    "properties": {
        "ungrouped_input_bool": {
            "type": "boolean",
            "default": true
        }
    },
    "required": ["ungrouped_input_bool"],
    "allOf": [
        { "$ref": "#/$defs/input_options" },
        { "$ref": "#/$defs/output_options" }
    ]
}
```

The generated parameter template:

```
{
    "input_file": {
        "description": "description for input_file",
        "optional": False
    },
    "input_num": {
        "description": "description for input_num",
        "optional": True
    },
    "output_dir": {
        "description": "description for output_dir",
        "optional": False
    },
```

```
    "ungrouped_input_bool": {
        "description": None,
        "optional": False
    }
}
```

Parsing the main file

If the workflow definition doesn't include a `nextflow_schema.json` file, HealthOmics parses the main workflow definition file.

HealthOmics analyzes the `params` expressions found in the main workflow definition file and in the `nextflow.config` file. All `params` with default values are marked as optional.

For parsing to work correctly, note the following requirements:

- HealthOmics parses only the main workflow definition file. To ensure all parameters are captured, we recommend that you wire all **params** through to any submodules and imported workflows.
- The config file is optional. If you define one, name it `nextflow.config` and place it in the same directory as the main workflow definition file.

The following example shows a definition file and the generated parameter template.

```
params.input_file = "default.txt"
params.threads = 4
params.memory = "8GB"

workflow {
    if (params.version) {
        println "Using version: ${params.version}"
    }
}
```

The generated parameter template:

```
{
    "input_file": {
        "description": None,
        "optional": True
    },
```

```
    "threads": {
        "description": None,
        "optional": True
    },
    "memory": {
        "description": None,
        "optional": True
    },
    "version": {
        "description": None,
        "optional": False
    }
}
```

For default values that are defined in nextflow.config, HealthOmics collects `params` assignments and parameters declared within `params {}`, as shown in the following example. In assignment statements, `params` must appear in the left side of the statement.

```
params.alpha = "alpha"
params.beta = "beta"

params {
    gamma = "gamma"
    delta = "delta"
}

env {
   // ignored, as this assignment isn't in the params block
   VERSION = "TEST"
}

// ignored, as params is not on the left side
interpolated_image = "${params.cli_image}"
```

The generated parameter template:

```
{
    // other params in your main workflow defintion
    "alpha": {
        "description": None,
        "optional": True
    },
```

```
    "beta": {
        "description": None,
        "optional": True
    },
    "gamma": {
        "description": None,
        "optional": True
    },
    "delta": {
        "description": None,
        "optional": True
    }
}
```

Nested parameters

Both `nextflow_schema.json` and `nextflow.config` allow nested parameters. However, the HealthOmics parameter template requires only the top-level parameters. If your workflow uses a nested parameter, you must provide a JSON object as the input for that parameter.

Nested parameters in schema files

HealthOmics skips nested **params** when parsing a `nextflow_schema.json` file. For example, if you define the following `nextflow_schema.json` file:

```
{
    "properties": {
        "input": {
            "properties": {
                "input_file": { ... },
                "input_num": { ... }
            }
        },
        "input_bool": { ... }
    }
}
```

HealthOmics ignores `input_file` and `input_num` when it generates the parameter template:

```
{
    "input": {
        "description": None,
```

```
        "optional": True
    },
    "input_bool": {
        "description": None,
        "optional": True
    }
}
```

When you run this workflow, HealthOmics expects an `input.json` file similar to the following:

```
{
   "input": {
       "input_file": "s3://bucket/obj",
       "input_num": 2
   },
   "input_bool": false
}
```

Nested parameters in config files

HealthOmics doesn't collect nested **params** in a `nextflow.config` file, and skips them during parsing. For example, if you define the following `nextflow.config` file:

```
params.alpha = "alpha"
  params.nested.beta = "beta"

  params {
      gamma = "gamma"
      group {
          delta = "delta"
      }
  }
```

HealthOmics ignores `params.nested.beta` and `params.group.delta` when it generates the parameter template:

```
{
    "alpha": {
        "description": None,
        "optional": True
    },
    "gamma": {
```

```
        "description": None,
        "optional": True
    }
}
```

Examples of Nextflow interpolation

The following table shows Nextflow interpolation examples for params in the main file.

Parameters	Required
params.input_file	Yes
params.input_file = "s3://bucket/data.json"	No
params.nested.input_file	N/A
params.nested.input_file = "s3://bucket/data.json"	N/A

The following table shows Nextflow interpolation examples for params in the `nextflow.config` file.

Parameters	Required
`params.input_file = "s3://bucket/data.json"`	No
`params { input_file = "s3://bucket/data.json" }`	No
`params { nested { input_file = "s3://bucket/data.json" }`	N/A

Parameters	Required
`}`	
`input_file = params.input_file`	N/A

Container images for private workflows

HealthOmics supports container images hosted in Amazon ECR private repositories. You can create container images and upload them to the private repository. You can also use your Amazon ECR private registry as a pull through cache to synchronize the contents of upstream registries.

Your Amazon ECR repository must reside in the same AWS Region as the account calling the service. A different AWS account can own the container image, as long as the source image repository provides appropriate permissions. For more information, see Policies for cross-account Amazon ECR access.

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.

> **Note**
>
> HealthOmics doesn't support ARM containers and doesn't support access to public repositories.

For information about configuring IAM permissions for HealthOmics to access Amazon ECR, see HealthOmics Resource permissions.

Topics

- Synchronizing with third-party container registries
- General considerations for Amazon ECR container images
- Environment variables for HealthOmics workflows
- Using Java in Amazon ECR container images
- Add task inputs to an Amazon ECR container image

Synchronizing with third-party container registries

You can use Amazon ECR pull through cache rules to synchronize repositories in a supported upstream registry with your Amazon ECR private repositories. For more information, see Sync an upstream registry in the *Amazon ECR User Guide*.

The pull through cache automatically creates the image repository in your private registry when you create the cache, and it automatically synchronizes with the cached image when there are changes to the upstream image.

HealthOmics supports pull through cache for the following upstream registries:

- Amazon ECR Public
- Kubernetes container image registry
- Quay
- Docker Hub
- Microsoft Azure Container Registry
- GitHub Container Registry
- GitLab Container Registry

HealthOmics doesn't support pull through cache for an upstream Amazon ECR private repository.

Benefits of using Amazon ECR pull through cache include:

1. You avoid having to manually migrate container images to Amazon ECR or to synchronize updates from the third party repository.
2. Workflows access the synchronized container images in your private repository, which is more reliable than downloading content at run time from a public registry.
3. Because Amazon ECR pull through caches use a predictable URI structure, the HealthOmics service can automatically map the Amazon ECR private URI with the upstream registry URI. You aren't required to update and replace URI values in the workflow definition.

Topics

- Configuring pull through cache
- Registry mappings
- Image mappings

Configuring pull through cache

Amazon ECR provides a registry for your AWS account in each Region. Make sure you create the Amazon ECR configuration in the same region where you plan to run the workflow.

The following sections describe the configuration tasks for pull through cache.

Configuration tasks

- Create a pull through cache rule
- Registry permissions for upstream registry
- Repository creation templates
- Creating the workflow

Create a pull through cache rule

Create an Amazon ECR pull through cache rule for each upstream registry that has images you want to cache. A rule specifies a mapping between an upstream registry and the Amazon ECR private repository.

For an upstream registry that requires authentication, you provide your credentials using AWS Secrets Manager.

> **Note**
>
> Don't change a pull through cache rule while an active run is using the private repository. The run could fail or, more critically, result in your pipeline using unexpected images.

For more information, see Creating a pull through cache rule in the *Amazon Elastic Container Registry User Guide*.

Create a pull through cache rule using the console

To configure pull through cache, follow these steps using the Amazon ECR console:

1. Open the Amazon ECR console : https://console.aws.amazon.com/ecr
2. From the left menu, under **Private registry**, expand **Features & Settings**. then choose **Pull through cache**.

3. From the **Pull through cache** page, choose **Add rule**.
4. In the **Upstream registry** panel, choose the upstream registry to sync with your private registry, then choose **Next**.
5. If the upstream registry requires authentication, the console opens a new page where you specify the SageMaker AI secret that contains your credentials. Choose **Next**.
6. Under **Specify namespaces**, in the **Cache namespace** panel, choose whether to create the private repositories using a specific repository prefix or with no prefix. If you choose to use a prefix, specify the prefix name in **Cache repository prefix**.
7. In the **Upstream namespace** panel, choose whether to pull from upstream repositories using a specific repository prefix or with no prefix. If you choose to use a prefix, specify the prefix name in **Upstream repository prefix**.

   The **Namespace example** panel shows an example pull request, upstream URL, and the URL of the cache repository that is created.
8. Choose **Next**.
9. Review the configuration and choose **Create** to create the rule.

For more information, see Create a pull through cache rule (AWS Management Console).

Create a pull through cache rule using the CLI

Use the Amazon ECR **create-pull-through-cache-rule** command to create a pull through cache rule. For upstream registries that require authentication, store the credentials in an Secrets Manager secret.

The following sections provide examples for each supported upstream registry.

For Amazon ECR Public

The following example creates a pull through cache rule for the Amazon ECR Public registry. It specifies a repository prefix of `ecr-public`, which results in each repository created using the pull through cache rule to have the naming scheme of `ecr-public/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --upstream-registry-url public.ecr.aws \
     --region us-east-1
```

For Kubernetes Container Registry

The following example creates a pull through cache rule for the Kubernetes public registry. It specifies a repository prefix of `kubernetes`, which results in each repository created using the pull through cache rule to have the naming scheme of `kubernetes/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix kubernetes \
     --upstream-registry-url registry.k8s.io \
     --region us-east-1
```

For Quay

The following example creates a pull through cache rule for the Quay public registry. It specifies a repository prefix of `quay`, which results in each repository created using the pull through cache rule to have the naming scheme of `quay/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix quay \
     --upstream-registry-url quay.io \
     --region us-east-1
```

For Docker Hub

The following example creates a pull through cache rule for the Docker Hub registry. It specifies a repository prefix of `docker-hub`, which results in each repository created using the pull through cache rule to have the naming scheme of `docker-hub/`*`upstream-repository-name`*. You must specify the full Amazon Resource Name (ARN) of the secret containing your Docker Hub credentials.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix docker-hub \
     --upstream-registry-url registry-1.docker.io \
     --credential-arn arn:aws:secretsmanager:us-east-1:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-1
```

For GitHub Container Registry

The following example creates a pull through cache rule for the GitHub Container Registry. It specifies a repository prefix of `github`, which results in each repository created using the pull

through cache rule to have the naming scheme of `github/`*`upstream-repository-name`*. You must specify the full Amazon Resource Name (ARN) of the secret containing your GitHub Container Registry credentials.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix github \
     --upstream-registry-url ghcr.io \
     --credential-arn arn:aws:secretsmanager:us-east-1:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-1
```

For Microsoft Azure Container Registry

The following example creates a pull through cache rule for the Microsoft Azure Container Registry. It specifies a repository prefix of `azure`, which results in each repository created using the pull through cache rule to have the naming scheme of `azure/`*`upstream-repository-name`*. You must specify the full Amazon Resource Name (ARN) of the secret containing your Microsoft Azure Container Registry credentials.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix azure \
     --upstream-registry-url myregistry.azurecr.io \
     --credential-arn arn:aws:secretsmanager:us-east-1:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-1
```

For GitLab Container Registry

The following example creates a pull through cache rule for the GitLab Container Registry. It specifies a repository prefix of `gitlab`, which results in each repository created using the pull through cache rule to have the naming scheme of `gitlab/`*`upstream-repository-name`*. You must specify the full Amazon Resource Name (ARN) of the secret containing your GitLab Container Registry credentials.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix gitlab \
     --upstream-registry-url registry.gitlab.com \
     --credential-arn arn:aws:secretsmanager:us-east-1:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-1
```

For more information, see Create a pull through cache rule (CLI) in the *Amazon ECR User Guide*.

You can use the **get-run-task** CLI command to retrieve information about the container image used for a specific task:

```
aws omics get-run-task --id 1234567 --task-id <task_id>
```

The output includes the following information about the container image:

```
"imageDetails": {
   "image": "string",
   "imageDigest": "string",
   "sourceImage": "string",
         ...
}
```

Registry permissions for upstream registry

Use registry permissions to allow HealthOmics to use the pull through cache and to pull the container images into the Amazon ECR private registry. Add an Amazon ECR Registry policy to the registry that provides the containers used in runs.

The following policy grants permission for the HealthOmics service to create repositories with the specified pull through cache prefix(es) and to initiate upstream pulls into these repositories.

1. From the Amazon ECR console, open the left menu, under **Private registry**, expand **Registry permissions**. then choose **Generate statement**.
2. On the top right side, choose JSON. Enter a policy similar to the following:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPTCinRegPermissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "omics.amazonaws.com"
            },
```

```
            "Action": [
                "ecr:CreateRepository",
                "ecr:BatchImportUpstreamImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-east-1:123456789012:repository/ecr-public/*",
                "arn:aws:ecr:us-east-1:123456789012:repository/docker-hub/*"
            ]
        }
    ]
}
```

Repository creation templates

To use pull through caching in HealthOmics, the Amazon ECR repository must have a repository creation template. The template defines configuration settings for when you or Amazon ECR create a private repository for an upstream registry.

Each template contains a repository namespace prefix, which Amazon ECR uses to match new repositories to a specific template. Templates specify the configuration for all repository settings including resource-based access policies, tag immutability, encryption, and lifecycle policies.

For more information, see Repository creation templates in the *Amazon Elastic Container Registry User Guide*.

How to create a repository creation template:

1. From the Amazon ECR console, open the left menu, under **Private registry**, expand **Features and settings**. then choose **Repository creation templates**.
2. Choose **Create template**.
3. In **Template details**, choose **Pull through cache**.
4. Choose whether to apply this template to a specific prefix or to all repositories that don't match another template.

   If you choose **A specific prefix**, enter the namespace prefix value in **Prefix**. You specified this prefix when you created the PTC rule.
5. Choose **Next**.
6. In **Add repository creation configuration** page, enter **Repository permissions**. Use one of the sample policy statements, or enter one similar to the following example:

JSON

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

7. Optionally, you can add repository settings such as lifecycle policy and tags. Amazon ECR applies these rules for all container images created for pull through cache that use the specified prefix.
8. Choose **Next**.
9. Review the configuration and choose **Next**.

Creating the workflow

When you create a new workflow or workflow version, review the registry mappings and update them if required. For details, see Create a private workflow.

Registry mappings

You define registry mappings to map between prefixes in your private Amazon ECR registry and the upstream registry names.

For more information about Amazon ECR registry mappings, see Creating a pull through cache rule in Amazon ECR.

The following example shows registry mappings to Docker Hub, Quay, and Amazon ECR Public.

```
{
    "registryMappings": [
        {
            "upstreamRegistryUrl": "registry-1.docker.io",
            "ecrRepositoryPrefix": "docker-hub"
        },
        {
            "upstreamRegistryUrl": "quay.io",
            "ecrRepositoryPrefix": "quay"
        },
        {
            "upstreamRegistryUrl": "public.ecr.aws",
            "ecrRepositoryPrefix": "ecr-public"
        }
    ]
}
```

Image mappings

You define image mappings to map between the image names as defined in your private Amazon ECR workflows and the image names in the upstream registry.

You can use image mappings with registries that support pull through cache. You can also use image mappings with upstream registries where HealthOmics doesn't support pull through cache. You need to manually synchronize the upstream registry with your private repository.

For more information about Amazon ECR image mappings, see Creating a pull through cache rule in Amazon ECR.

The following example shows mappings from private Amazon ECR images to a public genomics image and the latest Ubuntu image.

```
{
    "imageMappings": [
        {
            "sourceImage": "public.ecr.aws/aws-genomics/broadinstitute/gatk:4.6.0.2",
            "destinationImage": "123456789012.dkr.ecr.us-east-1.amazonaws.com/
broadinstitute/gatk:4.6.0.2"
        },
        {
            "sourceImage": "ubuntu:latest",
```

```
            "destinationImage": "123456789012.dkr.ecr.us-east-1.amazonaws.com/custom/
ubuntu:latest",
        }
    ]
}
```

General considerations for Amazon ECR container images

- Architecture

  HealthOmics supports x86_64 containers. If your local machine is ARM-based, such as Apple Mac, use a command such as the following to build an x86_64 container image:

  ```
  docker build --platform amd64 -t my_tool:latest .
  ```

- Entrypoint and shell

  HealthOmics workflow engines inject bash scripts as a command override to the container images used by workflow tasks. Thus, container images should be built without a specified ENTRYPOINT such that a bash shell is the default.

- Mounted paths

  A shared filesystem is mounted to container tasks at /tmp. Any data or tooling built into the container image at this location will be overridden.

  The workflow definition is available to tasks via a read-only mount at /mnt/workflow.

- Image size

  See HealthOmics workflow fixed size quotas for the maximum container image sizes.

Environment variables for HealthOmics workflows

HealthOmics provides environment variables that have information about the workflow running in the container. You can use the values of these variables in the logic of your workflow tasks.

All HealthOmics workflow variables start with the `AWS_WORKFLOW_` prefix. This prefix is a protected environment variable prefix. Don't use this prefix for your own variables in workflow containers.

HealthOmics provides the following workflow environment variables:

AWS_REGION

This variable is the region where the container is running.

AWS_WORKFLOW_RUN

This variable is the name of the current run.

AWS_WORKFLOW_RUN_ID

This variable is the run identifier of the current run.

AWS_WORKFLOW_RUN_UUID

This variable is the run UUID of the current run.

AWS_WORKFLOW_TASK

This variable is the name of the current task.

AWS_WORKFLOW_TASK_ID

This variable is the task identifier of the current task.

AWS_WORKFLOW_TASK_UUID

This variable is the task UUID of the current task.

The following example shows typical values for each environment variable:

```
AWS Region: us-east-1
Workflow Run: arn:aws:omics:us-east-1:123456789012:run/6470304
Workflow Run ID: 6470304
Workflow Run UUID: f4d9ed47-192e-760e-f3a8-13afedbd4937
Workflow Task: arn:aws:omics:us-east-1:123456789012:task/4192063
Workflow Task ID: 4192063
Workflow Task UUID: f0c9ed49-652c-4a38-7646-60ad835e0a2e
```

Using Java in Amazon ECR container images

If a workflow task uses a Java application such as GATK, consider the following memory requirements for the container:

- Java applications use stack memory and heap memory. By default, the maximum heap memory is a percentage of the total available memory in the container. This default depends on the

specific JVM distribution and JVM version, so consult the relevant documentation for your JVM or explicitly set the heap memory maximum using Java command line options (such as `-Xmx`).

- Don't set the maximum heap memory to be 100% of the container's memory allocation, because the JVM stack also requires memory. Memory is also required for the JVM garbage collector and any other operating system processes running in the container.
- Some Java applications, such as GATK, can use native method invocations or other optimizations such as memory mapping files. These techniques require memory allocations that are performed “off heap”, which aren't controlled by the JVM maximum heap parameter.

  If you know (or suspect) that your Java application allocates off-heap memory, make sure your task memory allocation includes the off-heap memory requirements.

  If these off-heap allocations cause the container to run out of memory, you typically won't see a Java **OutOfMemory** error, because the JVM doesn't control this memory.

Add task inputs to an Amazon 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 tasks 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.

See HealthOmics workflow fixed size quotas for the maximum size of container image that HealthOmics supports.

HealthOmics Workflow README files

You can upload a README.md file containing instructions, diagrams, and essential information for your workflow. Each workflow version supports one README file, which you can update at any time.

README requirements include:

- README file must be in markdown (.md) format
- Maximum file size: 500 KiB

Topics

- Use an existing README
- Rendering conditions

Use an existing README

READMEs exported from Git repositories contain relative links that typically do not work outside the repository. HealthOmics Git integration automatically converts these to absolute links for proper rendering in the console, eliminating the need for manual URL updates.

For READMEs imported from Amazon S3 or local drives, images and links must either use public URLs or have their relative paths updated for proper rendering.

> **Note**
>
> Images must be publicly hosted to display in the HealthOmics console. Images stored in GitHub Enterprise Server or GitLab Self-Managed repositories cannot be rendered.

Rendering conditions

The HealthOmics console interpolates publicly accessible images and links using absolute paths. To render URLs from private repositories, the user must have access to the repository. For GitHub Enterprise Server or GitLab Self-Managed repositories, which use custom domains, HealthOmics cannot resolve relative links or render images stored in these private repositories.

The following table shows the markdown elements that are supported by the AWS console README view.

Element	AWS console
Alerts	Yes, but without icons
Badges	Yes
Basic text formatting	Yes
Code blocks	Yes, but does not have syntax highlight and copy button functionality

Element	AWS console
Collapsible sections	Yes
Headings	Yes
Image formats	Yes
Image (clickable)	Yes
Line breaks	Yes
Mermaid diagram	Only can open graph, move graph position, and copy code
Quotes	Yes
Subscript and superscript	Yes
Tables	Yes, but does not support text alignment
Text alignment	Yes

Using image and link URLs

Depending on your source provider, structure your base URLs for pages and images in the following formats.

- `{username}`: The username where the repository is hosted.
- `{repo}`: The repository name.
- `{ref}`: The source reference (branch, tag, and commit ID).
- `{path}`: The file path to the page or image in the repository.

Source provider	Page URL	Image URL
GitHub	`https://github.com/{username}/{repo}/blob/{ref}/{path}`	`https://github.com/{username}/{repo}/`

Source provider	Page URL	Image URL

		`blob/{ref}/{path}?raw=true`
		
		`https://raw.githubusercontent.com/{username}/{repo}/{ref}/{path}`

GitLab	`https://gitlab.com/{username}/{repo}/-/blob/{ref}/{path}`	`https://gitlab.com/{username}/{repo}/-/raw/{ref}/{path}`
Bitbucket	`https://bitbucket.org/{username}/{repo}/src/{ref}/{path}`	`https://bitbucket.org/{username}/{repo}/raw/{ref}/{path}`

GitHub, GitLab, and Bitbucket support both page and image URLs that link to a public repository. The following table shows each source provider's support for rendering image and link URLs for private repositories.

Private repository support		
Source provider	**Page URL**	**Image URL**
GitHub	Only with access to repository	No
GitLab	Only with access to repository	No
Bitbucket	Only with access to repository	No

Requesting Sentieon licenses for private workflows

If your private workflow uses Sentieon software, you need a Senieon license. Follow these steps to request and set up a license for the Sentieon software:

- Request a Sentieon license
  - Send an email to the Sentieon support group (support@sentieon.com) to request a software license.
    - Provide your AWS Canonical User ID in the email.
    - Find your AWS Canonical User ID by following these instructions.
- Update your HealthOmics service role to grant it access to the Sentieon licensing server proxy and Sentieon Omics bucket in your Region. The following example grants access in `us-east-1`. If required, replace this text with your Region.

  JSON

```
{
          "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectAcl",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::omics-ap-us-east-1/*",
                "arn:aws:s3:::sentieon-omics-license-us-east-1/*"
            ]
        }
    ]
 }
```

- Generate an AWS support case to get access to the Sentieon license server proxy.
  - To create a support case, navigate to support.console.aws.amazon.com.
  - Provide your AWS account and Region in the support case. Your account is added to the allowlist for the licensing server proxy.
- Build your private workflow using the Sentieon container and the Sentieon license script.
  - For additional instructions on using Sentieon tools inside private workflows, see Sentieon-Amazon-Omics in GitHub.
- Sentieon software version 202112.07 and higher support the HealthOmics licensing server proxy. To use Sentieon software versions earlier than 202112.07, contact Sentieon support.

Workflow linters in HealthOmics

After you create a workflow, we recommend that you run a linter on the workflow before you start the first run. The linter detects errors that can cause the run to fail.

For WDL, HealthOmics automatically runs a linter when you create the workflow. The linter output is available in the `statusMessage` field of the **get-workflow** response. Use the following CLI command to retrieve the status output (use the workflow ID of the WDL workflow that you created):

```
aws omics get-workflow
   –id 123456
   –query 'statusMessage'
```

HealthOmics provides linters that you can run on the workflow defnition before you create the workflow. Run these linters on existing pipelines that you're migrating to HealthOmics.

- **WDL** – A public Amazon ECR image to run a WDL linter.

- **Nextflow** – A public Amazon ECR image to run Linter rules for Nextflow. You can access the source code for this linter from GitHub.
- **CWL** – not available

HealthOmics workflow operations

To create a private workflow, you need:

- **Workflow definition file:** A workflow definition file written in WDL, Nextflow, or CWL. The workflow definition specifies the inputs and outputs for runs that use the workflow. It also includes specifications for the runs and run tasks for your workflow, including compute and memory requirements. The workflow definition file must be in `.zip` format. For more information, see Workflow definition files in HealthOmics.
  - You can use Amazon Q CLI to build and validate your workflow definition files in WDL, Nextflow, and CWL. For more information, see Example prompts for Amazon Q CLI and the HealthOmics Agentic generative AI tutorial on GitHub.

- **(Optional) Parameter template file:** A parameter template file written in JSON. Create the file to define the run parameters, or HealthOmics generates the parameter template for you. For more information, see Parameter template files for HealthOmics workflows.
- **Amazon ECR container images:** Create private Amazon ECR repositories for each container used in the workflow. Create container images for the workflow and store them in a private repository, or synchronize the contents of a supported upstream registry with your ECR private repository.
- **(Optional) Sentieon licenses:** Request a Sentieon license to use the Sentieon software in private workflows.

For workflow definition files larger than 4 MiB (zipped), choose one of these options during workflow creation:

- Upload to an Amazon Simple Storage Service folder and specify the location.
- Upload to an external repository (max size 1 GiB) and specify the repository details.

After you create a workflow, you can update the following workflow information with the `UpdateWorkflow` operation:

- Name
- Description
- Default storage type
- Default storage capacity (with workflow ID)
- README.md file

To change other information in the workflow, create a new workflow or workflow version.

Use workflow versioning to organize and structure you workflows. Versions also help you to manage the introduction of iterative workflow updates. For more information about versions, see Create a workflow version.

Topics

- Create a private workflow
- Update a private workflow
- Delete a private workflow
- Verify the workflow status

- Referencing genome files from a workflow definition

Create a private workflow

Create a workflow using the HealthOmics console, AWS CLI commands, or one of the AWS SDKs.

> **Note**
>
> Don't include any personally identifiable information (PII) in workflow names. These names are visible in CloudWatch logs.

When you create a workflow, HealthOmics assigns a universally unique identifier (UUID) to the workflow. The workflow UUID is a Globally Unique Identifier (guid) that's unique across workflows and workflow versions. For data provenance purposes, we recommend that you use the workflow UUID to uniquely identify workflows.

If your workflow tasks use any external tools (executables, libraries, or scripts), you build these tools into a container image. You have the following options for hosting the container image:

- Host the container image in the ECR private registry. Prerequisites for this option:
  - Create an ECR private repository, or choose an existing repository.
  - Configure the ECR resource policy as described in Amazon ECR permissions.
  - Upload your container image to the private repository.
- Synchronize the container image with the contents of a supported third-party registry. Prerequisites for this option:
  - In the ECR private registry, configure a pull through cache rule for each upstream registry. For more information, see Image mappings.
  - Configure the ECR resource policy as described in Amazon ECR permissions.
  - Create repository creation templates. The template defines settings for when Amazon ECR creates the private repository for an upstream registry.
  - Create prefix mappings to remap container image references in the workflow definition to ECR cache namespaces.

When you create a workflow, you provide a workflow definition that contains information about the workflow, runs, and tasks. HealthOmics can retrieve the workflow definition as a .zip archive stored locally or in an Amazon S3 bucket, or from a supported Git-based repository.

Topics

- Creating a workflow using the console
- Creating a workflow using the CLI
- Creating a workflow using an SDK

Creating a workflow using the console

Steps to create a workflow

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose **Create workflow**.
4. On the **Define workflow** page, provide the following information:
    1. **Workflow name**: A distinctive name for this workflow. We recommend setting workflow names to organize your runs in the AWS HealthOmics console and CloudWatch logs.
    2. **Description** (optional): A description of this workflow.
5. In the **Workflow definition** panel, provide the following information:
    1. **Workflow language** (optional): Select the specification language of the workflow. Otherwise, HealthOmics determines the language from the workflow definition.
    2. For **Workflow definition source**, choose to import the definition folder from a Git-based repository, an Amazon S3 location, or from a local drive.
        a. For **Import from a repository service**:

> **Note**
>
> HealthOmics supports public and private repositories for GitHub, GitLab, Bitbucket, GitHub self-managed, GitLab self-managed.

i. Choose a **Connection** to connect your AWS resources to the external repository. To create a connection, see Connect with external code repositories.

> **Note**
>
> Customers in the TLV region need to create a connection in the IAD (us-east-1) region to create a workflow.

ii. In **Full repository ID**, enter your repository ID as user-name/repo-name. Verify you have access to the files in this repository.

iii. In **Source reference** (optional), enter a repository source reference (branch, tag, or commit ID). HealthOmics uses the default branch if no source reference is specified.

iv. In **Exclude file patterns**, enter the file patterns to exclude specific folders, files, or extensions. This helps manage data size when importing repository files. There is a max of 50 patterns, and the patters must follow the glob pattern syntax. For example:

A. `tests/`

B. `*.jpeg`

C. `large_data.zip`

b. For **Select definition folder from S3**:

i. Enter the Amazon S3 location that contains the zipped workflow definition folder. The Amazon S3 bucket must be in the same region as the workflow.

ii. If your account doesn't own the Amazon S3 bucket, enter the bucket owner's AWS account ID in the **S3 bucket owner's account ID**. This information is required so that HealthOmics can verify the bucket ownership.

c. For **Select definition folder from a local source**:

i. Enter the local drive location of the zipped workflow definition folder.

3. **Main workflow definition file path** (optional): Enter the file path from the zipped workflow definition folder or repository to the `main` file. This parameter is not required if there is only one file in the workflow definition folder, or if the main file is named "main".

6. In the **README file** (optional) panel, select the **Source of the README file** and provide the following information:

- For **Import from a repository service**, in **README file path**, enter the path to the README file within the repository.
- For **Select file from S3**, in **README file in S3**, enter the Amazon S3 URI for the README file.
- For **Select file from a local source**: in **README - optional**, chose **Choose file** to select the markdown (.md) file to upload.

7. In the **Default run storage configuration** panel, provide the default run storage type and capacity for runs that use this workflow:

    1. **Run storage type**: Choose whether to use static or dynamic storage as the default for the temporary run storage. The default is static storage.
    2. **Run storage capacity** (optional): For static run storage type, you can enter the default amount of run storage required for this workflow. The default value for this parameter is 1200 GiB. You can override these default values when you start a run.

8. **Tags** (optional): You can associate up to 50 tags with this workflow.
9. Choose **Next**.
10. On the **Add workflow parameters** (optional) page, select the **Parameter source**:

    1. For **Parse from workflow definition file**, HealthOmics will automatically parse the workflow parameters from the workflow definition file.
    2. For **Provide parameter template from Git repository**, use the path to the parameter template file from your repository.
    3. For **Select JSON file from local source**, upload a JSON file from a local source that specifies the parameters.
    4. For **Manually enter workflow parameters**, manually enter parameter names and descriptions.

11. In the **Parameter preview** panel, you can review or change the parameters for this workflow version. If you restore the JSON file, you lose any local changes that you made.
12. Choose **Next**.
13. On the **Container URI remapping** page, in the **Mapping rules** panel, you can define URI mapping rules for your workflow.

    For **Source of mapping file**, select one of the following options:

    - **None** – No mapping rules required.

- **Select JSON file from S3** – Specify the S3 location for the mapping file.
- **Select JSON file from a local source** – Specify the mapping file location on your local device.
- **Manually enter mappings** – enter the registry mappings and image mappings in the **Mappings** panel.

14. The console displays the **Mappings** panel. If you chose a mapping source file, the console displays the values from the file.

    a. In **Registry mappings**, you can edit the mappings or add mappings (maximum of 20 registry mappings).

       Each registry mapping contains the following fields:

       - **Upstream registry URL** – The URI of the upstream registry.
       - **ECR repository prefix** – The repository prefix to use in the Amazon ECR private repository.
       - (Optional) **Upstream repository prefix** – The prefix of the repository in the upstream registry.
       - (Optional) **ECR account ID** – Account ID of the account that owns the upstream container image.

    b. In **Image mappings**, you can edit the image mappings or add mappings (maximum of 100 image mappings).

       Each image mapping contains the following fields:

       - **Source image** – Specifies the URI of the source image in the upstream registry.
       - **Destination image** – Specifies the URI of the corresponding image in the private Amazon ECR registry.

15. Choose **Next**.
16. Review the workflow configuration, then choose **Create workflow**.

Creating a workflow using the CLI

If your workflow files and the parameter template file are on your local machine, you can create a workflow using the following CLI command.

```
aws omics create-workflow  \
  --name "my_workflow"   \
  --definition-zip fileb://my-definition.zip \
  --parameter-template file://my-parameter-template.json
```

The `create-workflow` operation returns the following response:

```
{
  "arn": "arn:aws:omics:us-west-2:....",
  "id": "1234567",
  "status": "CREATING",
  "tags": {
      "resourceArn": "arn:aws:omics:us-west-2:...."
  },
  "uuid": "64c9a39e-8302-cc45-0262-2ea7116d854f"
}
```

Optional parameters to use when creating a workflow

You can specify any of the optional parameters when you create a workflow. For syntax details, see CreateWorkflow in the AWS HealthOmics API Reference.

Topics

- Specify the workflow definition Amazon S3 location
- Use the workflow definition from a Git-based repository
- Specify a Readme file
- Specify the main definition file
- Specify the run storage type
- Specify the GPU configuration
- Configure pull through cache mapping parameters

Specify the workflow definition Amazon S3 location

If your workflow definition file is located in an Amazon S3 folder, specify the location using the `definition-uri` parameter, as shown in the following example. If your account does not own the Amazon S3 bucket, provide the owner's AWS account ID.

```
aws omics create-workflow  \
```

```
  --name Test  \
  --definition-uri s3://omics-bucket/workflow-definition/  \
  --owner-id  123456789012
    ...
```

Use the workflow definition from a Git-based repository

To use the workflow definition from a supported Git-based repository, use the `definition-repository` parameter in your request. Don't provide any other `definition` parameter, as a request fails if it includes more than one input source.

The `definition-respository` parameter contains the following fields:

- **connectionArn** – ARN of the Code Connection that connects your AWS resources to the external repository.
- **fullRepositoryId** – Enter the repository ID as `owner-name/repo-name`. Verify you have access to the files in this repository.
- **sourceReference** (Optional) – Enter a repository reference type (BRANCH, TAG, or COMMIT) and a value.

  HealthOmics uses the latest commit on the default branch if you don't specify a source reference.
- **excludeFilePatterns** (Optional) – Enter the file patterns to exclude specific folders, files, or extensions. This helps manage data size when importing repository files. Provide a maximum of 50 patterns. the patterns must follow the glob pattern syntax. For example:
  - `tests/`
  - `*.jpeg`
  - `large_data.zip`

When you specify the workflow definition from a Git-based repository, use `parameter-template-path` to specify the parameter template file. If you don't provide this parameter, HealthOmics creates the workflow without a parameter template.

The following example shows the parameters related to content from a Git-based private repository:

```
aws omics create-workflow \
```

```
    --name custom-variant \
    --description "Custom variant calling pipeline" \
    --engine "WDL" \
    --definition-repository '{
            "connectionArn": "arn:aws:codeconnections:us-
east-1:123456789012:connection/abcd1234-5678-90ab-cdef-1234567890ab",
            "fullRepositoryId": "myorg/my-genomics-workflows",
            "sourceReference": {
                "type": "BRANCH",
                "value": "main"
            },
            "excludeFilePatterns": ["tests/**", "*.log"]
      }' \
    --main "workflows/variant-calling/main.wdl" \
    --parameter-template-path "parameters/variant-calling-params.json" \
    --readme-path "docs/variant-calling-README.md" \
    --storage-type "DYNAMIC" \
```

For more examples, see the blog post How To Create an AWS HealthOmics Workflows from Content in Git.

Specify a Readme file

You can specify the README file location using one of the following parameters:

- **readme-markdown** – String input or a file on your local machine.
- **readme-uri** – The URI of a file stored on S3.
- **readme-path** – The path to the README file in the repository.

Use readme-path only in conjunction with **definition-respository**. If you don't specify any README parameter, HealthOmics imports the root level README.md file in the repository (if it exists).

The following examples show how specify the README file location using readme-path and readme-uri.

```
# Using README from repository
aws omics create-workflow \
    --name "documented-workflow" \
    --definition-repository '...' \
    --readme-path "docs/workflow-guide.md"
```

```
# Using README from S3
aws omics create-workflow \
    --name "s3-readme-workflow" \
    --definition-repository '...' \
    --readme-uri "s3://my-bucket/workflow-docs/readme.md"
```

For more information, see HealthOmics Workflow README files.

Specify the main definition file

If you are including multiple workflow definition files, use the `main` parameter to specify the main definition file for your workflow.

```
aws omics create-workflow  \
  --name Test  \
  --main multi_workflow/workflow2.wdl  \
    ...
```

Specify the run storage type

You can specify the default run storage type (DYNAMIC or STATIC) and run storage capacity (required for static storage). For more information about run storage types, see Run storage types in HealthOmics workflows.

```
aws omics create-workflow  \
  --name my_workflow   \
  --definition-zip fileb://my-definition.zip \
  --parameter-template file://my-parameter-template.json   \
  --storage-type 'STATIC'  \
  --storage-capacity 1200  \
```

Specify the GPU configuration

Use the accelerators parameter to create a workflow that runs on an accelerated-compute instance. The following example shows how to use the `accelerators` parameter. You specify the GPU configuration in the workflow definition. See Accelerated-computing instances.

```
aws omics create-workflow --name workflow name \
   --definition-uri s3://amzn-s3-demo-bucket1/GPUWorkflow.zip \
   --accelerators GPU
```

Configure pull through cache mapping parameters

If you're using the Amazon ECR pull through cache mapping feature, you can override the default mappings. For more information about the container setup parameters, see Container images for private workflows.

In the following example, file `mappings.json` contains this content:

```
{
    "registryMappings": [
        {
            "upstreamRegistryUrl": "registry-1.docker.io",
            "ecrRepositoryPrefix": "docker-hub"
        },
        {
            "upstreamRegistryUrl": "quay.io",
            "ecrRepositoryPrefix": "quay",
            "accountId": "123412341234"
        },
        {

            "upstreamRegistryUrl": "public.ecr.aws",
            "ecrRepositoryPrefix": "ecr-public"
        }
    ],
    "imageMappings": [{
            "sourceImage": "docker.io/library/ubuntu:latest",
            "destinationImage": "healthomics-docker-2/custom/ubuntu:latest",
            "accountId": "123412341234"
        },
        {
            "sourceImage": "nvcr.io/nvidia/k8s/dcgm-exporter",
            "destinationImage": "healthomics-nvidia/k8s/dcgm-exporter"
        }
    ]
}
```

Specify the mapping parameters in the create-workflow command:

```
aws omics create-workflow  \
     ...
--container-registry-map-file file://mappings.json
```

```
    ...
```

You can also specify the S3 location of the mapping parameters file:

```
aws omics create-workflow  \
     ...
--container-registry-map-uri s3://amzn-s3-demo-bucket1/test.zip
    ...
```

Creating a workflow using an SDK

You can create a workflow using one of the SDKs. The following example shows how to create a workflow using the Python SDK

```
import boto3

omics = boto3.client('omics')

with open('definition.zip', 'rb') as f:
   definition = f.read()

response = omics.create_workflow(
   name='my_workflow',
   definitionZip=definition,
   parameterTemplate={ ... }
)
```

Update a private workflow

You can update a workflow using the HealthOmics console, AWS CLI commands, or one of the AWS SDKs.

> **Note**
>
> Don't include any personally identifiable information (PII) in workflow names. These names are visible in CloudWatch logs.

Topics

- Updating a workflow using the console

- Updating a workflow using the CLI
- Updating a workflow using an SDK

Updating a workflow using the console

Steps to update a workflow

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose the workflow to update.
4. On the **Workflow** page:
   - If the workflow has versions, make sure that you select the **Default version**.
   - Choose **Edit selected** from the **Actions** list.
5. On the **Edit workflow** page, you can change any of the following values:
   - **Workflow name**.
   - **Workflow description**.
   - The default **Run storage type** for the workflow.
   - The default **Run storage capacity** (if the run storage type is static storage). For more information about the default run storage configuration, see Creating a workflow using the console.
6. Choose **Save changes** to apply the changes.

Updating a workflow using the CLI

As shown in the following example, you can update the workflow name and description. You can also change the default run storage type (STATIC or DYNAMIC) and run storage capacity (for static storage type). For more information about run storage types, see Run storage types in HealthOmics workflows.

```
aws omics update-workflow    \
  --id 1234567    \
  --name my_workflow      \
  --description "updated workflow"    \
  --storage-type 'STATIC'    \
```

```
  --storage-capacity 1200
```

You don't receive a response to the `update-workflow` request.

Updating a workflow using an SDK

You can update a workflow using one of the SDKs.

The following example shows how to update a workflow using the Python SDK

```
import boto3

omics = boto3.client('omics')

response = omics.update_workflow(
   name='my_workflow',
   description='updated workflow'
)
```

Delete a private workflow

When you no longer need a workflow, you can delete it using the HealthOmics console, AWS CLI commands, or one of the AWS SDKs. You can delete a workflow that meets the following criteria:

- Its status is ACTIVE or FAILED.
- It has no active shares.
- You've deleted all the workflow versions.

Deleting a workflow doesn't affect any ongoing runs that are using the workflow.

Topics

- Deleting a workflow using the console
- Deleting a workflow using the CLI
- Deleting a workflow using an SDK

Deleting a workflow using the console

To delete a workflow

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose the workflow to delete.
4. On the **Workflow** page, choose **Delete selected** from the **Actions** list.
5. In the **Delete workflow** modal, enter "confirm" to confirm the deletion.
6. Choose **Delete**.

Deleting a workflow using the CLI

The following example shows how you can use the AWS CLI command to delete a workflow. 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
```

HealthOmics doesn't send a response to the `delete-workflow` request.

Deleting a workflow using an SDK

You can delete a workflow using one of the SDKs.

The following example shows how to delete a workflow using the Python SDK.

```
import boto3

omics = boto3.client('omics')

response = omics.delete_workflow(
   id='1234567'
)
```

Verify the workflow status

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 includes workflow details, including the status, as shown.

```
{
    "arn": "arn:aws:omics:us-west-2:....",
    "creationTime": "2022-07-06T00:27:05.542459"
    "id": "1234567",
    "engine": "WDL",
    "status": "ACTIVE",
    "type": "PRIVATE",
    "main": "workflow-crambam.wdl",
    "name": "workflow_name",
    "storageType": "STATIC",
    "storageCapacity": "1200",
    "uuid": "64c9a39e-8302-cc45-0262-2ea7116d854f"
  }
```

You can start a run using this workflow after the status transitions to `ACTIVE`.

Referencing genome files from a workflow definition

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.

Workflow versioning in HealthOmics

If you need to make a changes to a workflow, you can create either a new workflow or a new workflow version. Versions are immutable, except for allowed configuration changes that don't impact the execution logic.

Workflow versions provide the following benefits:

- Versions form a logical group of workflows that are related. You can add a user-defined name to each workflow version to manage them more easily (especially for a workflow with a large number of versions).
- You can run multiple versions of a workflow at the same time.
- All versions of a workflow share the same workflow ID and base ARN, which can simplify pipeline management after you modify a workflow.
- Workflow versions provide the same level of data provenance as workflows. Versions are immutable, and HealthOmics creates a unique ARN for each workflow version. The version ARN includes the workflow ID and the version name, as shown in the following example:

  `arn:aws:omics:us-west-2:123456789012:workflow/1234567/version/myUniqueVersionName`
- If you own a shared workflow, you can update the workflow without disrupting the subscribers (who can continue to use the previous version). Subscribers can access all workflow versions. If you create a new version, you don't need to reshare the workflow.
- When you start a workflow run, you can specify the workflow version.
  - Users can choose to remain on a stable version for production runs, and try out the latest version for a test run.
  - Users can revert to the previous version of a workflow, if they encounter problems with the new version.
  - Subscribers of a shared workflow can choose which version to use.

Topics

- Default workflow version
- Create a workflow version
- Update a workflow version
- Delete a workflow version

Default workflow version

After you create one or more versions of a workflow, HealthOmics treats the original workflow as the default version. When you start a run, you can optionally specify a workflow version for the run. If you don't specify a version when you start a run, HealthOmics uses the default version.

In the console, HealthOmics indicates the original workflow with a **Default version** label. The console uses this label only after you create one or more workflow versions. The original workflow always remains the default version. You can't assign any other version to be the default.

You can't delete a workflow's default version if there are other versions associated with the workflow. For more information, see Delete a private workflow.

Create a workflow version

When you create a new version of a workflow, you need to specify the configuration values for the new version. It doesn't inherit any configuration values from the workflow.

When you create the version, provide a version name that is unique for this workflow. You cannot change the name after HealthOmics creates the version.

The version name must start with a letter or number and it can include upper-case and lower-case letters, numbers, hyphens, periods and underscores. The maximum length is 64 characters. For example, you can use a simple naming scheme, such as version1, version2, version3. You can also match your workflow versions with your own internal versioning conventions, such as 2.7.0, 2.7.1, 2.7.2.

Optionally, use the version description field to add notes about this version. For example: **Fix for syntax error in workflow definition**.

> **Note**
>
> Don’t include any personally identifiable information (PII) in the version name. Version names appear in the workflow version ARN.

HealthOmics assigns a unique ARN to the workflow version. The ARN is unique based on the combination of workflow ID and version name.

> **Warning**
>
> After you delete a workflow version, HealthOmics lets you reuse the version name for a different workflow version. Best practice is to not reuse version names. If you do reuse a name, the workflow and each version have a unique UUID that you can use for provenance.

Topics

- Create a workflow version using the console
- Create a workflow version using the CLI
- Create a workflow version using an SDK
- Verify the status of a workflow version

Create a workflow version using the console

Steps to create a workflow version

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose the workflow for the new version.
4. On the **Workflow details** page, choose **Create new version**.
5. On the **Create version** page, provide the following information:
    1. **Version name**: Enter a name for the workflow version that is unique across the workflow.
    2. **Version description** (optional): You can use the description field to add notes about this version.
6. In the **Workflow definition** panel, provide the following information:
    1. **Workflow language** (optional): Select the specification language for the workflow version. Otherwise, HealthOmics determines the language from the workflow definition.
    2. For **Workflow definition source**, choose to import the definition folder from a Git-based repository, an Amazon S3 location, or from a local drive.
        a. For **Import from a repository service**:

> **Note**
>
> HealthOmics supports public and private repositories for GitHub, GitLab, Bitbucket, GitHub self-managed, GitLab self-managed.

        i. Choose a **Connection** to connect your AWS resources to the external repository. To create a connection, see Connect with external code repositories.

> **Note**
>
> Customers in the TLV region need to create a connection in the IAD (us-east-1) region to create a workflow.

ii. In **Full repository ID**, enter your repository ID as user-name/repo-name. Verify you have access to the files in this repository.

iii. In **Source reference** (optional), enter a repository source reference (branch, tag, or commit ID). HealthOmics uses the default branch if no source reference is specified.

iv. In **Exclude file patterns**, enter the file patterns to exclude specific folders, files, or extensions. This helps manage data size when importing repository files. There is a max of 50 patterns, and the patters must follow the glob pattern syntax. For example:

A. `tests/`

B. `*.jpeg`

C. `large_data.zip`

b. For **Select definition folder from S3**:

i. Enter the Amazon S3 location that contains the zipped workflow definition folder. The Amazon S3 bucket must be in the same region as the workflow.

ii. If your account doesn't own the Amazon S3 bucket, enter the bucket owner's AWS account ID in the **S3 bucket owner's account ID**. This information is required so that HealthOmics can verify the bucket ownership.

c. For **Select definition folder from a local source**:

i. Enter the local drive location of the zipped workflow definition folder.

3. **Main workflow definition file path** (optional): Enter the file path from the zipped workflow definition folder or repository to the `main` file. This parameter is not required if there is only one file in the workflow definition folder, or if the main file is named "main".

7. In the **README file** (optional) panel, select the **Source of the README file** and provide the following information:

- For **Import from a repository service**, in **README file path**, enter the path to the README file within the repository.
- For **Select file from S3**, in **README file in S3**, enter the Amazon S3 URI for the README file.

- For **Select file from a local source**: in **README - optional**, chose **Choose file** to select the markdown (.md) file to upload.

8. In the **Default run storage configuration** panel, provide the default run storage type and capacity for runs that use this workflow:

    1. **Run storage type**: Choose whether to use static or dynamic storage as the default for the temporary run storage. The default is static storage.
    2. **Run storage capacity** (optional): For static run storage type, you can enter the default amount of run storage required for this workflow. The default value for this parameter is 1200 GiB. You can override these default values when you start a run.

9. **Tags** (optional): You can associate up to 50 tags with this workflow version.
10. Choose **Next**.
11. On the **Add workflow parameters** (optional) page, select the **Parameter source**:

    1. For **Parse from workflow definition file**, HealthOmics will automatically parse the workflow parameters from the workflow definition file.
    2. For **Provide parameter template from Git repository**, use the path to the parameter template file from your repository.
    3. For **Select JSON file from local source**, upload a JSON file from a local source that specifies the parameters.
    4. For **Manually enter workflow parameters**, manually enter parameter names and descriptions.

12. In the **Parameter preview** panel, you can review or change the parameters for this workflow version. If you restore the JSON file, you lose any local changes that you made.
13. On the **Container URI remapping** page, in the **Mapping rules** panel, you can define URI mapping rules for your workflow.

    For **Source of mapping file**, select one of the following options:

    - **None** – No mapping rules required.
    - **Select JSON file from S3** – Specify the S3 location for the mapping file.
    - **Select JSON file from a local source** – Specify the mapping file location on your local device.
    - **Manually enter mappings** – enter the registry mappings and image mappings in the **Mappings** panel.

14. The console displays the **Mappings** panel. If you chose a mapping source file, the console displays the values from the file.

    a. In **Registry mappings**, you can edit the mappings or add mappings (maximum of 20 registry mappings).

       Each registry mapping contains the following fields:

       - **Upstream registry URL** – The URI of the upstream registry.
       - **ECR repository prefix** – The repository prefix to use in the Amazon ECR private repository.
       - (Optional) **Upstream repository prefix** – The prefix of the repository in the upstream registry.
       - (Optional) **ECR account ID** – Account ID of the account that owns the upstream container image.

    b. In **Image mappings**, you can edit the image mappings or add mappings (maximum of 100 image mappings).

       Each image mapping contains the following fields:

       - **Source image** – Specifies the URI of the source image in the upstream registry.
       - **Destination image** – Specifies the URI of the corresponding image in the private Amazon ECR registry.

15. Choose **Next**.
16. Review the version configuration, then choose **Create version**.

When the version is created, the console returns to the workflow detail page and displays the new version in the **Workflows and versions** table.

Create a workflow version using the CLI

You can create a workflow version using the `CreateWorkflowVersion` API operation. For optional parameters, HealthOmics uses the following defaults:

Parameter	Default
Engine	Determined from the workflow definition

Parameter	Default
Storage type	STATIC
Storage capacity (for static storage)	1200 GiB
Main	Determined based on the contents of the workflow definition folder. For details, see HealthOmics workflow definition requirements.
Accelerators	none
Tags	none

The following CLI example creates a workflow version with static storage as the default run storage:

```
aws omics create-workflow-version \
--workflow-id 1234567  \
--version-name "my_version" \
--engine WDL \
--definition-zip fileb://workflow-crambam.zip \
--description "my version description" \
--main file://workflow-params.json \
--parameter-template file://workflow-params.json \
--storage-type='STATIC'   \
--storage-capacity 1200   \
--tags example123=string  \
--accelerators GPU
```

If your workflow definition file is located in an Amazon S3 folder, enter the location using the `definition-uri` parameter instead of `definition-zip`. For more information, see CreateWorkflowVersion in the AWS HealthOmics API Reference.

You receive the following response to the `create-workflow-version` request.

```
{
  "workflowId": "1234567",
  "versionName": "my_version",
```

```
  "arn": "arn:aws:omics:us-west-2:123456789012:workflow/1234567/version/3",
  "status": "ACTIVE",
  "tags": {
    "environment": "production",
    "owner": "team-alpha"
  },
  "uuid": "0ac9a563-355c-fc7a-1b47-a115167af8a2"
}
```

Create a workflow version using an SDK

You can create a workflow using one of the SDKs.

The following example shows how to create a workflow version using the Python SDK

```
import boto3

omics = boto3.client('omics')

with open('definition.zip', 'rb') as f:
   definition = f.read()

response = omics.create_workflow_version(
   workflowId='1234567',
   versionName='my_version',
   requestId='my_request_1'
   definitionZip=definition,
   parameterTemplate={ ... }
)
```

Verify the status of a workflow version

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

```
aws omics get-workflow-version
--workflow-id 9876543
--version-name "my_version"
```

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

```
{
```

```
  "workflowId": "1234567",
  "versionName": "3.0.0",
  "arn": "arn:aws:omics:us-west-2:123456789012:workflow/1234567/version/3.0.0",
  "status": "ACTIVE",
  "description": ...
  "uuid": "0ac9a563-355c-fc7a-1b47-a115167af8a2"
}
```

Before you can start a run with this workflow version, the status must transition to `ACTIVE`.

Update a workflow version

You can update the description and the default run storage configuration for a private workflow version. To change any other information in the workflow version, create a new version.

Topics

- Update a workflow version using the console
- Update a workflow version using the CLI
- Update a workflow version using an SDK

Update a workflow version using the console

To update a workflow version

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose the workflow.
4. On the **Workflow** page, choose the workflow version to update and choose **Edit selected** from the **Actions** list.
   - If you choose the default version, the console opens the **Edit workflow** page. For more information, see Update a private workflow.
   - If you choose a user-defined version, the console opens the **Edit version** page.
5. On the **Edit version** page, provide the following information
   - **Version description** (optional) - A description of this version.
6. In the **Default run storage configuration** panel, provide the following default values for runs that use this workflow version. You can override the default values when you start a run:

- For **Run storage type**, select **Static** or **Dynamic**.
- For static run storage, select the default amount of **Run storage capacity** for runs that use this workflow version. The default value for this parameter is 1200 GiB.

7. Choose **Save changes**.

The console returns to the workflow detail page and displays a page banner with the updated workflow version.

Update a workflow version using the CLI

You can update parameters for a workflow version using the following CLI command. The combination of workflow ID and version name uniquely identifies the version.

```
aws omics update-workflow-version
--workflow-id 1234567
--version-name "my_version"
--storage-type 'STATIC'
--storage-capacity 2400
--description "version description"
```

You receive no response to the `update-workflow-version` request.

Update a workflow version using an SDK

You can update a workflow version using one of the SDKs. The following python SDK example shows how to update the storage type and description for a workflow version.

```
import boto3

omics = boto3.client('omics')

response = omics.update_workflow_version(
   workflowID=1234567,
   versionName='3.0.0',
   storageType='DYNAMIC',
   description='new version description'
)
```

Delete a workflow version

You can delete a user-defined workflow version using the console, CLI, or one of the SDKs. Deleting a workflow version doesn't affect any ongoing runs that are using the workflow version.

You can't delete the Default workflow version. You delete all the user-defined versions, then delete the workflow.

Topics

- Delete a workflow version using the console
- Delete a workflow version using the CLI
- Delete a workflow version using an SDK

Delete a workflow version using the console

To delete a workflow version

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Private workflows**.
3. On the **Private workflows** page, choose the workflow.
4. On the **Workflow** page, choose the workflow version to delete and choose **Delete selected** from the **Actions** list.
5. In the **Delete workflow version** modal, enter "confirm" to confirm the deletion.
6. Choose **Delete**.

The console displays a page banner with the deleted workflow version.

Delete a workflow version using the CLI

You can delete a user-defined workflow version using the following CLI command. The combination of workflow ID and version name uniquely identifies the version.

```
aws omics delete-workflow-version
--workflow-id 9876543
--version-name "my_version"
```

You receive no response to the `delete-workflow-version` request.

Delete a workflow version using an SDK

You can delete a workflow using one of the SDKs.

The following example shows how to delete a workflow using the Python SDK.

```
import boto3

omics = boto3.client('omics')

response = omics.delete_workflow_version(
   workflowID=1234567,
   versionName='3.0.0'
)
```

Using HealthOmics runs

After you create a workflow, you can start runs using the workflow.

When you start a run, HealthOmics allocates temporary run storage for the workflow engine to use during the run. To ensure data isolation and security, HealthOmics provisions the storage at the start of each run, and deprovisions it at the end of the run.

HealthOmics provides several quotas related to workflow runs and tasks. Default values are intentionally conservative, to help you avoid unexpected cost overruns. You can request an increase in these quotas. For more information, see HealthOmics service quotas.

When you start a run, HealthOmics assigns a run ID and a run uuid to the run. Runs in an account have unique run IDs. However, HealthOmics reuses deleted run IDs, so a run and a deleted run can have the same run ID. Also, it's rare but possible for a shared workflow to have the same run ID as a run in your account.

The **run uuid** is a Globally Unique Identifier (guid) that you can use to identify runs across accounts or to distinguish between two runs in your account that have the same run ID.

> **Note**
>
> For data provenance purposes, we recommend that you use the **run uuid** to uniquely identify runs. The **run uuid** is also the best identifier to link to your internal lab information management system (LIMs) or sample tracking system.

You can use Amazon Q CLI to optimize your runs and analyze run performance. For more information, see Example prompts for Amazon Q CLI and the HealthOmics Agentic generative AI tutorial on GitHub.

Topics

- Run storage types in HealthOmics workflows
- Run retention mode for HealthOmics runs
- HealthOmics run inputs
- Run lifecycle in a HealthOmics workflow
- HealthOmics run outputs
- Run failure reasons
- Task lifecycle in a HealthOmics run
- Run optimization for a private HealthOmics workflow
- Run operations in HealthOmics

Run storage types in HealthOmics workflows

When you start a run, HealthOmics allocates temporary run storage for the workflow engine to use during the run. HealthOmics provides the temporary run storage as a file system.

For a given workflow or workflow run, you can choose dynamic or static run storage. By default, HealthOmics provides DYNAMIC run storage.

> **Note**
>
> Run storage usage incurs charges to your account. For pricing information about static and dynamic run storage, see *HealthOmics pricing*.

The following sections provide information to consider when deciding which run storage type to use.

Dynamic run storage

We recommend using dynamic run storage for most runs, including runs that require faster start times, runs where you don't know the storage needs in advance, and for iterative development testing cycles.

You don’t need to estimate the required storage or throughput for the run. HealthOmics dynamically scales the storage size up or down, based on file system utilization during the run. HealthOmics also dynamically scales throughput based on the workflow's needs. 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 most workflows and is also an advantage during development/test cycles.

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.

For dynamic run storage, the run provisions a filesystem that uses NFS protocol. NFS treats CREATE, DELETE, and RENAME file operations as non-idempotent, which may occasionally lead to race conditions for these operations that your code needs to handle gracefully. For example, your code should not fail if it tries to delete a file that does not exist. Before adopting dynamic run storage, we recommend adjusting your workflow code to make it resilient to non-idempotent file operations. See Code examples for safe handling of non-idempotent operations.

Code examples for safe handling of non-idempotent operations

The following python example shows how to delete a file without failing if the file does not exist.

```
import os
import errno

def remove_file(file_path):
    try:
        os.remove(file_path)
    except OSError as e:
        # If the error is "No such file or directory", ignore it (or log it)
        if e.errno != errno.ENOENT:
            # Otherwise, raise the error
            raise

# Example usage
remove_file("myfile")
```

The following examples use the Bash shell. To safely remove a file even if it doesn't exist, use:

```
rm -f my_file
```

To safely move (rename) a file, run the move command only if the file `old_name` exists in the current directory.

```
[ -f old_name ] && mv old_name new_name
```

For creating a directory, use the following command:

```
mkdir -p mydir/subdir/
```

Static run storage

For static run storage, the run provisions a filesystem that uses the Lustre protocol. This protocol is resilient to non-idempotent file operations by default. You do not need to adjust your workflow code to handle non-idempotent file operations.

HealthOmics allocates a fixed amount of run storage. You specify this value when you start the run. The default run storage is 1200 GiB, if you don't specify a value. When you specify a value for 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.

For static run storage, HealthOmics provisions the following throughput values:

- Baseline throughput of 200 MB/s per TiB of storage capacity provisioned.
- Burst throughput up to 1300 MB/s per TiB of storage capacity provisioned.

If the specified storage size is too low, the run fails with an **Out of storage for file system** error. Static run storage is a good fit for predictable workflows with known storage requirements.

Static run storage is suitable for large, bursty workloads with high task concurrency (for example, a large volume of RNASeq samples processed in parallel). It provides higher file system throughput per GiB and lower cost per GiB than dynamic run storage.

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.

Run retention mode for HealthOmics runs

After a run completes, HealthOmics archives the run metadata to CloudWatch. By default, CloudWatch keeps the run data indefinitely, unless you change the CloudWatch retention policy. Run outputs are also stored in Amazon S3 until you delete them.

One of the adjustable HealthOmics service quotas is the **maximum number of runs (active and inactive)** in a region. HealthOmics retains run metatdata for up to this number of runs for use by the console and API operations (ListRuns and GetRun). When you start a run, you can set the run retention mode parameter to indicate the retention behavior for the run. The parameter supports the values REMOVE and RETAIN.

For a new run with retention mode set to REMOVE, if HealthOmics tries to add the run after it has already saved the maximum number of runs, it automatically removes the metadata for the oldest run that has set REMOVE mode. This removal doesn't affect the data stored in CloudWatch or Amazon S3.

RETAIN is the default value for run retention mode. For runs in this mode, the system doesn't delete the run metadata. If HealthOmics reaches the maximum number of runs, all set to RETAIN, you won't be able to create additional runs until you delete some runs.

If you're planning to run a batch of more than the maximum number of runs at the same time, make sure to set the run retention mode to REMOVE. Otherwise, the batch fails when HealthOmics tries to start the next run after the maximum.

Additional considerations for using REMOVE retention mode:

- When you first start using REMOVE as the retention mode, consider deleting one or more runs that use RETAIN mode, to free up slots. As you start additional REMOVE runs, the automatic removal takes over, so enough slots are available for new runs.
- If you want to re-run an archived run (or a set of runs), use the HealthOmics rerun CLI tool. For more information and examples of how to use this tool, see Omics rerun in the HealthOmics tools GitHub repository.
- We recommend that you configure a unique name for each run. After HealthOmics removes a run, you can't use the console or API to find the run name or run ID. However, you can use CloudWatch to search for the run name, so use unique names to get the best search results.
- You can use the CloudWatch **start-query** command to get information about an archived run. If the run name isn't unique, the query may return multiple manifests. The start-time and end-time parameters define the time range for the search.

```
aws logs start-query  \
    --log-group-name "/aws/omics/WorkflowLog" \
    --query-string 'filter @logStream like "manifest" and @message like "myRunName"'
 \
    --end-time <END-EPOCH-TIME> --start-time <START-EPOCH-TIME>
```

The **start-query** command returns a query ID. Passing the query ID to the **get-query-results** command returns the query results.

```
aws logs get-query-results --query-id QueryId
```

HealthOmics run inputs

If the workflow definition specifies input files for the workflow or workflow tasks, HealthOmics stages the files to a scratch volume that's dedicated to the workflow run. These input files are read-only, which prevents tasks from modifying potential inputs to other tasks in the workflow. For directory imports, the directories are also read-only.

Many genomics applications assume that index files are co-located with the sequence files (such as a companion `bai` file for a `bam` file). To include index files, specify them as task inputs in the workflow definition.

Topics

- Managing run parameters size
- Amazon S3 input parameter formats
- Amazon S3 input archive states

Managing run parameters size

When you start a run, you specify run inputs in the run parameters JSON object or file. You can specify up to 50 KB of run parameters for the workflow. You can use the following techniques to remain within this size constraint:

- **Use directory imports**

  To specify a large number of input files, specify one parameter as the Amazon S3 location that contains all the files, rather than specifying a parameter for each file location. For more information, see the next topic (Amazon S3 input parameter formats).

- **Use a sample sheet**

  A sample sheet is a CSV or TSV file with one column for the fastq.gz address (or two for paired read) and additional columns for metadata such as sample names. You specify the sample sheet as a run input parameter instead of a parameter for each input file.

  Your workflow defines how your sample sheet maps to data structures in the workflow. While you could write code for sample sheets in WDL and CWL, they're more common in NextFlow. For an example, see sample sheet on the **nf-core** GitHub site.

Amazon S3 input parameter formats

For an input parameter that accepts an Amazon S3 location, the parameter can specify the location of one file or a whole directory of files. Using a directory has the following advantages:

- Convenience – You specify the directory name as the parameter. You don't list each file name.
- Compactness – The input parameter maximum file size is 50 KB. If you provide a long list of input file names, you can exceed this maximum.

Amazon S3 is a flat object-storage system, so it doesn't support directories. You group files into a "directory" by giving each file the same object key prefix. For more information about Amazon S3 object key prefixes, see Organizing objects using prefixes.

HealthOmics interprets the input parameter value as follows:

- If the Amazon S3 location doesn't end with a forward slash or use the glob pattern, HealthOmics expects the parameter value to be the key for one Amazon S3 object.

  For example, you specify `s3://myfiles/runs/inputs/a/file1.fastq` to input file1.fastq
- If the Amazon S3 location ends with a forward slash, HealthOmics interprets the parameter value as an Amazon S3 prefix. It loads all the Amazon S3 objects with that prefix.

  For example, you can specify `s3://myfiles/runs/inputs/a/` to load all objects whose keys start with this prefix.
- For Nextflow, HealthOmics paritally supports the glob pattern for Amazon S3 URIs in input parameters.

  For example, you can specify "`s3://myfiles/runs/inputs/a/*.gz`" to input all .gz files whose keys start with this prefix.

Nextflow Handling of Glob pattern in Amazon S3 inputs

Glob Pattern	HealthOmics Match Behavior	Notes
s3://bucket/directory/*.txt	Matches all `.txt` objects at any depth under the prefix s3://bucket/directory/. For example, matches s3://bucket/directory/abc.txt or s3://bucket/directory/subDir/123.txt etc.	
s3://bucket/directory/**/*.txt	Matches all `.txt` objects at any depth under the prefix s3://bucket/directory/. For example, matches s3://bucket/directory/abc.txt or	In S3, `**` is equivalent to `*`.

Glob Pattern	HealthOmics Match Behavior	Notes
	s3://bucket/directory/subDir/123.txt etc.	
s3://bucket/directory/{a,b}.txt	s3://bucket/directory/a.txt, s3://bucket/directory/b.txt	
s3://bucket/directory/?.txt	Matches objects at the prefix root whose filename is a single character followed by `.txt`. For example, it matches s3://bucket/directory/a.txt but not s3://bucket/directory/someDir/a.txt or s3://bucket/directory/someDir/subDir/a.txt	
s3://bucket/directory/[0-9].txt	s3://bucket/directory/0.txt, s3://bucket/directory/1.txt, ... ,s3://bucket/directory/9.txt	
s3://bucket/directory/[0-9].txt	s3://bucket/directory/1.txt, s3://bucket/directory/2.txt, s3://bucket/directory/3.txt	
s3://bucket/directory/[0-9].txt	s3://bucket/directory/b.txt, s3://bucket/directory/c.txt, ... ,s3://bucket/directory/Y.txt	

Language-specific handling of double-slash in Amazon S3 inputs

HealthOmics retains the native engine behavior for each workflow engine when handling double-slashes in Amazon S3 URIs, so that you don't need to make any changes to your workflows when you migrate them to HealthOmics. The following sections describe how each engine handles various scenarios.

WDL

If the input parameter includes a double-slash in the middle or at the end of the URI, the WDL engine retains the double-slash.

Input parameter	Expected location
s3://myfiles/runs/inputs//file1.fastq	s3://myfiles/runs/inputs//file1.fastq
s3://myfiles/runs/inputs//	s3://myfiles/runs/inputs//

Nextflow

If the input parameter includes a double-slash in the middle of the URI, the Nextflow engine retains double-slash. For a double-slash at the end of the URI, the Nextflow engine resolves it to a single slash.

Input parameter	Expected location
s3://myfiles/runs/inputs//file1.fastq	s3://myfiles/runs/inputs//file1.fastq
s3://myfiles//runs/inputs//*.gz	s3://myfiles//runs/inputs//*.gz
s3://myfiles//runs/inputs//	s3://myfiles//runs/inputs/

CWL

If the input parameter includes a double-slash in the middle or at the end of the URI, the CWL engine retains the double-slash.

Input parameter	Expected location
s3://myfiles//runs/inputs//file1.fastq	s3://myfiles//runs/inputs//file1.fastq

Input parameter	Expected location	
s3://myfiles//runs/inputs//	s3://myfiles//runs/inputs//	

Amazon S3 input archive states

HealthOmics can retrieve Amazon S3 objects that S3 delivers in real time. For objects that are in the following archived storage states, **restore** the objects to make them available to HealthOmics:

- Flexible Retrieval or Deep Archive storage classes in Amazon S3 Glacier.
- Archived Access or Deep Archive Access tiers in Intelligent tiering.

For information about restoring objects, see Restoring an archived object in the *Amazon S3 User Guide*.

Run lifecycle in a HealthOmics workflow

You can track the progress of a run by monitoring the run status. HealthOmics updates the run status as a run proceeds through its lifecycle.

You can retrieve run status using any of the following methods:

- The HealthOmics console displays the status of each run on the **Runs** page.
- The **GetRun** API operation returns the current run status.
- You can monitor run status using EventBridge events. For more information, see Using EventBridge with AWS HealthOmics.

Topics

- Run status values
- Task Retries
- Pricing implications of run status

Run status values

When you start a run, HealthOmics sets the run status to **Pending**. As the run proceeds through its lifecycle, HealthOmics updates the status value to reflect its current progress.

> **Note**
>
> You don't incur charges during any run status other than Running. See the next section for details.

HealthOmics supports the following run status values:

Pending

The run is in the queue, waiting to start. Runs typically remain in Pending for a brief period before they start.

- Runs can remain in Pending for a longer time if you submit many jobs at the same time.
- Runs remain in Pending after your account reaches the maximum number of concurrent runs.
- A run remains in Pending if the run is part of a run group that has reached any of its resource maximum values.
- You can adjust run priorities so that specific queued runs start before others. For more information about run priority, see Run priority.

Starting

HealthOmics creates the run and provisions the resources required for the run (such as temporary run storage and the engine node).

- HealthOmics provisions temporary run storage at the start of the run, and deprovisions the run storage when the run is Stopping.

Running

A run remains in Running status during the import process, the processing of each task, and the export process.

- HealthOmics imports the input files to the temporary run storage file system. The input files are read-only, to prevent tasks from modifying the inputs to other tasks in a workflow.
- During file export, HealthOmics exports the output files from the run storage file system to the S3 location.
- HealthOmics delivers the run logs and task logs to CloudWatch in real time while the run status is Running. For more information, see Logs in CloudWatch.

Stopping

After completion of the export process, the run transitions to the Stopping status.

- HealthOmics deprovisions all resources (including the run storage file system and the engine node).

Completed

The run transitions to Completed after HealthOmics completes the resource deprovisioning.

- HealthOmics has completed all run tasks and exported the output data without error.
- The run outputs are available in the specified Amazon S3 URI output location. For WDL and CWL, HealthOmics generates a run output summary file, which provides information about the HealthOmics run outputs.
- The final run manifest logs and engine logs (if applicable) are available in CloudWatch.
- For runs that support task retries, a run with Completed status can include one or more tasks that failed. As long as a task retry succeeded for each failed task, HealthOmics transitions the run to Completed. HealthOmics assigns a new task ID to each retry, so the run includes task IDs for the failed attempts and the completed attempt.

Failed

HealthOmics encountered one or more errors and failed to complete all the run tasks.

- A failed run transitions through Stopping status while HealthOmics deprovisions the resources.

Cancelled

A user initiated a request to cancel the run.

- HealthOmics stops any running tasks and deprovisions all resources.
- HealthOmics doesn't export any run output data when a user cancels a run. You don't have access to any intermediate files for a cancelled run.
- Your account incurs charges for the tasks and resources that the run consumed during Running status before the cancellation.
- There are no charges if you cancel a run in Pending or Starting status.

Task Retries

HealthOmics supports task retries for tasks that fail because of service errors (5XX HTTP status codes).

If every task in the run eventually completes, even if they required retries, HealthOmics transitions the run to Completed. HealthOmics assigns a new task ID to each retry, so the run includes task IDs for the failed attempts and the completed attempt.

The default retry behavior depends on which definition language the workflow uses. The default for Nextflow is no retries. For WDL and CWL, HealthOmics attempts up to two retries of a failed task, but you can opt out of task retry for specific tasks or for all tasks in a workflow. Task retry is useful to address intermittent service errors. However, you might consider opting out a task that is idempotent.

For specific information about each workflow definition language, see the following topics:

- WDL – Configure task retry behavior in the workflow definition. See Configure WDL task retry behavior.
- Nextflow – Configure task retry behavior in the Nextflow config file or the workflow definition. See Configure Nextflow task retry behavior.
- CWL – Configure task retry behavior in the workflow definition. See Configure CWL task retry behavior.

Pricing implications of run status

Your account can incur charges while the run status is Running. You don't incur charges during any other run status. For example, there is no charge for resources when the run is Starting or Stopping.

A run with Running status has the following billing implications:

- Your account incurs charges for run storage file-system usage while the run status is Running. For information about the run storage types, See Run storage types in HealthOmics workflows.
- Your account incurs charges for running tasks, based on the compute and memory resources that you specified for each task in the workflow definition, and based on the task duration. For more information, see Compute and memory requirements for HealthOmics tasks.
- Each task has a minimum billing threshold of one minute. If you run a task for less than a minute, you incur a charge for the minimum one minute of usage. If possible, group small tasks together to optimize costs. Grouping tasks also reduces run time by avoiding the spin-up of multiple sequential tasks.

For additional information about HealthOmics pricing, see the HealthOmics Pricing.

HealthOmics run outputs

When a WDL or CWL run completes, the outputs include an output summary file (in JSON format) that lists all the outputs produced by the run. You can use the output summary file for these purposes:

- Programmatically determine the output files that the run generated.
- Validate that the run produced all the expected outputs.

Topics

- Run output summary for WDL
- Run output summary for CWL

Run output summary for WDL

When a WDL run completes, HealthOmics creates an output summary file named **output.json**.

For each output of the workflow, there is a corresponding key/value pair in the file.
The key contains the workflow name and output name in the following format: `WorkflowName.output_name`. For a file output, the value is an S3 URI pointing to the output location in S3 where the file is stored. For an Array[File] output, the value is an array of S3 URIs.

The following example shows the **output.json** file for a workflow named **BWAMappingWorkflow**.

```
{
  "BWAMappingWorkflow.bam_indexes": [
    "s3://omics-outputs/8886192/out/bam_indexes/0/
pbmc8k_S1_L007_R1_001.sorted.bam.bai",
    "s3://omics-outputs/8886192/out/bam_indexes/1/pbmc8k_S1_L008_R1_001.sorted.bam.bai"
  ],
  "BWAMappingWorkflow.mapping_stats": "s3://omics-outputs/8886192/out/mapping_stats/
genome_mapping_final_stats.txt",
  "BWAMappingWorkflow.merged_bam": "s3://omics-outputs/8886192/out/merged_bam/
genome_mapping.merged.bam",
  "BWAMappingWorkflow.merged_bam_index": "s3://omics-outputs/8886192/out/
merged_bam_index/genome_mapping.merged.bam.bai",
  "BWAMappingWorkflow.reference_index_tar": "s3://omics-outputs/8886192/out/
reference_index_tar/reference_index.tar",
```

```
  "BWAMappingWorkflow.sorted_bams": [
    "s3://omics-outputs/8886192/out/sorted_bams/0/pbmc8k_S1_L007_R1_001.sorted.bam",
    "s3://omics-outputs/8886192/out/sorted_bams/1/pbmc8k_S1_L008_R1_001.sorted.bam"
  ],
  "BWAMappingWorkflow.unmapped_bams": [
    "s3://omics-outputs/8886192/out/unmapped_bams/0/
pbmc8k_S1_L007_R1_001.unmapped.bam",
    "s3://omics-outputs/8886192/out/unmapped_bams/1/pbmc8k_S1_L008_R1_001.unmapped.bam"
  ]
}
```

If the workflow produces outputs with non-file types (such as String, Int, Float, or Bool), the field value is a JSON primitive. For example:

```
{
  "MyWorkflow.my_int_ouput": 1,
  "MyWorkflow.my_bool_output": false,
    ...
}
```

Run output summary for CWL

When a CWL run completes, HealthOmics creates an output summary file named **outputs.json** at the following location:

```
{my-S3outputpath}/{runId}/{run-uuid}/logs/outputs.json
```

The output summary file includes a list of outputs. Each output is a key/value pair, where the key is the name of the output. The value is an object that includes the following properties:

- location – The fully qualified path to the output file
- basename – The filename portion of the path
- class – The type of the output, which is typically File
- size – The size of the file in bytes

In the following example, the output.json file has a list of two output files.

```
{
  "example_output": {
    "location": "{my-S3outputpath}/{runId}/{run-uuid}/out/output.txt",
```

```
    "basename": "output.txt",
    "class": "File",
    "size": 13
  },
  "another_output": {
    "location": "{my-S3outputpath}/{runId}/{run-uuid}/out/metrics.json",
    "basename": "metrics.json",
    "class": "File",
    "size": 256
  }
}
```

Run failure reasons

If a run fails, use the GetRun API operation to retrieve the failure reason.

Review the failure reason to help you troubleshoot why the run failed. The following table lists each failure reason along with a description of the error.

Failure reason	Error description
ASSUME_ROLE_FAILED	HealthOmics doesn't have permission to assume the role. Specify the HealthOmics principal in the trust relationship for the role.
CANNOT_START_CONTAINER_ERROR	Unable to start workflow task: *name*, id: *ID* container using image: *image name*. Make sure that the image is valid and try again.
CANNOT_START_CONTAINER_SIZE_ERROR	Unable to start workflow task: *name*, id: *ID* container using image: *image name*. Make sure that the image size is less than 45 GiB (95 GiB for a GPU instance) and try again.
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.
EXPORT_FAILED	The export failed. Check that the output bucket exists and the run role has write permission to the bucket.

Failure reason	Error description
FILE_SYSTEM_OUT_OF_SPACE	The file system doesn't have enough space. Increase the file system size and run again.
IMAGE_VERIFICATION_FAILURE	Unable to verify image *image name*. To correct the issue, try pulling the image and then push it to your ECR repository again.
IMPORT_FAILED	The import failed. Check that the input file exists and the run role can access input.
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 Activating read sets in HealthOmics.
INPUT_URI_NOT_FOUND	The provided URI does not exist: *uri*. Check that the URI path exists and confirm that the role can access the object.
INSTANCE_RESERVATION_FAILED	There isn't enough instance capacity to complete the workflow run. Wait and try the workflow run again.
INVALID_ECR_IMAGE_URI	The Amazon ECR image URI structure isn't valid. Provide 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 *ID*.
INVALID_URI_INPUT	The URI structure isn't a valid *uri*. Check the URI structure and try again.
MODIFIED_INPUT_RESOURCE	The provided URI *uri* was modified after the run started. Retry the run.
OUT_OF_MEMORY_ERROR	The workflow task *ID* ran out of memory. Increase the memory value in the workflow definition and try the run again.

Failure reason	Error description
RUN_TASK_FAILED	The run failed because the task failed. To debug the task failure, use the **GetRunTask** API operation and the Amazon CloudWatch Logs stream.
RUN_TIMED_OUT	Run timeout after *number* minutes.
SERVICE_ERROR	There was a transient error in the service. Try the workflow run again.
TASK_TIMED_OUT	Task *id* timed out after *number* seconds.
UNSUPPORTED_INPUT_SIZE	The total input size is too high. Decrease the input size and try again.
WORKFLOW_RUN_FAILED	Workflow run failed. Review the CloudWatch Logs engine log stream: *ID* to debug the failure.
WORKFLOW_VER_VALIDATION_FAILED	HealthOmics doesn't support requested Nextflow version: *version* --. The latest supported version is *version*. Modify your Nextflow version to a supported version and try again.
UNSUPPORTED_GPU_INSTANCE_TYPE	The requested instance type is not supported in *Region*. Retry the run with a GPU instance type supported in this Region. Available instance types are *GPU instance types*.

Guidance for unresponsive runs

When developing new workflows, runs or specific tasks could become "stuck" or "hang" if there are issues with your code, and tasks fail to exit processes properly. This can be challenging to troubleshoot and catch, as it is normal for tasks to run for extended periods. To prevent and identify unresponsive runs, follow the suggested best practices in the following sections.

Best practices for preventing unresponsive runs

- Ensure you are closing all the files opened in your task code. Opening too many files can ocassionally lead to threading issues within the workflow engines.

- Background processes created by a workflow task should exit when the task exits. However, if a background process does not exit cleanly, you must explicitly shut down that process in your task code.
- Ensure your processes do not loop without exiting. This can cause an unresponsive run, and requires a change to your workflow definition code to resolve.
- Provide appropriate memory and CPU allocation to your tasks. Analyze the CloudWatch logs or use the Run Analyzer on successfully completed runs of your workflow to verify you have optimal compute allocation. Use the Run Analyzer `headroom` parameter to include additional headroom, ensuring processes have sufficient resources to complete. Include at least 5% headroom in allocated memory and CPU, to account for background operating system processes.
  - Additionally, increase the instance bandwidth size if the instance requires a higher throughput. Amazon EC2 instances with fewer than 16 vCPUs (size 4xl and smaller) can experience throughput bursting. For more information on Amazon EC2 instance throughput, see Amazon EC2 available instance bandwidth.
- Ensure you are using the correct file system size for your runs. For unresponsive runs that are using static run storage, consider increasing the static run storage allocation to enable higher IO throughput and storage capacity on the file system. Analyze the run manifest to see the maximum file system storage, use the Run Analyzer to determine if the file system allocation needs to be increased.

Best practices for catching unresponsive runs

- When developing new workflows, use a run group with the max run time limit set to catch runaway code. For instance, if a run should take 1 hour to complete, place it in a run group that times out after 2 or 3 hours (or a different time period based on your use case) to catch run-away jobs. Also, apply a buffer to account for variance in processing times.
- Set up a series of run groups with different maximum runtime limits. For instance, you could assign short runs to a run group that terminates the runs after a few hours, and a long runs group that terminates runs after a few days, based on your expected workflow duration.
- HealthOmics has a default maximum run duration service limit of 604,800 Seconds, or 7 days, which is adjustable through a request in the quotas tool. Only request a service limit increase of this quota if you have runs that approach a week in duration. If you have a mix of short and long runs and are not using run groups, consider putting the long-running runs in a separate account with a higher maximum run duration service limit.

- Inspect the CloudWatch logs for tasks that you suspect could be unresponsive. If a task normally outputs regular log statements and has not done so for an extended period, the task is likely stuck or frozen.

What to do if you encounter an unresponsive run

- Cancel the run to avoid incurring additional costs.
- Inspect the task logs to check if any processes failed to exit correctly.
- Inspect the engine logs to identify any abnormal engine behaviors.
- Compare the task and engine logs from the unresponsive run to those of identical, successfully completed runs. This can help identify any differences that may have caused the unresponsive behavior.
- If you are unable to determine the root cause, raise a support case and include the following:
  - ARN of the stuck run and ARN of an identical run that completed successfully.
  - Engine logs (available once the run has been cancelled or fails)
  - Task logs for the unresponsive task. We don't require task logs for all tasks in the workflow to troubleshoot.

Task lifecycle in a HealthOmics run

A *task* is a single process within a run. HealthOmics maps each task in your workflow to an omics computing instance type that best fits the task's required resources. You specify the required resources in the workflow definition. For more information, See Compute and memory requirements for HealthOmics tasks.

HealthOmics provides temporary run storage for the task to use. HealthOmics copies the task input files to the temporary run storage as read-only files. HealthOmics provides symbolic links so that the task can access the input files from the working directory. The task has access only to the files that you declare in the workflow definition file.

Task status values

You can track the progress of a task by monitoring the task status. When you start a run, HealthOmics sets the task status to **Pending** for each task in the run. When the task starts and progresses through its lifecycle, HealthOmics updates the status value to reflect its current progress.

You can retrieve task status using any of the following methods:

- The HealthOmics console displays the status of each task in a run on the **Run details** page.
- The **GetRunTask** API operation returns the task status.
- You can monitor task status using EventBridge events. For more information, see Using EventBridge with AWS HealthOmics.

You can retrieve the current status of a task using the **GetRunTask** API operation. The HealthOmics console displays the status for each task in a run on the **Run details** page.

HealthOmics supports the following task status values:

Pending

Your task is in the queue, waiting to start. Tasks stay in pending for a brief period before they start.

- Tasks remain in pending after your account has reached the maximum number of concurrent tasks.
- Tasks remain in pending if the run is part of a run group that has reached any of its resource maximum values.
- You can adjust run priorities so that specific queued runs and their tasks start before other queued runs. For more information about run priority, see Run priority

Starting

HealthOmics is creating the task and provisioning the resources required for the task, such as the workflow task node.

Running

The task status is Running while HealthOmics is processing the task.

Stopping

After completing the task processing and exporting the output data, the task transitions to Stopping.

- HealthOmics deprovisions the workflow task node.

Completed

HealthOmics has finished processing the task and has transferred the output data to the run storage file system.

Failed

HealthOmics encountered an error while processing the task and didn't complete it.

- The task transitions to Stopping status (HealthOmics deprovisions the resources) and then to Failed status.
- If the error is a service error (5XX HTTP status code), and the workflow supports retries for this task, HealthOmics attempts to process the task again. HealthOmics assigns a new task ID to the retry.

Cancelled

HealthOmics stops the task after a user-initiated request to cancel the run.

- The task transitions to Stopping status (HealthOmics deprovisions the resources) and then to Cancelled status.

Troubleshooting workflow tasks

The following are best practices and considerations for troubleshooting your tasks.

- 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 tasks 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.

Run optimization for a private HealthOmics workflow

You can optimize runs for total cost, total run time, or a combination of both. HealthOmics provides data and tools to help you with run optimization decisions. Run optimization doesn't apply to Ready2Run workflows, because you don't have any control over how the service manages resource provisioning for these workflows.

The first step is to understand the current task resource usage and cost for the tasks in the run, and then apply methods for optimizing the run cost and performance.

Topics

- Run Analyzer
- Determine run costs
- Determine run time usage
- Methods to optimize runs
- Impact of file size variance between runs
- Methods to optimize resource concurrency

Run Analyzer

HealthOmics provides an open source tool named Run Analyzer. This tool extracts task-level resource usage information for a run and suggests optimization opportunities for cost and run performance.

> **Note**
>
> Run analyzer estimates task costs and potential cost savings based on AWS list prices at the time you run the tool. Assess the optimization recommendations and implement those that make sense for your use cases. Test the optimizations that you adopt to make sure that they work for your run.

Run Analyzer performs the following tasks:

- Evaluates memory and compute bottlenecks.
- Identifies tasks that are over-provisioned for memory or CPU, and recommends new instance sizes that can reduce costs.

- Computes cost estimates for individual tasks and computes the potential cost savings if you apply the recommendations.
- Gives you a timeline view of tasks so you can verify the task dependencies and processing sequence. The timeline also helps you to identify long running tasks.
- Provides recommendations about the file-system size for the run storage.
- Shows you task provisioning times so that you can identify areas where large container loads may be slowing down provisioning time.
- The tool includes an input parameter (headroom) you can use to control the aggressiveness of the optimization recommendations.

The following sections include specific suggestions for using Run Analyzer to optimize runs.

Determine run costs

You can use the following methods and guidelines to determine run costs:

- To view the total run costs for a billing period, follow these steps:
  1. Open the Billing and Cost Management console and choose **Bills**.
  2. In **Charges by service**, expand **Omics**.
  3. Expand the **region**, then view the cost of all your runs itemized by omics instance type, run storage type, and Ready2Run workflow.
- To generate a cost report that includes information for each run, follow these steps:
  1. Open the Billing and Cost Management console and choose **Data Exports**.
  2. Choose **Create** to create a new data export.
  3. Enter an **Export name** for the data export. Keep the other fields at their default values to create a CUR (cost and usage) report.
  4. For **Time granularity**, select hourly or daily.
  5. Under **Data export storage settings**, perform these configuration steps:
     a. Configure an Amazon S3 bucket for the data export.
     b. For **File versioning**, select whether to overwrite the existing export file or create a new file each time.

The system generates the first report within the next 24 hours and generates subsequent reports once a day.

6. For more information about how to create the data export, see Creating data exports in the *AWS Data Exports User Guide*.

- You can tag your runs to monitor and optimize costs by category, such as by team or by project. If you use tags, follow these steps to view run costs by tag category:

  1. Open the Billing and Cost Management console and choose **Cost Explorer**.
  2. In **Report parameters > Group by**, chose **Tag** as the dimension. and select the desired **Tag** name.

- To see resource usage for tasks, view the run manifest logs in CloudWatch. For more information, see Monitoring HealthOmics with CloudWatch Logs.
- Use the Run Analyzer tool to extract task resource usage information for a run.

Determine run time usage

You can use the following methods to help you investigate run time usage:

- From the **Runs** page of the console, you can view the total run time for a run.
- From the **Run details** page, you can view the following items:
  - View the total run time for a run.
  - View the run time for each task in the run.
  - Choose one of the links to view the logs in Amazon S3, or to view the run logs or run manifest logs in CloudWatch.
- From the **Run tasks** list, choose the **View logs** link for a task to view the task logs in CloudWatch.
- The response to the `listRuns` API operation includes the run start time and stop time, so you can calculate the total run time.
- The Run Analyzer tool shows task durations on a timeline view. This tool provides a visual representation of the task processing sequence, which you can match with the expected order.

Methods to optimize runs

HealthOmics automatically provisions, manages, and optimizes resources that perform data staging (such as data imports and data exports). HealthOmics also starts and runs the workflow engine for your workflow. However, you can influence run start times, task start times, and overall task run time by setting various run configurations. Your overall approach to the workflow definition and design also impacts task run time. The following list describes factors that can affect run and task performance:

Run storage type

The run storage type has an impact on run performance and run provisioning time. Dynamic run storage provisions faster and never runs out of memory, because it scales dynamically with your run storage needs. Dynamic run storage is also a good fit for workflows in development, where you may often start and stop a workflow to troubleshoot issues.

Static run storage requires longer file system provisioning times, but can complete some runs faster, typically if the runs have high task concurrency or require greater than 9.6 TiB of file system capacity. Static run storage is well suited for long running workflows with high I/O requirements.

To help you evaluate the cost vs. performance of each run storage type for a given run, you can try A/B testing to see which run storage type delivers better performance. Also, consider using dynamic run storage for your development cycles, then use static run storage for production runs at scale.

For more information about run storage types Run storage types in HealthOmics workflows

Over-provision run static storage

If your workflow task computation is constrained by I/O, consider over-provisioning the static run storage. Storage cost increases with its size, but maximum throughput of the file system also increases. If an expensive compute task is experiencing I/O bottlenecks, increasing the file system size to reduce the task run time may reduce the overall cost.

Reduce container image sizes

When each task starts, HealthOmics loads the container you specified for the task. Larger containers take longer to load. Optimize your containers to be as small as possible to improve the efficiency of launching new tasks. If you add large datasets to your containers, consider storing the datasets in S3 and having your workflow import the data from S3. For the maximum container sizes that HealthOmics supports, see HealthOmics workflow fixed size quotas.

Task size

You can combine small, sequential tasks into a single task to save task provisioning time. Also, HealthOmics has a one-minute minimum task duration charge, so combining tasks may reduce costs. Within the combined task, you may be able to use Unix pipes to avoid the I/O cost of serializing and deserializing files.

File compression

Avoid overly compressing workflow intermediate files. Most genomics formats use “gzip” or “block gzip” compression. Decompressing the task input file and recompressing the task output file can consume a large percentage of the overall task CPU usage. Some genomics applications allow you to set the compression level when serializing outputs. By reducing the level of compression, you can reduce CPU time, although larger files increase the time spent writing to disk. Depending on the task and the application, you can find the optimal compression level for intermediate files that result in the shortest run time. We recommend that you start by targeting the tasks with the largest output files. A compression level of 2 works well for several scenarios. You can start with this level for your use-case, and compare results by trying other compression levels.

Thread count

If you specify threads in your task definition, set the number of threads to the same value as the number of requested vCPUs.

Specify compute and memory

If you don't specify memory or compute resources in your task, HealthOmics assigns the smallest instance type (`omics.c.large`) as the default . Explicitly declare your memory and compute requirements if you want HealthOmics to assign a larger instance type.

HealthOmics allocates the number of vCPUs, memory, and GPU resources that you request. For instance, if you ask for 15vCPUs and 33GiB, HealthOmics allocates an omics.m.4xl instance (16vCPUs, 64GB) for your task, but your task can use only 15 vCPUs and 33GiB. Therefore, we recommend that you request vCPUs and memory resources that match an omics instance.

Batch multiple samples into one run

Because file system provisioning takes time at the start of the run, you can save on provisioning time by batching multiple samples into the same run. Consider the following factors before deciding on this approach:

- A single bad sample can cause a workflow to fail, so batching samples could increase the number of failed workflows. If you aren't confident that your workflow will succeed most of the time, one run per sample could be a better approach.
- HealthOmics allocates one run storage file system for the whole workflow. For a batch of samples, make sure to specify a large enough amount of run storage to process all the samples.
- There is a maximum amount of run storage per workflow, so that may constrain the number of samples you can add to the batch.
- The minimum run storage size is 1.2 TiB, so batching may reduce costs if the workflow uses much less storage than the minimum for each sample.
- Run storage can handle multiple simultaneous connections, so having multiple tasks using the same run storage shouldn't cause I/O bottlenecks.
- Each run has its own set of tags. If you tag workflows with information for budgeting or tracking, it may be better to use separate runs.
- IAM roles apply to the whole run. Each user has access to all the data for a batch of samples. Having separating workflows gives you the ability to use more fine-grained permissions.
- HealthOmics sets account-level quotas for maximum number of concurrent workflows and maximum number of concurrent tasks in a workflow. For information on how to request an increase for these quotas, see HealthOmics service quotas.

Use parameters for container images

Parameterize your container images rather than embedding their URIs in the workflow. Wheb they are run parameters, HealthOmics validates that the run has access to your containers before the run starts. Otherwise, the task fails during the run, when you have incurred charges for any completed tasks. Also, because these are parameterized inputs, HealthOmics generates a checksum in the run manifest, which improves the run provenance.

Use a linter

Use a linter to find common workflow errors before you run a new workflow. For more information, see Workflow linters in HealthOmics.

Use EventBridge to flag issues

Use EventBridge customized alerts to catch anomalies that are specific to your business logic.

Use sequence stores

Consider using a sequence store for your source data to save on storage costs. For more information, see the Store omics data cost-effectively at any scale with HealthOmics blog post.

Impact of file size variance between runs

Users often design and test runs using a small set of testing data, then encounter a wide variety of data with significant file-size variance in production runs. Make sure you account for this variance when you optimize the run.

The following list describes recommendations for optimization where there is significant variance in file sizes:

Vary file sizes in your testing data

Try to use testing data during development that has a representative amount of variance.

Use Run Analyzer

Use the Run Analyzer tool across a variety of samples to account for variance in data sizes.

You can use the run analyzer to understand variance between runs in your production data samples. Use `--batch` mode in Run Analyzer to generate statistics for a batch of runs and analyze the maximum compute resources required to handle outliers in your data sets.

For example, you can give run analyzer a full flow cell of data in batch mode to understand peak vCPU and memory utilization for the full flow cell.

Reduce size variance of the input datasets

If you see high variance in sample sizes, you can bifurcate samples upstream of HealthOmics and select different file system sizes for each batch to save on run storage costs.

In WDL, use the `size` function to bifurcate resource allocation for individual tasks for large versus small samples. Apply this strategy to your most expensive tasks to have the most impact.

In Nextflow, use conditional resources for tiering resource allocation based on file size or file name. For more information, see Conditional process resources on the Nextflow GitHub site.

Don't optimize too soon

Finalize your workflow code and logic before investing in significant performance tuning efforts. Changing your code can have significant impacts on required resources. If you optimize

a run too soon in the development process, you may over-optimize or you may need to optimize again if the workflow definition changes later.

Re-run the Run Analyzer tool periodically

If you make changes to your workflow definition over time or if your sample variance changes, periodically run the Run Analyzer tool to help you made additional optimizations.

Methods to optimize resource concurrency

HealthOmics provides the following capabilities to help you control and manage costs when processing runs at scale:

- Use run groups to control your costs and resource usage. You can set maximum values in the run group for number of concurrent runs, vCPUs, GPUs, and total run time per task. If separate teams or groups use the same account, you can create a separate run group for each team. You can control resource usage and costs per team and by configuring the run group maximum values. For more information, see Using HealthOmics run groups.
- During development, you can configure a separate run group with lower maximum values to catch runaway tasks.
- Service Quotas also help to protect your account from excessive resource requests. For information about Service Quotas, including how to request quota value increases, see HealthOmics service quotas

Run operations in HealthOmics

You can start, rerun, clone, cancel or delete a run:

- **Start** – HealthOmics creates a new run using the configuration settings you specify and then starts the run.
- **Rerun** – HealthOmics creates a new run that's a duplicate of the run that you specify. You can rerun a deleted run using the HealthOmics **rerun** tool.
- **Clone** – You can clone an existing run using the console. The console opens the **Clone run** page and prefills the configuration fields using the values from the existing run. You can modify the values as required and start the cloned run.
- **Cancel** – You can cancel a run that hasn't completed yet. When you cancel a run, HealthOmics doesn't save any of the run outputs.

- **Delete** – You can delete completed runs manually, or set the run retention mode for HealthOmics to delete the oldest runs automatically. For more information about retention mode, see the section called "Run retention modes".

Topics

- Start a run in HealthOmics
- Rerun a run in HealthOmics
- Clone a run in HealthOmics
- Cancel a run in HealthOmics
- Delete a run in HealthOmics

Start a run in HealthOmics

When you start a run, you specify the resources that HealthOmics allocates for use during the run.

Specify the run storage type and storage amount (for static storage). To ensure data isolation and security, HealthOmics provisions the storage at the start of each run, and deprovisions it at the end of the run. For additional information, see Run storage types in HealthOmics workflows.

Specify an Amazon S3 location for the output files. If you run a high volume of workflows concurrently, use separate Amazon S3 output URIs for each workflow to avoid bucket throttling. For more information, see Organizing objects using prefixes in the *Amazon S3 User Guide* and Scale Storage Connections Horizontally in the *Optimizing Amazon S3 Performance* whitepaper.

You can also specify 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.

If a workflow has one or more versions, you can specify a version when you start the run. If you don't specify a version, HealthOmics starts the default workflow version.

When using the HealthOmics API, you can provide a unique request ID for each run. The request ID is an idempotency token that HealthOmics uses to identify duplicate requests. and starts the run only once.

> **Note**
>
> You specify an IAM service role when you start a run. Optionally, the console can create the service role for you. For more information, see Service roles for AWS HealthOmics.

Topics

- HealthOmics run parameters
- Starting a run using the console
- Starting a run using the API
- Get information about a run

HealthOmics run parameters

When you start a run, you specify run inputs in the run parameters JSON file or you can enter the parameter values inline. For information about managing the size of the run parameters JSON file, see Managing run parameters size.

HealthOmics supports the following JSON types for parameter values.

JSON type	Example key and value	Notes
boolean	"b":true	Value is not in quotes, and all lowercase.
integer	"i":7	Value is not in quotes.
number	"f":42.3	Value is not in quotes.
string	"s":"characters"	Value is in quotes. Use string type for text values and URIs. The URI target must be the expected input type.
array	"a":[1,2,3]	Value is not in quotes. Array members must each have

JSON type	Example key and value	Notes
		the type defined by the input parameter.
object	"o":{"left":"a", "right":1}	In WDL, object maps to WDL Pair, Map, or Struct

Starting a run using the console

To start a run

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, choose **Start run**.
4. In the **Run details** panel, provide the following information

   - **Workflow source** - Choose **Owned workflow** or **Shared workflow**.
   - **Workflow ID** - The workflow ID associated with this run.
   - **Workflow version** (Optional) - Select a workflow version to use for this run. If you don't select a version, the run uses the workflow default version.
   - **Run name** - A distinctive name for this run.
   - **Run priority** (Optional) - The priority of this run. Higher numbers specify a higher priority, and the highest priority tasks are run first.
   - **Run storage type** - Specify the storage type here to override the default run storage type specified for the workflow. Static storage allocates a fixed amount of storage for the run. Dynamic storage scales up and down as required for each task in the run.
   - **Run storage capacity** - For static run storage, specify the amount of storage needed for the run. This entry overrides the default run storage amount specified for the workflow.
   - **Select S3 output destination** - The S3 location where the run outputs will be saved.
   - **Output bucket owner's account ID** (Optional) - If your account doesn't own the output bucket, enter the bucket owner's AWS account ID. This information is required so that HealthOmics can verify the bucket ownership.

- **Run metadata retention mode** - Choose whether to retain the metadata for all runs or have the system remove the oldest run metadata when your account reaches the maximum number of runs. For more information, see Run retention mode for HealthOmics runs.

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 run parameters. You can either upload a JSON file that specifies the parameters or manually enter the values.
9. Choose **Next**.
10. In the **Run group** panel, you can optionally specify a run group for this run. For more information, see Using HealthOmics run groups.
11. In the **Run cache** panel, you can optionally specify a run cache for this run. For more information, see Configuring a run with run cache using the console.
12. Choose **Review and start run**.
13. After you review the run configuration, choose **Start run**.

Starting a run using the API

Use the **start-run** API operation to create and start a run.

The following example specifies the workflow ID and service role. This example sets the retention mode to REMOVE. For more information about retention mode, see Run retention mode for HealthOmics runs.

```
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 `outputUri` can be used to track where output data is written.

```
{
    "arn": "arn:aws:omics:us-west-2:....:run/1234567",
    "id": "123456789",
```

```
    "uuid":"96c57683-74bf-9d6d-ae7e-f09b097db14a",
    "outputUri":"s3://bucket/folder/8405154/96c57683-74bf-9d6d-ae7e-f09b097db14a"
    "status": "PENDING"
}
```

Include a parameter file

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 `--parameters file://<input_file.json>` to your `start-run` request. For more information about run parameters, see HealthOmics run inputs.

Provide a request ID

You can provide a unique `requestId` for each run. The request ID is an idempotency token that HealthOmics uses to catch duplicate requests. It won't start a run if the request ID is a duplicate of a previous run.

If you use infrastructure (such as Lambda functions or step functions) for orchestrating run starts, best practice is to provide a unique request ID for each StartRun request. This ensures that if your infrastructure inadvertently starts a run that it already started, HealthOmics won't start the duplicate run. For example, if the infrastructure is attemping to recover from an upstream error, it may rerun a script that tries to start runs that are duplicate requests.

Choose a workflow version

You can specify a workflow version for the run. If you don't specify a version, HealthOmics starts the run with the default workflow version.

```
aws omics start-run
     --workflow-id workflow id \
      ...
     --workflow-version-name '1.2.1'
```

Override the run storage type

You can override the default run storage type that was set in the workflow.

```
aws omics start-run
```

```
       --workflow-id workflow id \
        ...
       --storage-type STATIC
       --storage-capacity 2400
```

Run a GPU workflow

You can also specify a GPU workflow ID, as shown in the following example:

```
aws omics start-run
       --workflow-id workflow id \
       --role-arn arn:aws:iam::1234567892012:role/service-role/
OmicsWorkflow-20221004T164236 \
       --name GPUTestRunModel \
       --output-uri s3://amzn-s3-demo-bucket1
```

Get information about a 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",
    "outputUri":"s3://bucket/folder/8405154/96c57683-74bf-9d6d-ae7e-f09b097db14a"
    "status": "COMPLETED",
    "workflowId": "4074992",
    "workflowType": "PRIVATE",
    "workflowVersionName": "3.0.0",
```

```
    "roleArn": "arn:aws:iam::123456789012:role/service-role/
OmicsWorkflow-20221004T164236",
    "name": "RunGroupMaxGpuTest",
    "runGroupId": "9938959",
    "digest":
 "sha256:a23a6fc54040d36784206234c02147302ab8658bed89860a86976048f6cad5ac",
    "accelerators": "GPU",
    "outputUri": "s3://amzn-s3-demo-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:3884c62eb0e53fa92459ed9bff133ae6",
      "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.

Rerun a run in HealthOmics

For runs you haven't deleted yet, use the console or API to rerun the run. For runs that youv'e deleted, use the HealthOmics **rerun** tool.

Topics

- Rerun a run using the console
- Rerun a run using the API
- Using the Rerun tool

Rerun a run using the console

From the console, follow these steps to rerun a run:

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, select the run to rerun.
4. From the action menu above the table, choose **Re-run**.

Rerun a run using the API

Use the **StartRun** API operation to rerun an existing run. Provide the following required inputs:

- A service role ARN (`roleArn`).
- The ID of the run to duplicate (`runId`).
- An Amazon S3 location where the run saves the run outputs (`outputUri`).

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

```
     --role-arn arn:aws:iam::1234567892012:role/service-role/
OmicsWorkflow-20221004T164236 \
     --output-uri s3://workflow-output-b6f2fce1
```

Using the Rerun tool

For a deleted run, you can download and use the HealthOmics **rerun** tool to rerun the run. The tool retrieves run information from the CloudWatch Logs manifest. Download the **rerun** tool from the HealthOmics Tool GitHub repository.

The following example shows how to use the **rerun** tool.

```
aws-healthomics-rerun 9876543
```

If the run exists in CloudWatch, you receive a response similar to the following example output. If the workflow no longer exists, you receive an error message.

```
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": {}
}
```

Clone a run in HealthOmics

You can clone an existing run using the HealthOmics console. Cloning creates a new run using the cloned run's configuration values. You can modify these default values and add other optional inputs.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, select the run to clone.
4. From the action menu above the table, choose **Clone run**. The console opens the Clone run form. The form is identical to **Start run**, except the console populates the form with all relevant values from the cloned run.

   The console creates a new run ID for the run clone, and adds this run ID as a suffix to the run name.

   As you proceed through the form pages, you can adjust the configuration values as required.
5. After you review the run configuration, choose **Start run**.

Cancel a run in HealthOmics

You can cancel a run if its status is `PENDING`, `STARTING`, `RUNNING`, or `STOPPING`.

> **Note**
>
> When you cancel a run, HealthOmics doesn't save any of the run outputs.

From the console, follow these steps to cancel a run:

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.

3. On the **Runs** page, choose the run to cancel.
4. The console opens the **Run details** page. From the status banner at the top of the page, choose **Stop run**.
5. Enter **confirm** to stop the run.

To cancel a run using the API, use the **CancelRun** API operation.

The following example shows how to cancel a run using the AWS CLI . 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
```

Delete a run in HealthOmics

When you no longer need a run, you can delete it using the AWS CLI, API, or console. You can delete a run when its status is `COMPLETED` or `CANCELED`.

From the console, follow these steps to delete a run:

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, select one or more runs to delete.
4. From the action menu above the table, choose **Delete**.
5. In the modal form, type **confirm** to confirm the deletion.

The following AWS CLI command deletes a run. 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
```

Using HealthOmics run groups

You can optionally create a run group to cap 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.

- 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 maximum 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 you set a run group maximum to a higher value than your service maximum in a region, HealthOmics applies the service maximum.

Topics

- Run priority
- Create a run group using the console
- Create a run group using the CLI
- Delete a run group using the console
- Delete a run group using the CLI

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 run. For more information, see Start a run in HealthOmics.

Create a run group using the console

To create a run group

1. Open the HealthOmics console.
2. If required, open 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 using the CLI

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-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
```

Delete a run group using the console

You can delete a run group if there are no runs associated with that run group with the status of PENDING, STARTING, RUNNING, or STOPPING.

To delete a run group, follow these steps.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Run groups**.
3. On the **Run groups** page, choose the run group to delete and choose **Delete** in the xx.

Delete a run group using the CLI

You can delete a run group 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
```

Call caching for HealthOmics runs

AWS HealthOmics supports call caching, also known as resume, for private workflows. Call caching saves the outputs of completed workflow tasks after a run finishes. Subsequent runs can use the task outputs from the cache, rather than computing the task outputs again. Call caching reduces compute resource usage, which results in shorter run durations and compute cost savings.

You can access the cached task output files after the run completes. To perform advanced task debugging and troubleshooting, you can cache intermediate task files by specifying these files as task outputs in the workflow definition.

You can use call caching to save the completed task results from failed runs. The next run starts from the last successfully completed task, rather than computing the completed tasks again.

If HealthOmics doesn't find a matching cache entry for a task, the run doesn't fail. HealthOmics recomputes the task and its dependent tasks.

For information about troubleshooting call caching issues, see Troubleshooting call caching issues.

Topics

- How call caching works
- Creating a run cache
- Updating a run cache
- Deleting a run cache
- Contents of a run cache
- Engine-specific caching features
- Using the run cache

How call caching works

To use call caching, you create a run cache and configure it to have an associated Amazon S3 location for the cached data. When you start a run, you specify the run cache. A run cache isn't dedicated to one workflow. Runs from multiple workflows can use the same cache.

During the export phase of a run, the system exports the completed task outputs to the Amazon S3 location. To export intermediate task files, declare these files as task outputs in the workflow

definition. Call caching also internally saves metadata and creates unique hashes for each cache entry.

For each task in a run, the workflow engine detects whether there is a matching cache entry for this task. If there is no matching cache entry, HealthOmics computes the task. If there is a matching cache entry, the engine retrieves the cached results.

To match cache entries, HealthOmics uses the hashing mechanism that's included in the native workflow engines. HealthOmics extends these existing hash implementations to account for HealthOmics variables, such as S3 eTags and ECR container digests.

HealthOmics supports call caching for these workflow language versions:

- WDL versions 1.0, 1.1, and the development version
- Nextflow version 23.10 and 24.10
- All CWL versions

> **Note**
>
> HealthOmics doesn't support call caching for Ready2Run workflows.

Topics

- Shared responsibility model
- Caching requirements for tasks
- Run cache performance
- Cache data retention and invalidation events

Shared responsibility model

There is a shared responsibility between users and AWS to determine whether tasks and runs are good candidates for call caching. Call caching achieves the best outcomes when all tasks are idempotent (repeated executions of a task using the same inputs produce the same results).

However, if a task includes non-deterministic elements (such as random number generations or system time), repeated executions of the task using the same inputs may result in different outputs. This can impact the effectiveness of call caching in the following ways:

- If HealthOmics uses a cache entry (created by a previous run) that is not identical to the output that the task execution would produce for the current run, the run may yield different results than the same run with no caching.
- HealthOmics may not find a matching cache entry for a task that should match, because of non-deterministic task outputs. If it doesn't find the valid cache entry, the run unnecessarily recomputes the task, which reduces the cost saving benefits of using call caching.

The following are known task behaviors that can cause non-deterministic results that affect call caching outcomes:

- Using random number generators.
- Dependence on the system time.
- Using concurrency (race-conditions can cause output variance).
- Fetching local or remote files beyond what is specified in the task input parameters.

For other scenarios that can cause non-deterministic behavior, see Non-deterministic process inputs on the Nextflow documentation site.

If you suspect that a task produces outputs that are non-deterministic, consider using workflow engine features to avoid caching specific tasks that are non-deterministic. For instructions on how to opt out of caching for individual tasks in each supported workflow language, see Engine-specific caching features.

We recommend that you thoroughly review your specific workflow and task requirements before enabling call caching in any environments in which ineffective call caching or different outputs than expected can present risk. For example, the potential limitations of call caching should be carefully considered in determining whether call caching is appropriate for clinical use cases.

Caching requirements for tasks

HealthOmics caches task outputs for tasks that meet the following requirements:

- The task must define a container. HealthOmics won't cache outputs for a task with no container.
- The task must produce one or more outputs. You specify task outputs in the workflow definition.
- The workflow definition must not use dynamic values. For example, if you pass a parameter to a task with a value that increments with every run, HealthOmics doesn't cache the task outputs.

> **Note**
>
> If multiple tasks in a run use the same container image, HealthOmics provides the same image version to all of these tasks. After HealthOmics pulls the image, it ignores any updates to the container image for the duration of the run. This approach provides a predictable and consistent experience and prevents potential issues that could arise from updates to the container image that are deployed mid-run.

Run cache performance

When you turn on call caching for a run, you may notice the following impacts on run performance:

- During the first run, HealthOmics saves the cache data for tasks in the run. You may experience longer export times for this run, because call caching increases the amount of export data.
- In subsequent runs, when resuming a run from cache, it may shorten the number of processing steps and reduce your run time.
- If you also choose to declare intermediate files as outputs, then your export times might be even longer since this data can be more verbose.

Cache data retention and invalidation events

The main purpose of a run cache is to optimize computation of tasks in the run. If there is a valid matching cache entry for a task, HealthOmics uses the cache entry instead of recomputing the task. Otherwise, HealthOmics reverts to the default service behavior, which is to recompute the task and its dependent tasks. By using this approach, cache misses don't cause the run to fail.

We recommend that you manage the run cache size. Over time, cache entries may no longer be valid because of workflow engine or HealthOmics service updates or because of changes you made in the run or the run tasks. The following sections provide additional details.

Topics

- Manifest version updates and data freshness
- Run cache behavior
- Control run cache size

Manifest version updates and data freshness

Periodically, the HealthOmics service may introduce new features or workflow engine updates that invalidate some or all run cache entries. In this situation, your runs can experience a one-time cache miss.

HealthOmics creates a JSON manifest file for each cache entry. For runs started after February 12th 2025, the manifest file includes a version parameter. If a service update invalidates any cache entries, HealthOmics increments the version number so that you can identify the legacy cache entries for removal.

The following example shows a manifest file with the version set to 2:

```
{
     "arn": "arn:aws:omics:us-west-2:12345678901:runCache/0123456/
cacheEntry/1234567-195f-3921-a1fa-ffffcef0a6a4",
     "s3uri": "s3://example/1234567-d0d1-e230-
d599-10f1539f4a32/1348677/4795326/7e8c69b1-145f-3991-a1fa-ffffcef0a6a4",
     "taskArn": "arn:aws:omics:us-west-2:12345678901:task/4567891",
     "workDir": "/mnt/workflow/1234567-d0d1-e230-d599-10f1539f4a32/workdir/call-
TxtFileCopyTask/5w6tn5feyga7noasjuecdeoqpkltrfo3/wxz2fuddlo6hc4uh5s2lreaayczduxdm",
     "files": [
         {
             "name": "output_txt_file",
             "path": "out/output_txt_file/outfile.txt",
             "etag": "ajdhyg9736b9654673b9fbb486753bc8"
         }
     ],
     "nextflowContext": {},
     "otherOutputs": {},
     "version": 2,
  }
```

For runs with cache entries that are no longer valid, rebuild the cache to create new valid entries. Perform the following steps for each run:

1. Start the run once with cache retention set to CACHE ALWAYS. This run creates the new cache entries.
2. For subsequent runs, set the cache retention to its former setting (CACHE ALWAYS or CACHE ON FAILURE).

To clean-up cache entries that are no longer valid, you can delete these cache entries from the cache Amazon S3 bucket. HealthOmics never reuses these cache entries. If you choose to retain entries that aren't valid, there is no impact on your runs.

> **Note**
>
> Call caching saves task output data in the Amazon S3 location specified for the cache, which incurs charges to your AWS account.

Run cache behavior

You can set run cache behavior to save the task outputs for runs that fail (cache on failure) or for all runs (cache always). When you create a run cache, you set the default cache behavior for all runs that use this cache. You can override the default behavior when you start a run.

Cache on failure is useful if you're debugging a workflow that fails after several tasks completed successfully. The subsequent run resumes from the last successfully completed task if all the unique variables considered by the hash are identical to the prior run.

Cache always is useful if you're updating a task in a workflow that completes successfully. We recommend that you follow these steps:

1. Create a new run. Set the **Cache behavior** to **Cache always**, and start the run.
2. After the run completes, update the task in the workflow and start a new run with behavior set **Cache always**. This run processes the updated task and any subsequent tasks that have a dependency on the updated task. All other tasks use the cached results.
3. Repeat step 2 as required, until development is complete for the updated task.
4. Use the updated task as needed on future runs. Remember to switch subsequent runs to **Cache on failure** if you plan to use new or different inputs for these runs.

> **Note**
>
> We recommend **Cache always** mode while using the same test data set, but not for a batch of runs. If you set this mode for a large batch of runs, the system can export large amounts of data to Amazon S3, resulting in increased export times and storage costs.

Control run cache size

HealthOmics doesn't delete or auto-archive any run cache data or apply Amazon S3 clean-up rules for managing the cache data. We recommend that you perform regular cache clean-ups to save on Amazon S3 storage costs and to keep your run cache size manageable. You can delete files directly or set data retention/replication policies on the run cache bucket.

For example, you can configure an Amazon S3 lifecycle policy to expire objects after 90 days, or you can manually clean-up the cache data at the end of each development project.

The following information can help you manage cache data size:

- You can view how much data is in the cache by checking Amazon S3. HealthOmics doesn't monitor or report on cache size.
- If you delete a valid cache entry, the subsequent run doesn't fail. HealthOmics recomputes the task and its dependent tasks.
- If you modify cache names or directory structures such that HealthOmics can’t find a matching entry for a task, HealthOmics recomputes the task.

If you need to check whether a cache entry is still valid, check the cache manifest version number. For more information, see Manifest version updates and data freshness.

Creating a run cache

When you create a run cache, you specify an Amazon S3 location for the cache data. This data must be immediately accessible. Call caching doesn't retrieve objects archived in Glacier (such as GFR and GDA storage classes).

If the Amazon S3 bucket for the cache data is owned by another AWS account, provide that account ID when you create the run cache.

Creating a run cache using the console

From the console, follow these steps to create a run cache.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Run caches**.
3. From the **Run caches** page, choose **Create run cache**.

4. In the **Run cache details** panel of the **Create run cache** page, configure these fields:

   a. Enter a name for the run cache.

   b. (Optional) Enter a description.

   c. Enter an S3 location for the cached output. Choose a bucket in the same Region as your workflow.

   d. (Optional) Enter the AWS account of the bucket owner to verify bucket ownership. If you don't enter a value, the default value is your account ID.

   e. Under **Cache behavior**, configure the default behavior (whether to cache outputs for failed runs or for all runs). When you start a run, you can optionally override the default behavior.

5. (Optional) Associate one or more tags with the run cache.

6. Choose **Create run cache**. The console displays the new run cache in the **Run caches** table.

Creating a run cache using the CLI

Use the **create-run-cache** CLI command to create a run cache. The default cache behavior is `CACHE_ON_FAILURE`.

```
aws omics create-run-cache \
      --name "workflow 123 run cache" \
      --description "my run cache" \
      --cache-s3-location "s3://amzn-s3-demo-bucket" \
      --cache-behavior "CACHE_ALWAYS"                \
      --cache-bucket-owner-id  "111122223333"
```

If the create is successful, you receive a response with the following fields.

```
{
  "arn": "string",
  "id": "string",
  "status": "ACTIVE"
  "tags": {}
  }
```

Updating a run cache

You can change the cache name, description, tags, or cache behavior, but not the S3 location for the cache.

Updating a run cache using the console

From the console, follow these steps to update a run cache.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Run caches**.
3. From the **Run caches** table, choose the run cache to update, then choose **Edit**.
4. In the **Run cache details** panel, you can update the run cache name, description, and cache behavior fields.
5. (Optional) Associate one or more new tags with the run cache, or remove existing tags.
6. Choose **Save run cache**.

Updating a run cache using the CLI

Use the **update-run-cache** CLI command to update a run cache.

```
aws omics update-run-cache \
      --name "workflow 123 run cache" \
      --id "workflow id" \
      --description "my run cache" \
      --cache-behavior "CACHE_ALWAYS"
```

If the update is successful, you receive a response with no data fields.

Deleting a run cache

You can delete a run cache if no active runs are using it. If any runs are using the run cache, wait for the runs to complete or you can cancel the runs.

Deleting a run cache removes the resource and its metadata, but doesn't delete the data in Amazon S3. After you delete the cache, you can't reattach it or use it for subsequent runs.

The cached data remains in Amazon S3 for your inspection. You can remove old cache data using standard S3 **Delete** operations. Alternatively, create an Amazon S3 lifecycle policy to expire cached data that you no longer use.

Deleting a run cache using the console

From the console, follow these steps to delete a run cache.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Run caches**.
3. From the **Run caches** table, choose the run cache to delete.
4. From the **Run caches** table menu, choose **Delete**.
5. From the modal dialog, save the Amazon S3 cache data link for future reference, then confirm that you want to delete the run cache.

   You can use the Amazon S3 link to inspect the cached data, but you can't relink the data to another run cache. Delete the cache data when you've finished the inspection.

Deleting a run cache using the CLI

Use the **delete-run-cache** CLI command to delete a run cache.

```
aws omics delete-run-cache \
      --id "my cache id"
```

If the delete is successful, you receive a response with no data fields.

Contents of a run cache

HealthOmics organizes your run cache with the following structure in your S3 bucket:

```
s3://{cache.S3location}/{cache.uuid}/runID/taskID/{cacheentry.uuid}/
```

The cache.uuid is the globally unique id for the cache. The cacheentry.uuid is the globally unique uuid for a cached task. HealthOmics assigns the uuids to caches and tasks.

For all workflow engines, the cache contains the following files:

- The **{cacheentryuuid}.json** file – HealthOmics creates this manifest file, which contains information about the cache, including a list of all items in the cache, and the cache version.
- Task output files – Each task output consists of one or more files, as defined by the task.

For a workflow that uses Nextflow, the Nextflow engine creates these additional files in the cache:

- The **command.out** file – This file contains the task execution stdout contents.
- The **.exitcode** file – This file contains the task exit code (an integer).

> **Note**
>
> If you want to access intermediate task files in your run cache for advanced troubleshooting, declare these files as task outputs in the workflow definition.

Engine-specific caching features

HealthOmics tries to provide a consistent implementation of call caching across workflow engines. There are some differences based on how each workflow engine handles specific cases:

- Nextflow
  - Caching across different Nextflow versions is not guaranteed. For example, if you run a task in v23.10.0 and subsequently run the same task in v24.10.8, HealthOmics might consider the second run to be a cache miss.
  - You can turn off caching for individual tasks by using the cache **false** directive. For information about this directive, see the Processes in the Nextflow specification.
  - HealthOmics uses Nextflow lenient mode, but doesn't support deep caching mode.
  - Caching evaluates each individual S3 object if you use a glob pattern in the S3 path to the inputs for a task. If you add a new object, HealthOmics recomputes only the tasks that use the new object.
  - HealthOmics doesn't cache task retries. This behavior is consistent with Nextflow's default behavior.
- WDL

- HealthOmics supports the new “directory” type for inputs when you use the development version of the WDL workflow. For call caching, if any object in the directory changes, HealthOmics recomputes all tasks that input the directory.
- HealthOmics supports task-level caching, but not workflow-level caching.
- You can disable caching for individual tasks by using the **volatile** attribute. For more information, see Disable task-level caching with the volatile attribute.

- CWL
  - Constant outputs from tasks aren't explicitly visible from the manifests. HealthOmics caches constant outputs as intermediate files.
  - You can control caching for individual tasks by using the WorkReuse feature.

Using the run cache

By default, runs don't use a run cache. To use a cache for the run, you specify the run cache and the run cache behavior when you start the run.

After a run completes, you can use the console, CloudWatch Logs, or API operations to track cache hits or troubleshoot cache issues. For details, see Tracking call caching information and Troubleshooting call caching issues.

If one or more tasks in a run generate non-deterministic outputs, we strongly recommend that you don’t use call caching for the run, or you opt out these specific tasks from caching. For more information, see Shared responsibility model.

> **Note**
>
> You provide an IAM service role when you start a run. To use call caching, the service role needs permission to access the run cache Amazon S3 location. For more information, see Service roles for AWS HealthOmics.

You can use Amazon Q CLI to analyze and manage your run cache data. For more information, see Example prompts for Amazon Q CLI and the HealthOmics Agentic generative AI tutorial on GitHub.

Topics

- Configuring a run with run cache using the console

- Configuring a run with run cache using the CLI
- Error cases for run caches
- Tracking call caching information

Configuring a run with run cache using the console

From the console, you configure the run cache for a run when you start the run.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, choose the run to start.
4. Choose **Start run** and complete steps 1 and 2 of **Start run** as described in Starting a run using the console.
5. In step 3 of **Start run**, choose **Select an existing run cache**.
6. Select the cache from the **Run cache ID** drop-down list.
7. To override the default run cache behavior, choose the **Cache behavior** for the run. For more information, see Run cache behavior.
8. Continue to step 4 of **Start run**.

Configuring a run with run cache using the CLI

To start a run that uses a run cache, add the cache-id parameter to the **start-run** CLI command. Optionally, use the `cache-behavior` parameter to override the default behavior that you configured for the run cache. The following example shows only the cache fields for the command:

```
aws omics start-run \
        ...
      --cache-id "xxxxxx"    \
      --cache-behavior  CACHE_ALWAYS
```

If the operation is successful, you receive a response with no data fields.

Error cases for run caches

For the following scenarios, HealthOmics may not cache task outputs, even for a run with cache behavior set to **Cache always**.

- If the run encounters an error before the first task completes successfully, there are no cache outputs to export.
- If the export process fails, HealthOmics doesn't save the task outputs to the Amazon S3 cache location.
- If the run fails due to a **filesystem out of space** error, call caching doesn't save any task outputs.
- If you cancel a run, call caching doesn't save any task outputs.
- If the run experiences a run timeout, call caching doesn't save any task outputs, even if you configured the run to use cache on failure.

Tracking call caching information

You can track call caching events (such as run cache hits) using the console, the CLI, or CloudWatch Logs.

Topics

- Track cache hits using the console
- Track call caching using the CLI
- Track call caching using CloudWatch Logs

Track cache hits using the console

In the run details page for a run, the **Run tasks** table displays **Cache hit** information for each task. The table also includes a link to the associated cache entry. Use the following procedure to view cache hit information for a run.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. On the **Runs** page, choose the run to inspect.
4. On the run details page, choose the **Run tasks** tab to display the tasks table.
5. If a task has a cache hit, the **Cache hit** column contains a link to the run cache entry location in Amazon S3.
6. Choose the link to inspect the run cache entry.

Track call caching using the CLI

Use the **get-run** CLI command confirm whether the run used a call cache.

```
aws omics get-run --id 1234567
```

In the response, if the `cacheId` field is set, the run uses that cache.

Use the **list-run-tasks** CLI command to retrieve the cache data location for each cached task in the run.

```
aws omics list-run-tasks --id 1234567
```

In the response, if the cacheHit field for a task is true, the cacheS3Uri field provides the cache data location for that task.

You can also use the **get-run-task** CLI command to retrieve the cache data location for a specific task:

```
aws omics get-run-task --id 1234567 --task-id <task_id>
```

Track call caching using CloudWatch Logs

HealthOmics creates cache activity logs in the `/aws/omics/WorkflowLog` CloudWatch log group. There is a log stream for each run cache: **runCache/<cache_id>/<cache_uuid>**.

For runs that use call caching, HealthOmics generates CloudWatch Logs entries for these events:

- creating a cache entry (CACHE_ENTRY_CREATED)
- matching a cache entry (CACHE_HIT)
- failing to match a cache entry (CACHE_MISS)

For more information about these logs, see Logs in CloudWatch.

Use the following CloudWatch Insights query on the `/aws/omics/WorkflowLog` log group to return the number of cache hits per run for this cache:

```
filter @logStream like 'runCache/<CACHE_ID>/'
 fields @timestamp, @message
 filter logMessage like 'CACHE_HIT'
```

```
 parse "run: *," as run
 stats count(*) as cacheHits by run
```

Use the following query to return the number of cache entries created by each run:

```
filter @logStream like 'runCache/<CACHE_ID>/'
 fields @timestamp, @message
 filter logMessage like 'CACHE_ENTRY_CREATED'
 parse "run: *," as run
 stats count(*) as cacheEntries by run
```

Sharing HealthOmics 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**
>
> HealthOmics automatically allows a shared workflow to access the Amazon ECR repository while the workflow is running in the subscriber's account. You don't need to grant additional repository access for shared workflows.

When you share a workflow, the subscriber can use any of the workflow versions. If you need version-level access control for a shared workflow, we recommend that you create separate workflows rather than using workflow versions.

Topics

- Subscribing to a shared workflow
- Monitoring status of a workflow share
- Sharing a private workflow using the console
- Sharing a private workflow using the CLI
- Accepting a shared workflow using the console
- Running a shared workflow using the console

- Running a shared workflow using the API

Subscribing to a shared workflow

To subscribe to a shared workflow, you follow these overall 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.

Monitoring status of a workflow share

HealthOmics sends an event to EventBridge for each status change of a workflow share. If you want to receive notifications about specific status changes, set up an EventBridge rule to monitor **Workflow share Status Change** events. For example:

- You want a notification each time you receive a workflow share request, and each time a user revokes a workflow share.
- After you initiate a workflow share request, you want to receive a notification when the user accepts or declines the request.

For details about using events, see Using EventBridge with AWS HealthOmics.

Sharing a private workflow using the 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.
2. If required, open 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.

Sharing a private workflow using the CLI

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.

Accepting a shared workflow using the 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.
2. If required, open 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.

Running a shared workflow using the console

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

1. Open the HealthOmics console.
2. If required, open 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 a run using the console.

Running a shared workflow using the API

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

```
aws omics get-workflow --id 1234567 \
--workflow-owner-id 55555555555
```

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 55555555555 \
--role-arn arn:aws:iam::1234567892012:role/service-role/OmicsWorkflow-20221004T164236 \
--name ArchiveTest --retention-mode REMOVE
```

Ready2Run workflows in HealthOmics

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.

HealthOmics supports versioning for Ready2Run workflows. For a Ready2Run workflow that offers versions, you can specify the version name when you start a run.

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

> **Note**
>
> Sentieon Ready2Run workflows are subscription-based. When you run a Sentieon Ready2Run workflow for the first time in an account, Sentieon automatically creates a two-week evaluation license for your AWS account. The license is valid for all Sentieon Ready2Run workflows. After the evaluation period ends, you can request a permanent license or request an extension to the evaluation license. See **Subscribing to Sentieon Ready2Run workflows** for details.

Topics

- Available Ready2Run workflows in HealthOmics
- Subscribing to Sentieon Ready2Run workflows
- Starting HealthOmics Ready2Run workflows using the console
- Starting HealthOmics Ready2Run workflows using the API

Available Ready2Run workflows in HealthOmics

The following table lists the Ready2Run workflows that are available in HealthOmics.

You can log in to the HealthOmics console to view detailed information about these workflows, including input parameters and workflow diagrams. For pricing information about Ready2Run workflows, see HealthOmics Pricing.

> **Note**
>
> Each Ready2Run workflow has a maximum input file size. These maximum file sizes aren't adjustable.

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
AlphaFold for 601-1200 residues	Google DeepMind	No	1	11:15
AlphaFold for up to 600 residues	Google DeepMind	No	1	7:30
Bases2Fastq for 2x150	Element Biosciences	No	1000	1:45
Bases2Fastq for 2x300	Element Biosciences	No	1000	1:30
Bases2Fastq for 2x75	Element Biosciences	No	500	0:45
ESMFold for up to 800 residues	Meta Research	No	1	0:15
GATK-BP fq2bam	Broad Institute	No	64	10:10

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
GATK-BP Germline bam2vcf for 30x genome	Broad Institute	No	39	2:45
GATK-BP Germline fq2vcf for 30x genome	Broad Institute	No	64	12:30
GATK-BP Somatic WES bam2vcf	Broad Institute	No	86	1:30
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 30X	NVIDIA Corporation	No	80	1:39
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 50X	NVIDIA Corporation	No	120	2:45
NVIDIA Parabricks BAM2FQ2BAM WGS for up to 5X	NVIDIA Corporation	No	20	0:18
NVIDIA Parabricks FQ2BAM WGS for up to 30X	NVIDIA Corporation	No	71	1:00

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
NVIDIA Parabricks FQ2BAM WGS for up to 50X	NVIDIA Corporation	No	137	1:45
NVIDIA Parabricks FQ2BAM WGS for up to 5X	NVIDIA Corporation	No	13	0:15
NVIDIA Parabrick s Germline DeepVariant WGS for up to 30X	NVIDIA Corporation	No	71	2:00
NVIDIA Parabrick s Germline DeepVariant WGS for up to 50X	NVIDIA Corporation	No	137	3:30
NVIDIA Parabrick s Germline DeepVariant WGS for up to 5X	NVIDIA Corporation	No	12	0:30

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 30X	NVIDIA Corporation	No	71	1:15
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 50X	NVIDIA Corporation	No	137	2:00
NVIDIA Parabricks Germline HaplotypeCaller WGS for up to 5X	NVIDIA Corporation	No	13	0:15
NVIDIA Parabricks Somatic Mutect2 WGS for up to 50X	NVIDIA Corporation	No	196	0:45
scRNAseq with KallistoBUStools	NF-Core	No	119	1:30
scRNAseq with Salmon Alevin-fry	NF-Core	No	119	2:30

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
scRNAseq with STARsolo	NF-Core	No	119	2:30
Sentieon Germline BAM WES for up to 300x	Sentieon, Inc.	Yes	9	1:00
Sentieon Germline BAM WGS for up to 32x	Sentieon, Inc.	Yes	18	1:30
Sentieon Germline FASTQ WES for up to 100x	Sentieon, Inc.	Yes	5	0:45
Sentieon Germline FASTQ WES for up to 300x	Sentieon, Inc.	Yes	26	2:00
Sentieon Germline FASTQ WGS for up to 32x	Sentieon, Inc.	Yes	51	3:30
Sentieon LongRead for ONT	Sentieon, Inc.	Yes	25	1:30
Sentieon LongRead for PacBio HiFi	Sentieon, Inc.	Yes	58	4:00

Workflow name	Publisher	Subscription required?	Maximum input file size (GiB)	Estimated run time (HH:MM)
Sentieon Somatic WES	Sentieon, Inc.	Yes	50	2:30
Sentieon Somatic WGS	Sentieon, Inc.	Yes	113	4:30
Ultima Genomics DeepVariant for up to 40x	Ultima Genomics	No	91	1:55

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:

- Increasing the maximum input file size
- Changing the compute resources or run storage
- Changing the workflow definition or containers
- Adding runs to a run group
- Sharing the workflow

If the publisher has shared the Ready2Run workflow on GitHub, you can make your own private workflow based on the Ready2Run workflow. The following table provides links to GitHub workflows for each publisher.

Publisher	Workflows on GitHub
Google DeepMind, Meta Research	Protein folding workflows
Element Biosciences	For information, contact Element Biosciences
Broad Institute	GATK workflows
NVIDIA Corporation	Parabricks workflows

Publisher	Workflows on GitHub
nf-core	NF-Core workflows
Sentieon	Sentieon workflows
Ultima Genomics	Ultima Genomics workflows

Subscribing to Sentieon Ready2Run workflows

Sentieon Ready2Run workflows are subscription-based. When you run a Sentieon Ready2Run workflow for the first time in an account, Sentieon automatically creates a two-week evaluation license for your AWS account. The license is valid for all Sentieon Ready2Run workflows. After the evaluation period ends, you can request a permanent license or request an extension to the evaluation license.

Follow these steps to subscribe to the Sentieon Ready2Run workflows:

- Find your AWS Canonical User ID by following these instructions.
- Send an email to the Sentieon support group (support@sentieon.com) to request a software license. Provide your AWS Canonical User ID in the email.

Starting HealthOmics Ready2Run workflows using the 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.
2. If required, open 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. Optionally, you can specify the workflow version. You can also add run priority to the run.
7. Enter or select an Amazon S3 location for the run output.
8. For **Run metadata retention mode**, choose whether to retain or remove run metadata.
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**.

Starting HealthOmics Ready2Run workflows using the API

Most of the API operations behave in a similar fashion for Ready2Run workflows 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 specify a workflow version, use the **workflow-version** parameter, as shown in this example.

```
aws-omics start-run \
  --workflow-type READY2RUN \
  ...
  --version-name '3.0.0'
```

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

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

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.

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.

Topics

- HealthOmics ETags and data provenance
- Creating a HealthOmics reference store
- Creating a HealthOmics sequence store
- Deleting HealthOmics reference and sequence stores
- Importing read sets into a HealthOmics sequence store
- Direct upload to a HealthOmics sequence store
- Exporting HealthOmics read sets to an Amazon S3 bucket
- Accessing HealthOmics read sets with Amazon S3 URIs
- Activating read sets in HealthOmics

HealthOmics ETags and data provenance

A HealthOmics ETag (entity tag) 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.

When the file type schema of the read set permits it, the sequence store updates fields 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.

Amazon S3 ETags

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 Object API 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 HealthOmics calculates ETags

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 a HealthOmics reference store

A reference store in HealthOmics is a data store for the storage of reference genomes. You can have a single reference store in each AWS account and Region. You can create a reference store using the console or CLI.

Topics

- Creating a reference store using the console
- Creating a reference store using the CLI

Creating a reference store using the console

To create a reference store

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Reference store**.
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.

Creating a reference store using the CLI

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetBucketLocation"

            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1",
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        }
    ]
}
```

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

JSON

```
{
    "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://amzn-s3-
demo-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://amzn-s3-demo-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 a HealthOmics sequence store

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. You can add tags to read sets and use IAM policies to control access to read sets. 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 and the location where S3 access logs are stored. 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 specify up to five read set tag keys. When you create or update a read set with a tag key that matches one of these keys, the read set tags are propagated to the corresponding Amazon S3 object. System tags created by HealthOmics are propagated by default.

Topics

- Creating a sequence store using the console
- Creating a sequence store using the CLI
- Updating a sequence store
- Updating read set tags for a sequence store
- Importing genomic files

Creating a sequence store using the console

To create a sequence store

1. Open the HealthOmics console.
2. If required, open 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.

4. For **Fallback location in S3**, specify an Amazon S3 location. HealthOmics uses the fallback location for storing any files that fail to create a read set during a direct upload. You need to grant the HealthOmics service write access to the Amazon S3 fallback location. For an example policy, see Configure a fallback location.

   Fallback locations aren't available for sequence stores created before May 16, 2023.

5. (Optional) For **Read set tag keys for S3 propagation**, you can enter up to five read set keys to propagate from a read set to the underlying S3 Objects. By propagating tags from a read set to the S3 object, you can grant S3 access permissions based on tags and/or end users to see the propagated tags through the Amazon S3 getObjectTagging API operation.

   a. Enter one key value in the text box. The console creates a new text box to add the next key.

   b. (Optional) Choose **Remove** to remove all the keys.

6. Under **Data Encryption**, select whether you want data encryption to be owned and managed by AWS or to use a customer managed CMK.

7. (Optional) Under **S3 Data access**, select whether to create a new role and policy to access the sequence store through Amazon S3.

8. (Optional) For **S3 access logging**, select `Enabled` if you want Amazon S3 to collect access log records.

   For **Access logging location in S3**, specify an Amazon S3 location to store the logs. This field is visible only if you enabled S3 access logging.

9. **Tags** (optional) - Provide up to 50 tags for this sequence store. These tags are separate from read set tags that are set during read set import/tag update

After you create the store, it's ready for Importing genomic files.

Creating a sequence store using the CLI

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://
amzn-s3-demo-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://amzn-s3-demo-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",
            "updatedTime": "2024-09-13T04:11:31.242Z",
            "fallbackLocation" : "s3://amzn-s3-demo-bucket",
            "status": "Active"
        }
    ]
}
```

You can use **get-sequence-store** to learn more about a sequence store by using its ID, as shown in the following example:

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

You receive the following response:

```
{
  "arn": "arn:aws:omics:us-west-2:123456789012:sequenceStore/sequencestoreID",
```

```
  "creationTime": "2024-01-12T04:45:29.857Z",
  "updatedTime": "2024-09-13T04:11:31.242Z",
  "description": null,
  "fallbackLocation": null,
  "id": "2015356892",
  "name": "MySequenceStore",
  "s3Access": {
      "s3AccessPointArn": "arn:aws:s3:us-
west-2:123456789012:accesspoint/592761533288-2015356892",
      "s3Uri": "s3://592761533288-2015356892-ajdpi90jdas90a79fh9a8ja98jdfa9jf98-
s3alias/592761533288/sequenceStore/2015356892/",
      "accessLogLocation": "s3://IAD-seq-store-log/2015356892/"
  },
  "sseConfig": {
      "keyArn": "arn:aws:kms:us-west-2:123456789012:key/eb2b30f5-635d-4b6d-b0f9-
d3889fe0e648",
      "type": "KMS"
  },
  "status": "Active",
  "statusMessage": null,
  "setTagsToSync": ["withdrawn","protocol"],
}
```


After creation, several store parameters can also be updated. This can be done through the Console or the API `updateSequenceStore` operation.

Updating a sequence store

To update a sequence store, follow these steps:

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Sequence stores**.
3. Choose the sequence store to update.
4. In the **Details** panel, choose **Edit**.
5. On the **Edit details** page, you can update the following fields:

   - **Sequence store name** - A unique name for this store.
   - **Description** - A description of this sequence store.
   - **Fallback location in S3**, specify an Amazon S3 location. HealthOmics uses the fallback location for storing any files that fail to create a read set during a direct upload.

- **Read set tag keys for S3 propagation** you can enter up to five read set keys to propagate to Amazon S3.
- (Optional) For **S3 access logging**, select `Enabled` if you want Amazon S3 to collect access log records.

  For **Access logging location in S3**, specify an Amazon S3 location to store the logs. This field is visible only if you enabled S3 access logging.
- **Tags** (optional) - Provide up to 50 tags for this sequence store.

Updating read set tags for a sequence store

To update read set tags or other fields for a sequence store, follow these steps:

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Sequence stores**.
3. Choose the sequence store that you want to update.
4. Choose the **Details** tab.
5. Choose **Edit**.
6. Add new read set tags or delete existing tags, as required.
7. Update the name, description, fallback location, or S3 data access, as required.
8. Choose **Save changes**.

Importing genomic files

To import genomic files to a sequence store, follow these steps:

To import a genomics file

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose 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 Importing read sets into a HealthOmics sequence store.
7. Click **Create import job**.

Deleting HealthOmics 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
```

Importing read sets into a HealthOmics sequence store

After you create your sequence store, create import jobs to upload read sets 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 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.

You can reuse the IAM access policy that you used to create the Reference store.

The following topics describe the major steps you follow to import a read set into you sequence store and then get information about the imported data.

Topics

- Upload files to Amazon S3
- Creating a manifest file
- Starting the import job
- Monitor the import job
- Find the imported sequence files
- Get details about a read set
- Download the read set data files

Upload files to Amazon S3

The following example shows how to move files into your Amazon S3 bucket.

```
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.

Creating a manifest file

You must also create a manifest file in JSON to model the import job in `import.json` (see the following example). If you create a sequence store in the console, you don't have to specify the `sequenceStoreId` or `roleARN`, so your manifest file starts with the `sources` input.

API manifest

The following example imports 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://amzn-s3-demo-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://amzn-s3-demo-bucket/SRR233106_1.filt.fastq.gz",
              "source2": "s3://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-bucket/SRR233106_1.filt.fastq.gz",
          "source2": "s3://amzn-s3-demo-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://amzn-s3-demo-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.

Starting the import job

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"
}
```

Monitor the import job

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://amzn-s3-demo-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",
          "readSetID": "1234567890"
      },
      {
          "sourceFiles":
          {
              "source1": "s3://amzn-s3-demo-bucket/SRR233106_1.filt.fastq.gz",
              "source2": "s3://amzn-s3-demo-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",
          "readSetID": "1234567890"
      },
      {
          "sourceFiles":
          {
              "source1": "s3://amzn-s3-demo-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",
          "readSetID": "1234567890"
      },
      {
          "sourceFiles":
          {
              "source1": "s3://amzn-s3-demo-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",
          "readSetID": "1234567890"
      }
  ]
}
```

Find the imported sequence files

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"
            }
        }
    ]
}
```

Get details about a read set

To view more details about a read set, use the **GetReadSetMetadata** 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",
"creationJobId": "33222111",
"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"
}
```


Download the read set data files

You can access the objects for an active read set using the Amazon S3 **GetObject** API operation. The URI for the object is returned in the **GetReadSetMetadata** API response. For more information, see Accessing HealthOmics read sets with Amazon S3 URIs.

Alternatively, use the HealthOmics **GetReadSet** API operation. You can use **GetReadSet** 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 HealthOmics sequence store

We recommend that you use the HealthOmics Transfer Manager to add files to your sequence store. For more information about using Transfer Manager, see this GitHub Repository. 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 files that fail to upload. Fallback locations are available for sequence stores created after May 15, 2023.

Topics

- Direct upload to a sequence store using the AWS CLI
- Configure a fallback location

Direct upload to a sequence store using the AWS CLI

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

To direct upload using AWS CLI commands

1. Create the parts by separating your data, as shown in the following example.

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

2. After your source files are in parts, create a multipart read set upload, as shown 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"
```

You get the `uploadID` and other metadata in the response. 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"
}
```

3. 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"
}
```

4. 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+vB9C3Ge8ADHkbKq752n3BCUzyl41qEkqlOD5M=",
        "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": "keZzVzJNChAqgOdZMvOmjBwrOPM0enPj1UAfs0nvRto=",
        "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"
    }
]
}
```

5. 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"
    }
]
}
```

6. After you upload all parts of your file, use **complete-multipart-read-set-upload** to conclude the upload process, as shown in the following example. 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"
}
```

7. 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
```

8. After the upload is complete, 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 \
 --id read set ID \
 --file SOURCE1 \
 --part-number 1 myfile.fastq.gz
```

9. 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"
}
```

Configure a fallback location

When you create or update a sequence store, you can configure an Amazon S3 bucket as the fallback location for files that fail to upload. The file parts for those read sets are transferred to the fallback location. Fallback locations are available for sequence stores created after May 15, 2023.

Create an Amazon S3 bucket policy to grant HealthOmics write access to the Amazon S3 fallback location, as shown in the following example:

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "omics.amazonaws.com"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
}
```

If the Amazon S3 bucket for fallback or access logs uses a customer managed key, add the following permissions to the key policy:

```
 {
```

```
    "Sid": "Allow use of key",
    "Effect": "Allow",
    "Principal": {
        "Service": "omics.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*"
}
```

Exporting HealthOmics read sets to an Amazon S3 bucket

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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1",
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ]
    }
  ]
}
```

JSON

```
{
```

```
"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>,
      "status": "COMPLETED",
      "creationTime": "2022-10-22T01:33:38.079000+00:00",
      "completionTime": "2022-10-22T01:34:28.941000+00:00"
  }
]
}
```

In addition to exporting your read sets, you can also share them by using the Amazon S3 access URIs. To learn more, see Accessing HealthOmics read sets with Amazon S3 URIs.

Accessing HealthOmics read sets with Amazon S3 URIs

You can use Amazon S3 URI paths to access your active sequence store read sets.

With the Amazon S3 URI path, you can use Amazon S3 operations to list, share, and download your read sets. Access through the S3 APIs accelerates collaboration and tool integration given many industry tools are built already to read from S3. In addition, you can share access to the S3 APIs with other accounts and provide cross-region read access to data.

HealthOmics doesn't support Amazon S3 URI access to archived read sets. When you activate a read set, 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, seethe 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:List Bucket 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 HealthOmics ETags and data provenance.

Topics

- Amazon S3 URI structure in HealthOmics storage
- Using Hosted or Local IGV to access read sets
- Using Samtools or HTSlib in HealthOmics
- Using Mountpoint HealthOmics
- Using CloudFront with HealthOmics

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 enableskms: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 in HealthOmics

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-ajdpi90jdas90a79fh9a8ja98jdfa9jf98-
s3alias/592761533288/sequenceStore/2015356892/readSet/9515444019/
import_source1.fastq.gz"
},
         },
        "index": {
            "totalParts": 1,
            "partSize": 53216,
            "contentLength": 10485760
            "s3Access": {
        "s3Uri": "s3://accountID-sequence store ID-ajdpi90jdas90a79fh9a8ja98jdfa9jf98-
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

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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 peform 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 HealthOmics variant stores
- Creating HealthOmics variant store import jobs
- Creating HealthOmics annotation stores
- Creating import jobs for HealthOmics annotation stores
- Creating HealthOmics annotation store versions
- Deleting HealthOmics analytics stores
- Querying HealthOmics analytics data
- Sharing HealthOmics analytics stores

Creating HealthOmics variant stores

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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

Topics

- Creating a variant store using the console
- Creating a variant store using the API

Creating a variant store using the console

You can create a variant store using the HealthOmics console.

1. Open the HealthOmics console.
2. If required, open 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 a variant store using the API

Use HealthOmics `CreateVariantStore` API operation to create variant stores. You can also perform this operation with the AWS CLI.

To create a variant store, you provide a name for the store and the ARN of a reference store. The variant store is ready to ingest data when its status changes to READY.

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 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 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 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-2:55555555555:referenceStore/5506874698"
            },
            "status": "ACTIVE",
            "storeArn": "arn:aws:omics:us-west-2:55555555555: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 doesn't 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 HealthOmics variant store import jobs

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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::55555555555: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://amzn-s3-demo-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>,
      hgvsc: 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 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://amzn-s3-demo-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 contains 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::55555555555: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 HealthOmics annotation stores

Important

AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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

- Creating an annotation store using the console
- Creating an annotation store using the API

Creating an annotation store using the console

Use the following procedure to create annotation stores with the HealthOmics console.

To create an annotation store

1. Open the HealthOmics console.
2. If required, open 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**.

Creating 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 GFF \
          --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 import jobs for HealthOmics annotation stores

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

Topics

- Creating an annotation import job using the API
- Additional parameters for TSV and VCF formats
- Creating TSV formatted annotation stores
- Starting VCF formatted import jobs

Creating 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.vep.vcf"
              }
          ],
          "roleArn": "arn:aws:iam::55555555555: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::55555555555: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.
`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://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-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.

Creating HealthOmics annotation store versions

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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",
     "creation Time": "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
```

Deleting HealthOmics analytics stores

Important

AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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 HealthOmics analytics data

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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

For variant stores created after Sept 26, 2024, HealthOmics partitions the store by sample ID. This partitioning means that HealthOmics uses the sample ID to optimize storing of the variant information. Queries that use sample information as filters will return results faster, as the query scans less data.

HealthOmics uses sample IDs as partition file names. Before you ingest data, check whether the sample ID contains any PHI data. If it does, change the sample ID before you ingest the data. For more information about what content to include and not include in sample IDs, see guidance on the AWS HIPAA compliance web page.

Topics

- Configuring Lake Formation to use HealthOmics
- Configuring Athena for queries
- Running queries on HealthOmics variant stores

Configuring Lake Formation to use HealthOmics

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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

Topics

- Creating or verify Lake Formation administrators
- Creating resource links using the Lake Formation console
- Configuring permissions for AWS RAM resource shares

Creating 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 **Administative roles and tasks**.

4. Add any additional administrators as required.

Creating 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, select the desired database.
4. From the **Create** menu, choose **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.

Configuring 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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "omics:*",
```

```
        "ram:AcceptResourceShareInvitation",
        "ram:GetResourceShareInvitations"
      ],
      "Resource": "*"
    }
  ]
}
```

Configuring Athena for queries

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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 HealthOmics variant stores

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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 HealthOmics analytics stores

> **Important**
>
> AWS HealthOmics variant stores and annotation stores are no longer open to new customers. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availability change.

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.

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

- Creating a share
- Retrieve information about a share
- View the shares that you own
- View accepted shares from other accounts
- Delete a share

Creating 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

Topics

- Important notice
- Tagging HealthOmics resources
- Sequence store read set tags
- Adding a tag to a HealthOmics resource
- Listing tags for a resource
- Removing tags from a data store

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-inputed 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.

JSON

```
{
    "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.

Sequence store read set tags

For sequence stores, tags created on the read set sit at the read set resource level. Read sets also contain objects under them that can be accessed, searched, and restricted using S3 APIs. By default, the sample ID (omics:sampleId) and subject ID (omics:subjectId) are added to the object.

Additionally, up to five tags can be synchronized between the read set and the objects under it. The configuration for which tags to sync is a store level configuration set during store creation or update using the `propogatedSetLevelTags` parameter.

If there is data already in the store, updating the keys may take time. During this update, HealthOmics changes the store status to **Updating**. On completion, HealthOmics sets the store status to **Active**. While the tags are propagating, permissions relying on the tags may not be enforced. Permissions will be enforced after the tag propagation is completed.

When tags are set or updated on the read set, the system decides whether to update the objects for that read set, based on the store configuration.

Adding a tag to a 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.

IAM permissions for HealthOmics

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
- Amazon ECR permissions
- 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.

To grant permission for users to perform actions on a workflow version, you must add the workflow and the specific workflow version to the resource list.

The following IAM policy allows a user to access all HealthOmics API actions, and to pass service roles to HealthOmics.

Example User policy

JSON

```
{
  "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 don'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.

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.

JSON

```
{
  "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"
          ]
      },
      {
          "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/*"
            ]
        }
    ]
}
```

Service roles for AWS HealthOmics

A service role is an AWS Identity and Access Management (IAM) role that grants permissions for an AWS service to access resources in your account. You provide a service role to AWS HealthOmics when you start an import job or start a run.

The HealthOmics console can create the required role for you. If you use the HealthOmics API to manage resources, create the service role using the IAM console. For more information, see Create a role to delegate permissions to an AWS service.

Service roles must have the following trust policy.

JSON

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

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

The trust policy allows the HealthOmics service to assume the role.

Topics

- Example IAM service policies
- Example CloudFormation template

Example IAM service policies

In these examples, resource names and account IDs are placeholders for you to replace with actual values.

The following example shows the policy for a service role that you can use for starting a run. The policy grants permissions to access the Amazon S3 output location, the workflow log group, and the Amazon ECR container for the run.

> **Note**
>
> If you're using call caching for the run, add the run cache Amazon S3 location as a resource in the s3 permissions.

Example Service role policy for starting a run

JSON

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

```
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/omics/
WorkflowLog:log-stream:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/omics/
WorkflowLog:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
```

```
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": [
                "arn:aws:ecr:us-east-1:123456789012:repository/*"
            ]
        }
    ]
}
```

The following example shows the policy for a service role that you can use for a store import job. The policy grants permissions to access the Amazon S3 input location .

Example Service role for Reference store job

JSON

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

        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }

    ]
}
```

Example CloudFormation template

The following sample CloudFormation 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"
                ]
              }
            }
          ]
        }
```

Amazon ECR permissions

Before the HealthOmics service can run a workflow in a container from your private Amazon ECR repository, you create a resource policy for the repository. The policy grants permission for the HealthOmics service to use the container. You add this resource policy to each private repository referenced by the workflow.

> **Note**
>
> The private repository and the workflow must be in the same region.

If different AWS accounts own the workflow and the repository, you need to configure cross-account permissions.

You don't need to grant additional repository access for shared workflows. However, you can create policies that allow or deny specific workflows access to the container image.

To use the Amazon ECR pull through cache feature, you need to create a registry permission policy.

The following sections describe how to configure Amazon ECR resource permissions for these scenarios. For more information about permissions in Amazon ECR, see Private registry permissions in Amazon ECR.

Topics

- Create a resource policy for the Amazon ECR repository
- Running workflows with cross-account containers
- Amazon ECR policies for shared workflows
- Policies for Amazon ECR pull through cache

Create a resource policy for the Amazon ECR repository

Create a resource policy to allow the HealthOmics service to run a workflow using a container in the repository. The policy grants permission for the HealthOmics service principal to access the required Amazon ECR actions.

Follow these steps to create the policy:

1. Open the private repositories page in the Amazon ECR console and select the repository you're granting access to.
2. From the side bar navigation, select **Permissions**.
3. Choose **Edit**.
4. Choose **Edit policy JSON**.
5. Add the following policy statement and then select **Save**.

   JSON

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

```
    "Statement": [
        {
            "Sid": "omics workflow access",
            "Effect": "Allow",
            "Principal": {
                "Service": "omics.amazonaws.com"
            },
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": "*"
        }
    ]
}
```

Running workflows with cross-account containers

If different AWS accounts own the workflow and the container, you need to configure the following cross-account permissions:

1. Update the Amazon ECR policy for the repository to explicitly grant permission to the account that owns the workflow.
2. Update the service role for the account that owns the workflow to grant it access to the container image.

The following example demonstrates an Amazon ECR resource policy that grants access to the account that owns the workflow.

In this example:

- Workflow account ID: 111122223333
- Container repository account ID: 444455556666
- Container name: samtools

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "omics.amazonaws.com"
            },
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowAccessToTheServiceRoleOfTheAccountThatOwnsTheWorkflow",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/DemoCustomer"
            },
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        }
    ]
}
```

To complete the setup, add the following policy statement to the service role of the account that owns the workflow. The policy grants permission for the service role to access the “samtools” container image. Make sure to replace the account numbers, container name, and region with your own values.

```
{
    "Sid": "CrossAccountEcrRepoPolicy",
    "Effect": "Allow",
```

```
    "Action": ["ecr:BatchCheckLayerAvailability", "ecr:BatchGetImage",
 "ecr:GetDownloadUrlForLayer"],
    "Resource": "arn:aws:ecr:us-west-2:444455556666:repository/samtools"
}
```

Amazon ECR policies for shared workflows

> **Note**
>
> HealthOmics automatically allows a shared workflow to access the Amazon ECR repository in the workflow owner's account, while the workflow is running in the subscriber's account. You don't need to grant additional repository access for shared workflows. For more information see Sharing HealthOmics workflows.

By default, subscriber don't have access to the Amazon ECR repository to use the underlying containers. Optionally, you can customize access to the Amazon ECR repository by adding condition keys to the repository's resource policy. The following sections provide example policies.

Restrict access to specific workflows

You can list individual workflows in a condition statement, so only these workflow can use containers in the repository. The **SourceArn** condition key specifies the ARN of the shared workflow. The following example grants permission for the specified workflow to use this repository.

JSON

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

```
          "ecr:BatchCheckLayerAvailability"
        ],
        "Resource": "*",
        "Condition": {
           "StringEquals": {
              "aws:SourceArn": "arn:aws:omics:us-
east-1:111122223333:workflow/1234567"
           }
        }
     }
  ]
}
```

Restrict access to specific accounts

You can list subscriber accounts in a condition statement, so that only these accounts have permission to use containers in the repository. The **SourceAccount** condition key specifies the AWS account of the subscriber. The following example grants permission for the specified account to use this repository.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "OmicsAccessPrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "omics.amazonaws.com"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
```

```
      }
    }
  ]
}
```

You can also deny Amazon ECR permissions to specific subscribers, as shown in the following example policy.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
          {
              "Sid": "OmicsAccessPrincipal",
              "Effect": "Allow",
              "Principal": {
                  "Service": "omics.amazonaws.com"
              },
              "Action": [
                  "ecr:GetDownloadUrlForLayer",
                  "ecr:BatchGetImage",
                  "ecr:BatchCheckLayerAvailability"
              ],
              "Resource": "*",
              "Condition": {
                "StringNotEquals": {
                  "aws:SourceAccount": "111122223333"
                }
            }
         }
      ]
  }
```

Policies for Amazon ECR pull through cache

To use Amazon ECR pull through cache, you create a registry permission policy. You also create a repository creation template, which defines the permissions for the repositories created by Amazon ECR pull through cache.

The following sections include examples of these policies. For more information about pull through cache, see Sync an upstream registry with an Amazon ECR private registry in the *Amazon Elastic Container Registry User Guide*.

Registry permission policy

To use Amazon ECR pull through cache, create a registry permission policy. The registry permissions policy provides control over replication and pull through cache permissions.

For cross-account replication, you must explictly allow each AWS account that can replicate its repositories to your registry.

By default, when you create a pull through cache rule, any IAM principal that has permission to pull images from a private registry can also use the pull through cache rule. You can use registry permissions to further scope down these permissions to specific repositories.

Add a registry permission policy to the account that owns the container image.

In the following example, the policy allows the HealthOmics service to create repositories for each upstream registry and to initiate upstream pull requests from the created repositories.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPTCinRegPermissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "omics.amazonaws.com"
            },
            "Action": [
                "ecr:CreateRepository",
                "ecr:BatchImportUpstreamImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-east-1:123456789012:repository/ecr-public/*",
                "arn:aws:ecr:us-east-1:123456789012:repository/docker-hub/*"
            ]
        }
```

```
    ]
}
```

Repository creation template

To use pull through cache in HealthOmics, the Amazon ECR repository must have a repository creation template. The template defines configuration settings for the private repositories created for an upstream registry.

Each template contains a repository namespace prefix, which Amazon ECR uses to match new repositories to a specific template. Templates can specify the configuration for all repository settings including resource-based access policies, tag immutability, encryption, and lifecycle policies. For more information, see Repository creation templates in the *Amazon Elastic Container Registry User Guide*.

In the following example, the policy allows the HealthOmics service to initiate upstream pull requests from the upstream repositories.

JSON

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

Policies for cross-account Amazon ECR access

For cross-account access, the owner of the private repository updates the registry permission policy and the repository creation template to allow access for the other account and that account's run role.

In the registry permission policy, add a policy statement to allow the other account's run role to access the Amazon ECR actions:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCrossAccountPTCinRegPermissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/RUN_ROLE"},
            "Action": [
                "ecr:CreateRepository",
                "ecr:BatchGetImage",
                "ecr:BatchImportUpstreamImage"
            ],
            "Resource": "arn:aws:ecr:us-east-1:123456789012:repository/path/*"
        }
    ]
}
```

In the repository creation template, add a policy statement to allow the other account's run role to access the new container images. Optionally, you can add condition statements to limit access to specific workflows:

JSON

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

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/RUN_ROLE"},
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:omics:us-
east-1:444455556666:workflow/WORKFLOW_ID",
                    "aws:SourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Add permissions for two additional actions (CreateRepository and BatchImportUpstreamImage) in the run role and specify the resource that the run role can access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountPTCRunRolePolicy",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository",
                "ecr:BatchImportUpstreamImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:us-east-1:123456789012::repository/{path}/*"
        }
    ]
```

```
}
```

HealthOmics 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.

For resource permissions related to Amazon ECR, see 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.

JSON

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

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

Permissions for data access using Amazon S3 URIs

You can access sequence store data using HealthOmics API operations or Amazon S3 API operations.

For HealthOmics API access, HealthOmics permissions are managed through an IAM policy. However, S3 Access requires two levels of configuration: explicit allow in the Store's S3 Access Policy and an IAM policy. To learn more about using IAM policies with HealthOmics, see Service roles for HealthOmics.

There are three ways to share the capability of reading objects using the Amazon S3 APIs:

1. Policy based sharing – This sharing requires enabling the IAM principal both in the S3 Access policy and writing an IAM policy and attaching it to the IAM principal. See the next topic for more details.
2. Presigned URLs – You can also generate a shareable pre-signed URL for a file in the sequence store. To learn more about creating presigned URLs using Amazon S3, see Using presigned URLs in the Amazon S3 documentation. The sequence store S3 access policy supports statements for limiting presigned URL capabilities.
3. Assumed roles – Create a role within the data owner's account that has an access policy that allows users to assume that role.

Topics

- Policy based sharing
- Example Restriction

Policy based sharing

If you access sequence store data using a direct S3 URI, HealthOmics provides enhanced security measures for the associated S3 bucket access policy.

The following rules apply to new S3 access policies. For existing policies, the rules apply when you next update the policy:

- The S3 access policies support the following policy elements
  - Version, Id, Statement, Sid, Effect, Principal, Action, Resource, Condition
- The S3 access policies support the following condition keys:
  - s3:ExistingObjectTag/<key>, s3:prefix, s3:signatureversion, s3:TlsVersion
  - Policies also support aws:PrincipalArn with the following condition operators: ArnEquals and ArnLike

If you try to add or update a policy to include an unsupported element or condition, the system rejects the request.

Topics

- Default S3 access policy
- Customizing the access policy
- IAM policy
- Tag-based access control

Default S3 access policy

When you create a sequence store, HealthOmics creates a default S3 access policy granting the data store owner's root account the following permissions for all accessible objects in the sequence store: S3:GetObject, S3GetObjectTagging, and S3:ListBucket. The default created policy is:

JSON

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

```
        {
            "Effect": "Allow",
            "Principal":
            {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action":
            [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890/object/111111111111/
sequenceStore/1234567890/*"
        },
        {
            "Effect": "Allow",
            "Principal":
            {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890/111111111111/
sequenceStore/1234567890/*"
        }
    ]
}
```

Customizing the access policy

If the S3 access policy is blank, no S3 access is allowed. If there is an existing policy and you need to remove s3 access, use `deleteS3AccessPolicy` to remove all access.

To add restrictions on the sharing or to grant access to other accounts, you can update the policy using the `PutS3AccessPolicy` API. Updates to the policy can't go beyond the prefix for the sequence store or the actions specified.

IAM policy

To allow a user or IAM principal access using Amazon S3 APIs, in addition to permission in the S3 access policy, an IAM policy needs to be created and attached to the principal to grant access. 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.

If the sequence store uses a customer managed key (CMK), the principal must also have rights to use the KMS key for decryption. For more information, see Cross-account KMS access in the AWS Key Management Service Developer Guide.

The following example gives a user access to a sequence store. You can fine-tune the access with additional conditions or resource-based filters.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
      "Action":
      [
         "s3:GetObject",
         "s3:GetObjectTagging"
      ],
      "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890/object/111111111111/
sequenceStore/1234567890/*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/omics:readSetStatus": "ACTIVE"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
      "Action": "s3:ListBucket",
```

```
      "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890",
      "Condition": {
        "StringLike": {
          "s3:prefix": "111111111111/sequenceStore/1234567890/*"
        }
      }
    }
  ]
}
```

Tag-based access control

To use tag based access control, the sequence store must first be updated to propagate the tag keys that will be used. This configuration is set during sequence store creation or updating. Once the tags are propagated, tag conditions can be used to further add restrictions. The restrictions can be placed in the S3 Access policy or on the IAM policy. The following is an example of a tab based S3 access policy that would be set:

```
{
    "Sid": "tagRestrictedGets",
    "Effect": "Allow",
    "Principal":
    {
        "AWS": "arn:aws:iam::<target_restricted_account_id>:root"
    },
    "Action":
    [
        "s3:GetObject",
        "s3:GetObjectTagging"
    ],
    "Resource": "arn:aws:s3:us-west-2:222222222222:accesspoint/111111111111-1234567890/
object/111111111111/sequenceStore/1234567890/*",
    "Condition":
    {
        "StringEquals":
        {
            "s3:ExistingObjectTag/tagKey1": "tagValue1",
            "s3:ExistingObjectTag/tagKey2": "tagValue2"
        }
    }
```

```
}
```

Example Restriction

Scenario: Creating a share where the data owner can restrict a user's ability to download "withdrawn" data.

In this scenario, a data owner (account #111111111111) managed a data store. This data owner shares the data with a broad range of third party users, including a researcher (account #999999999999). As part of managing the data, the data owner periodically get requests to withdraw a participants data. To manage this withdrawal, the data owner first restricts direct download access on receiving the request and eventually deletes the data per their requirements.

To meet this need, the data owner sets up a sequence store and each read set receives a tag for "status" that will be set to "withdrawn" if the withdrawal request comes through. For data with the tag set to this value, they want to make sure no user can run "getObject" on this file. To do this setup, the data owner will need to ensure two steps are taken.

Step 1. For the sequence store, ensure that the status tag is updated to be propagated.
This is done by adding the "status" key into the `propogatedSetLevelTags` when calling `createSequenceStore` or `updateSequenceStore`.

Step 2. Update the store's s3 Access Policy to restrict getObject on objects with the status tag set to withdrawn. This is done by updating the stores access policy using the `PutS3AccesPolicy` API. The following policy would allow customers to still see the withdrawn files when listing objects but prevent them from accessing them:

- Statement 1 (restrictedGetWithdrawal): Account 999999999999 can't retrieve objects that are withdrawn.
- Statement 2 (ownerGetAll): Account 111111111111, the data owner, can retrieve all objects, including objects that are withdrawn.
- Statement 3 (everyoneListAll): All shared accounts, 111111111111 and 999999999999, can run the **ListBucket** operation on the whole prefix.

JSON

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

```
    "Statement":
    [
        {
            "Sid": "restrictedGetWithdrawal",
            "Effect": "Allow",
            "Principal":
            {
                "AWS": "arn:aws:iam::999999999999:root"
            },
            "Action":
            [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890/object/111111111111/
sequenceStore/1234567890/*",
            "Condition":
            {
                "StringNotEquals":
                {
                    "s3:ExistingObjectTag/status": "withdrawn"
                }
            }
        },
        {
            "Sid": "ownerGetAll",
            "Effect": "Allow",
            "Principal":
            {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action":
            [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890/object/111111111111/
sequenceStore/1234567890/*",
            "Condition":
            {
                "StringEquals":
                {
```

```
                    "s3:ExistingObjectTag/omics:readSetStatus": "ACTIVE"
                }
            }
        },
        {
            "Sid": "everyoneListAll",
            "Effect": "Allow",
            "Principal":
            {
                "AWS": [
                    "arn:aws:iam::111111111111:root",
                    "arn:aws:iam::999999999999:root"
                ]
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:us-
west-2:222222222222:accesspoint/111111111111-1234567890",
            "Condition":
            {
                "StringLike":
                {
                    "s3:prefix": "111111111111/sequenceStore/1234567890/*"
                }
            }
        }
    ]
}
```

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 in 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. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- 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 140-3 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) 140-3.

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

Topics

- AWS owned keys
- Customer managed keys

- Creating a customer managed key
- Required IAM permissions for using a customer managed key
- Learn more

To protect sensitive customer data at rest, AWS HealthOmics provides encryption by default using a service-owned AWS Key Management Service (AWS KMS) key. Customer managed keys are also supported. To learn more about customer managed key, see Amazon Key Management Service.

All HealthOmics data stores (Storage and Analytics) support the use of customer managed keys. The encryption configuration cannot be changed after a data store has been created. If a data store is using an AWS owned 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.

When you use 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 keys

By default, HealthOmics uses AWS owned keys to automatically encrypt data at rest, because this data can contain sensitive information such as personally identifiable information (PII) or

Protected Health Information (PHI). AWS owned 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 keys to protect your data. You can't view, manage, or access AWS owned 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 aren't charged a monthly fee or a usage fee for using AWS owned keys, and they don't count against the AWS KMS quotas for your account. For more information, see AWS managed keys.

Customer managed keys

HealthOmics supports the use of symmetric customer managed keys 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 HealthOmics sends to AWS KMS on your behalf. Additional AWS KMS charges apply. For more information, see customer managed keys.

Creating 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 keys 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 a 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 a customer managed key with your HealthOmics Analytics resources, the calling principal requires kms:CreateGrant operations in the key policy. This allows the system to use a FAS Token to create 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 HealthOmics requires. See Using grants for more information.

For HealthOmics analytics, the following API operations must be permitted for the calling principal:

- kms:CreateGrant adds grants to a specific customer managed key, which allows access to grant operations in HealthOmics Analytics.
- 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. Also, this action provides customer managed key details that the service can use to validate that the caller has access to use the key.
- kms:Decrypt provides access to read or search operations for encrypted resources.

To use a customer managed key with your HealthOmics storage resources, the HealthOmics service principal and the calling principal must be permitted in the key policy. This allows the service to validate that the caller has access to the key and uses the service principal to execute the store management using the customer managed key. For HealthOmics storage, the key policy for the service principal must permit the following API operations:

- kms:DescribeKey provides the customer managed key details needed to validate the key. This is required for all operations.
- kms:GenerateDataKey provides access to encrypt resources at rest for all write operations. Also, this action provides customer managed key details that the service can use to validate that the caller has access to use the key.
- kms:Decrypt provides access to read or search operations for encrypted resources.

The following example shows a policy statement that allows a service principal to create and interact with a HealthOmics sequence or reference store that is encrypted using the customer managed key:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
{
    "Effect": "Allow",
    "Principal": {
        "Service": "omics.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*"
        }
    ]
}
```

The following example shows a policy that creates permissions for a data store to decrypt data from an Amazon S3 bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "omics:GetReference",
                "omics:GetReferenceMetadata"
            ],
            "Resource": [
                "arn:aws:omics:us-east-1:123456789012:referenceStore/*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::[[s3path]]/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-east-1:123456789012:key/key_id"
            ],
            "Condition": {
              "StringEquals": {
                  "kms:ViaService": [
                    "s3.us-east-1.amazonaws.com"
                  ]
              }
           }
        }
    ]
}
```

Required IAM permissions for using a customer managed key

When creating a resource such as a data store with AWS KMS encryption using a customer managed 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 HealthOmics.

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

Topics

- Analytics API permissions

- Storage API permissions
- How HealthOmics uses grants in AWS KMS
- Monitoring your encryption keys for AWS HealthOmics

Analytics API permissions

For HealthOmics analytics, the IAM user or role that creates the stores must have the kms:CreateGrant, kms:GenerateDataKey, kms:Decrypt, and kms:DescribeKey permissions plus the necessary HealthOmics permissions.

Storage API permissions

For HealthOmics storage APIs, the IAM user or role that calls the following API operations requires the listed permissions:

CreateReferenceStore, CreateSequenceStore

To create a store, the IAM caller must have `kms:DescribeKey` permissions plus the necessary HealthOmics permissions. The HealthOmics service principal calls `kms:GenerateDataKeyWithoutPlaintext` to perform access validation checks for data loading and access.

StartReadSetImportJob, StartReferenceImportJob

To start data import jobs, the IAM caller must have `kms:Decrypt` and `kms:GenerateDataKey` permissions for the KMS key on the store for the import, and `kms:Decrypt` permissions on the Amazon S3 bucket containing the objects to import. In addition, the role passed into the call must have `kms:Decrypt` permissions on the Amazon S3 bucket containing the objects to import. The IAM caller must also have permissions to pass the role to the job.

CreateMultipartReadSetUpload, UploadReadSetPart, CompleteMultipartReadSetUpload

To complete a multi-part upload, the IAM caller must have `kms:Decrypt` and `kms:GenerateDataKey` to create, upload, and complete the multi-part upload.

StartReadSetExportJob

To start a data export job, the IAM caller must have `kms:Decrypt` permission for the KMS key on the store to export from and `kms:GenerateDataKey` and `kms:Decrypt` permissions on the Amazon S3 bucket that receives the objects. In addition, the role passed into the call must

have `kms:Decrypt` permissions on the Amazon S3 bucket that receives the objects. The IAM caller must also have permissions to pass the role to the job.

StartReadsetActivationJob

To start a read set activation job, the IAM caller must have `kms:Decrypt` and `kms:GenerateDataKey` permissions for the objects.

GetReference, GetReadSet

To read objects from the store, the IAM caller must have `kms:Decrypt` permission for the objects.

Read Set S3 GetObject

To read objects from the store using the Amazon S3 `GetObject` API, the IAM caller must have `kms:Decrypt` permission for the objects. Set this permission for both customer managed key and AWS owned key configurations.

How 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 HealthOmics Workflows. HealthOmics Storage uses the customer managed key directly from the service principal, so do not use a grant. When you create an analytics store encrypted with a customer managed key, HealthOmics analytics creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give HealthOmics access to a KMS key in a customer account.

It isn't recommended to revoke or retire the grants that HealthOmics analytics creates on your behalf. If you revoke or retire the grant that gives HealthOmics permission to use the AWS KMS keys in your account, 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 HealthOmics, the change occurs immediately. To revoke access rights, we recommend that you delete the data store rather than revoking the grant. When you delete the data store, 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 key. The log entries in the CloudTrail log show

HealthOmics.amazonAWS.com in the userAgent field to clearly distinguish requests made by HealthOmics.

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

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

You aren't required to create your own grants. HealthOmics creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give 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 in 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 based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS HealthOmics identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS HealthOmics works with IAM)
- **IAM administrator** - write policies to manage access (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 as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM 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 defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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 include 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. You must specify a principal in a resource-based policy.

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.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	No
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how 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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "omics.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:omics:us-east-1:123456789012:*"
        }
      }
    }
```

```
  ]
}
```

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 Define custom IAM permissions with customer managed 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. 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. 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. 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:

```
omics
```

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

```
"Action": [
      "omics:action1",
      "omics: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. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, 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 called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

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 Define permissions with ABAC authorization 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "omics:*"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "Null": {
                  "aws:RequestTag/MyCustomTag": "true"
                }
            }
        }
    ]
}
```

Using Temporary credentials with HealthOmics

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for HealthOmics

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. 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 Create 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. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) 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 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 Validate policies with IAM Access Analyzer 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 Secure API access with MFA 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:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for AWS 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 tothat store through a Resource Share Invitation in the Resource Access Manager (RAM) console. For more information on Resource Share invitations through Lake Formation, see the Cross-account data sharing in Lake Formation. For an Omics admin policy, you also need the following permissions to access your Amazon S3 bucket.

- PutObject
- GetObject
- ListBucket
- AbortMultipartUpload
- ListMultipartUploadParts

JSON

```
{
 "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.

JSON

```
{
    "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 AmazonOmicsFullAccess.	February 23, 2023
HealthOmics started tracking changes	HealthOmics started tracking changes for its AWS managed policies.	November 29, 2022
AmazonOmicsReadOnlyAccess - New policy added	HealthOmics added a new policy that limits access to read only. To learn more, AmazonOmicsReadOnlyAccess.	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 `omics:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 omics: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 `omics:`*`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. The following table shows compliance certifications for the HealthOmics service.

Certification	Link
HIPAA	HIPAA Eligible Services Reference
HiTrust-CSF	Health Information Trust Alliance Common Security Framework

Certification	Link
FedRAMP Moderate (East/West)	Federal Risk and Authorization Management Program
ISO/CSA STAR	ISO and CSA STAR Certified
C5	Cloud Computing Compliance Controls Catalog
DoD CC SRG IL2	Department of Defense Cloud Computing Security Requirements Guide
ENS High	Esquema Nacional de Seguridad
FINMA	Swiss Financial Market Supervisory Authority
ISMAP	Information System Security Management and Assessment Program
OSPAR	Outsourced Service Provider's Audit Report
PCI	Payment Card Industry Data Security Standard
Pinakes	Banking association CCI - Third Party Qualification
PiTuKri	Criteria for Assessing the Information Security of Cloud Services
SOC 1,2,3	System and Organization Controls

For a list of all AWS services in scope for 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.

HealthOmics data stores use the sample ID for internal file naming and for tagging resources. Before you ingest data, check whether the sample ID contains any PHI data. If it does, change the

sample ID before you ingest the data. For more information, see guidance on the AWS HIPAA compliance web page.

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 CSPM – 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

The US East (N. Virginia) and US West (Oregon) regions support AWS PrivateLink FIPS endpoints. For these regions, you can also use the following service names:

- com.amazonaws.*region*.storage-omics-fips
- com.amazonaws.*region*.control-storage-omics-fips
- com.amazonaws.*region*.analytics-omics-fips
- com.amazonaws.*region*.workflows-omics-fips
- com.amazonaws.*region*.tags-omics-fips

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.

> **Note**
>
> For service updates, configure and monitor your Personal Health Dashboard. For more information on how to manage the dashboard, refer to Getting started with your AWS Health Dashboard.

Topics

- S3 access logging
- Monitoring HealthOmics with CloudWatch metrics
- Monitoring HealthOmics with CloudWatch Logs
- Logging AWS HealthOmics API calls using AWS CloudTrail
- Using EventBridge with AWS HealthOmics

S3 access logging

You can monitor Amazon S3 API access to HealthOmics sequence store data using the store-created access logs. You can use CloudWatch to monitor S3 access from HealthOmics API operations . CloudWatch provides visibility into Amazon S3 access originating from your own account. If you, as a data owner, share access to a third party account, access logging isn't available in CloudWatch. Instead, use the store's S3 Access Log. which logs all S3 access to the data in the configured Amazon S3 bucket .

Configure S3 Access Logs using the `CreateSequenceStore` or `UpdateSequenceStore` API operations. Also, make sure that the HealthOmics service principal (`omics.amazonaws.com`) has `s3:PutObject` permissions to the configured S3 prefix.

> **Note**
>
> Logs use the destination bucket's default encryption configuration. If the bucket uses a customer managed key, the service principal must have access to use the key for writing.

To turn off access logging, use `UpdateSequenceStore` and set the access log configuration to blank.

Monitoring HealthOmics with CloudWatch metrics

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, DeleteAnotationStore, 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 **>=**, 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**.

Monitoring HealthOmics with CloudWatch Logs

HealthOmics generates a variety of logs to help you understand and troubleshoot your runs. Logs are available in two places: CloudWatch and Amazon S3.

By default, runs have logging turned on. You can optionally turn off logging for a run by setting `LogLevel = OFF` in the **startrun** request.

> **Note**
>
> For service updates, configure and monitor your Personal Health Dashboard. For more information on how to manage the dashboard, refer to Getting started with your AWS Health Dashboard.

Topics

- Log types for HealthOmics workflows

- Logs in CloudWatch
- Logs in Amazon S3
- Interactive CloudWatch Logs in the CLI
- Accessing CloudWatch Logs from the console

Log types for HealthOmics workflows

HealthOmics provides the following types of logs for workflows:

- Engine logs – The underlying workflow engines (Nextflow, WDL, and CWL) produce engine logs for runs. These logs can help you troubleshoot workflow definition issues.
- Run manifest logs – These logs provide high level information about each run task, such as task status, start time, stop time, and fail reason (if the task failed).

  Run manifest logs also report resource utilization statistics that can be helpful for understanding resource optimization opportunities. These statistics include:

  - cpusAverage
  - cpusMaximum
  - cpusReserved
  - gpusReserved
  - memoryAverageGiB
  - memoryMaximumGiB
  - memoryReservedGiB
  - runningSeconds
- Run logs – Run logs provide the overall run status and the time when individual tasks are starting, running, stopping, and completed. Run logs also give you visibility into file import and export steps.
- Task logs – Task logs provide detailed logging information about individual tasks in your run. The outputs in your task log depend on the task definition and where you use log statements in your code. If your task logs don't provide the level of insight you need, consider adding additional log statements to your task definition to produce more insightful task logs.
- Run cache logs – Run cache logs provide the overall status of run caches and the caching of task outputs. Run cache logs give you visibility into cache hits and misses for each run that uses caching.

- Outputs.json – For WDL and CWL workflows, HealthOmics delivers an engine-generated file, named `outputs.json`, to your Amazon S3 bucket after run completion. This files includes a list and a map of all outputs for the run.

Logs in CloudWatch

CloudWatch generates workflow logs for failed runs and successful runs. All logs are available for failed runs and successful runs, except engine logs are available only for failed runs.

You can find the CloudWatch workflow logs in the following log group: `/aws/omics/WorkflowLog`. Also, the output of the **get-run** API operation provides the CloudWatch log stream ARNs for the engine logs and run logs.

By default, AWS keeps the CloudWatch Logs indefinitely. You can adjust the retention policy for the log group to set a retention period between 10 years and one day.

The following table provides a summary of the CloudWatch Logs in HealthOmics. All workflow logs are available for successful runs and failed runs, except engine logs are available only for failed runs.

Log name	Available in CloudWatch Logs	When is log available	Log stream format
Engine logs	Yes, for failed runs	After run completes	run/*runID*/engine
Run manifest logs	Yes	After run completes	manifest/run/*runID*/*runUUID*
Run logs	Yes	In real time	run/*runID*
Task logs	Yes	In real time	run/*runID*/task/*taskID*
Run cache logs	Yes	In real time	runCache/*runCacheId* /*runCacheUUID*
Outputs.json (WDL and CWL)	No	n/a	n/a

Logs in Amazon S3

Only the engine logs and the `outputs.json` file are delivered to Amazon S3.

After a run completes, the engine logs are delivered to your S3 bucket and are available indefinitely until you delete them. These logs are located in the logs directory of the S3 output URI that you specified for the workflow.

The path to the logs directory has the following format: `s3://{user_provided_path}/logs/`.

The following table provides a summary of the HealthOmics logs available in your Amazon S3 bucket.

Log name	Available in Amazon S3	When is log available	Log stream path
Engine logs	Yes	After run completes	s3://*user_provided_path* /logs/engine.log
Outputs.json (WDL and CWL)	Yes	After run completes	s3://*user_provided_path* /*runID*/*runUUID*/logs/outputs.json
Run manifest logs, run logs, and task logs	No	n/a	n/a

Interactive CloudWatch Logs in the CLI

You can interactively view the CloudWatch Logs using the Live Tail command in interactive mode. You can track run progress in real time and define up to 5 keywords to highlight in the logs:

```
aws logs start-live-tail  \
  --mode interactive  \
  --log-group-identifiers arn:aws:logs:region:account-ID:log-group:/aws/omics/
WorkflowLog
```

For more information, see Start live tail in the AWS CLI Command Reference.

Accessing CloudWatch Logs from the console

To access the logs for a run, you can link directly to these logs from the **Run details** page in HealthOmics console.

1. Open the HealthOmics console.
2. If required, open the left navigation pane (≡). Choose **Runs**.
3. Select the run from the Runs table.
4. In the run details page, you can choose any of these actions:

    a. From **Run summary**, choose **View run logs**. The console opens the run logs in the CloudWatch console.

    b. From **Run summary**, choose **View logs in Amazon S3**. The console opens the logs folder in the Amazon S3 console.

    c. From **Run tasks**, choose **View logs**, **View run logs** or **View run manifest logs** for a task. The console opens the logs in the CloudWatch console.

You can also navigate to the logs from the CloudWatch console:

1. Open the CloudWatch console https://console.aws.amazon.com/cloudwatch/.
2. From the left menu, choose **Log groups**.
3. Select the `/aws/omics/WorkflowLog` group.

    If the list of log groups is long, you can enter **omics** in the search text box to narrow down the list.
4. When the **Log group details** page opens, choose the log stream you want to view. The console displays the events for this log stream.

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 `CreateReferenceeStore`, `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 botocore/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 sends events to Amazon EventBridge when resources change status. Resources include import jobs, export jobs, resource shares, workflows, tasks, and runs. For each type of resource, there is a list of status changes that generate an event.

An event bus is a router that receives events and delivers them to destinations. Your account includes a default event bus that automatically receives events from AWS services. You can create additional custom event buses.

You create EventBridge rules to specify the actions to take when the event bus receives events. For example, you can create a rule that notifies you about status changes for a resource.

Common scenarios for using events include:

- To monitor when a user shares a resource with you or revokes the share.
- To monitor whether a run fails or completes successfully.

For more information about using EventBridge, see What is Amazon EventBridge?

Topics

- Set up EventBridge for HealthOmics
- EventBridge events in HealthOmics
- Event message structure
- Event message examples

Set up EventBridge for HealthOmics

Before you can monitor for EventBridge events, create an EventBridge bus and create rules for the events of interest.

Configure an EventBridge bus

You can use the default event bus for your AWS account or configure a custom event bus. To configure a custom event bus, follow these steps:

1. Open the EventBridge console: https://console.aws.amazon.com/events/.
2. In the left navigation, choose **Event buses**.
3. Choose **Create event bus**.
4. In the **Create event bus** form, enter a name for the bus.
5. Choose **Create** to create the bus.

Create an EventBridge rule

The following procedure shows how to create a simple rule. For more information about rules, see Rules in EventBridge.

1. Open the EventBridge console: https://console.aws.amazon.com/events/.
2. In the left navigation, choose **Rules**.
3. Choose **Create rule**. The console opens the **Create rule** form.
4. In **Define rule detail**, provide a name for the rule.
   - For **Name**, enter a name for the bus.
   - For **Event bus**, select the bus for this rule.
   - Choose **Next**.

5. In **Build event pattern**, under **Event source** select **AWS events or EventBridge partner events**.
6. Scroll down to **Event pattern**.
   a. For **Event source**, select **AWS services**.
   b. For **AWS service**, enter **omics** in the text filter and select **AWS HealthOmics** as the service.
   c. For **Event type** select the event of interest (or **All events**).
   d. Choose **Next**.
7. In **Select target(s)**, select a target for the event. For example, choose **AWS service**, the chose **CloudWatch log group**, and configure a log group.

   For many target types, EventBridge needs permission to send events to the target. The console creates these permissions for you.
8. (Optional) In **Configure tags**, associate tags with the rule.
9. In **Review and update**, review the configuration and choose **Create rule**.

EventBridge events in HealthOmics

The following table lists the events that HealthOmics sends to EventBridge, and the list of possible status values for the event.

Event name	Possible status values
Annotation Import Job Status Change	Submitted, in progress, cancelled, completed, failed, or completed with failures
Annotation Store Share Status Change	Pending, activating, active, deleting, deleted, failed
Annotation Store Status Change	Creating, created, updating, updated, deleting, deleted, or creation failed
Read Set Activation Job Status Change	Submitted, in progress, completed, failed, or completed with failures
Read Set Export Job Status Change	Submitted, in progress, completed, failed, or completed with failures

Event name	Possible status values
Read Set Import Job Status Change	Submitted, in progress, completed, failed, or completed with failures
Read Set Status Change	Processing upload, upload failed, active, archived, activating, or deleted
Reference Import Job Status Change	Submitted, in progress, completed, failed, or completed with failures
Reference Status Change	Active or deleted
Reference Store Status Change	Created, updated, active, or deleted
Run Status Change	Pending, starting, running, stopping, completed, deleted, failed, or cancelled
Sequence Store Status Change	Created, updated, active, or deleted
Task Status Change	Pending, starting, running, stopping, completed, deleted, failed, or cancelled
Variant Import Job Status Change	Submitted, in progress, cancelled, completed, failed, or completed with failures
Variant Store Share Status Change	Pending, activating, active, deleting, deleted, failed
Variant Store Status Change	Creating, created, updating, updated, deleting, deleted, or creation failed
Workflow Share Status Change	Pending, activating, active, deleting, deleted, failed
Workflow Status Change	Creation success, creation failure, deletion success, or deletion failure

Event message structure

HealthOmics provides best effort delivery to send state change event messages 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 to learn more information. Events include the following fields:

- `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.

Run events include the following fields:

- `uuid` — The universally unique identifier for the run.
- `workflowId` — Workflow identifier of the workflow associated with this run.
- `workflowName` — Name of the workflow associated with this run..
- `runId` — Run identifier.
- `runName` — Run name.
- `runOutputUri` — The URI for where the run will write its output data.

Event message examples

The following example is an event for a change in run status, showing the additional fields.

```
{
    "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",
        "runId": "1234567",
        "runName": "run name",
        "runOutputUri": "s3://amzn-s3-demo-bucket/run-output/2101313",
        "workflowId": "1234567",
        "workflowName": "workflow name"
    }
}
```

The following example is an event for a change in task status.

```
{
    "version": "0",
    "id": "718d6817-c868-26d3-8ef0-0dc9b2ac73f4",
    "detail-type": "Task Status Change",
    "source": "aws.omics",
    "account": "123456789012",
    "time": "2024-10-30T09:05:44Z",
    "region": "us-west-2",
    "resources": ["arn:aws:omics:us-west-2:123456789012:task/8888888"],
    "detail": {
        "omicsVersion": "1.0.0",
        "arn": "arn:aws:omics:us-west-2:123456789012:task/8888888",
        "status": "COMPLETED",
        "runArn": "arn:aws:omics:us-west-2:123456789012:run/2101313",
        "runUuid": "153893cd-097a-40ec-aec7-838a97cd2b21",
        "runId": "1234567",
        "runName": "run name",
        "workflowId": "1234567",
        "workflowName": "workflow name"
    }
}
```

The following is an example of an event for a read set status change.

```
{
  "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:myvariantstore2"],
    "detail": {
       "omicsVersion": "1.0.0",
       "arn": "arn:aws:omics:us-east-1:123456789012:myvariantstore2",
       "status": "CREATED",
       "storeId": "6710c5f02610",
       "storeName": "myvariantstore2"
    }
}
```

The following is an event 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:my_variant_store/
b64ea9a3-459f-4b68-92c3-3ddb83209fe9"],
    "detail": {
       "omicsVersion": "1.0.0",
       "arn": "arn:aws:omics:us-east-1:123456789012:my_variant_store/
b64ea9a3-459f-4b68-92c3-3ddb83209fe9",
       "status": "COMPLETED",
       "jobId": "b64ea9a3-459f-4b68-92c3-3ddb83209fe9",
       "storeId": "a74869f91e20",
       "storeName": "my_variant_store"
    }
}
```

Troubleshooting

The following topics can help you troubleshoot issues that you encounter when using HealthOmics workflows and data stores.

Topics

- Troubleshooting workflows
- Troubleshooting call caching issues
- Troubleshooting data stores
- Troubleshooting with Amazon Q CLI

Troubleshooting workflows

Topics

- How do I troubleshoot a failed run?
- How do I troubleshoot a failed task?
- Where do I find the engine logs for successfully completed runs?
- How can I reduce the input parameter size for a workflow?
- Why is my run not completing?

How do I troubleshoot a failed run?

Use the **GetRun** API operation to retrieve the failure reason. For more information, see Run failure reasons.

How do I troubleshoot a failed task?

Review the error code from the task failure message to understand the failure. Review the task logs in CloudWatch to see detailed logging messages for the task. If you aren't getting detailed log messages, you can revise your workflow to output additional log statements. For more information, see Monitoring HealthOmics with CloudWatch Logs.

Where do I find the engine logs for successfully completed runs?

HealthOmics publishes logs to CloudWatch for failed runs only. If a run completes successfully, HealthOmics delivers the engine logs to your Amazon S3 bucket. For more information, see Logs in Amazon S3.

How can I reduce the input parameter size for a workflow?

You can specify up to 50 KB of input parameters for a workflow. You can use directory imports or sample sheets to remain within this size constraint. For more information, see Managing run parameters size.

Why is my run not completing?

If there are issues with your code and the processes have not exited properly, your run could become unresponsive or "stuck". For more information on how to prevent and catch unresponsive runs, see Guidance for unresponsive runs.

Troubleshooting call caching issues

The following topics can help you troubleshoot issues that you encounter with call caching.

Topics

- Why isn't my run saving to the cache?
- Why isn't a task using the cache entry?
- Why is the call caching for a task disabled?

Why isn't my run saving to the cache?

1. Verify that the run is configured to use a cache by checking the cacheId field in the GetRun API operation response. Using the CLI, run this command: `aws omics get-run –id <run_id>`.
2. If the run was successful, verify the cache behavior returned in the GetRun response is CACHE_ALWAYS. If the cache behavior is set to CACHE_ON_FAILURE, runs will only save to the cache when they fail.

Why isn't a task using the cache entry?

In the `/aws/omics/WorkflowLog` CloudWatch log group, open the log stream for the run cache: **runCache/<cache_id>/<cache_uuid>**.

1. Verify that a previous run created a cache entry for the task that you expected to be cached. Runs that have saved to the cache will be recorded with a log message of CACHE_ENTRY_CREATED.
2. Locate the CACHE_MISS log for the task and run that completed. If there is no log entry, check that the run was configured to use the cache.
3. If a cache entry was created, verify that the CPUs, memory, GPUs and container digest are identical for both tasks. The task ARN for the task that created the cache entry is in the log message.
4. If the compute requirements for both tasks match, verify that the inputs have not changed between the tasks. To do this, open the engine logs. If the run has a status of FAILED, the logs will be in Cloudwatch Log Group /aws/omics/WorkflowLog. Otherwise the engine logs can be found in the output directory of the run.

Why is the call caching for a task disabled?

Check if the task is configured to opt out of caching using workflow engine features:

- For WDL workflows: Check if the task has volatile set to `true` in the meta section
- For Nextflow workflows: Check if the task has cache directive set to `false`
- For CWL workflows: Check if the task has enableReuse set to `false` for the WorkReuse feature

Troubleshooting data stores

Topics

- Why is S3 GetObject failing on my read set?
- Why can't I see my annotation store or variant store in Athena?
- Why can't I access my data store in Athena?

Why is S3 GetObject failing on my read set?

Most commonly, the failure is due to a missing permission. Sequence store S3 reading permission is a bi-directional configuration requiring both the sequence store S3 access policy to allow access and the IAM principal to have a policy attached allowing access. For more detail on the policy requirements see Permissions for data access using Amazon S3 URIs. Check that the following configurations are in place:

- The sequence store S3 access policy has explicitly allowed access to the IAM principal or the root of the principal's account.
- Check that the IAM principal has a policy explicitly providing permission to the resource being accessed. Note that the IAM principal policy must use the Access Point ARN and not the Access point Alias based path when defining permissions and that the ARN is in the condition and not used to specify a resource.
- If your store uses a customer managed key (CMK-KMS), ensure that the IAM principal has kms:decrypt permissions on the key. See the KMS cross-account access guide for configuring usage across accounts.

If you have a policy that's using tag based access controls, ensure the following:

- Ensure that the sequence store has finished synchronizing the tags. For this, the store's status needs to be **active** and not **updating**.
- Ensure that there are no typos in the tag key or key value on the read set and the policy.

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. For more information, see Configuring Lake Formation to use HealthOmics.

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.

Troubleshooting with Amazon Q CLI

Amazon Q CLI can help streamline your troubleshooting process by:

- Analyzing workflow runs and debug task failures
- Collecting relevant logs and error messages
- Creating AWS Support cases with all necessary debugging logs attached
- Redacts personal identifiable information (PII) from information submitted to AWS Support

For more information about using Amazon Q CLI with AWS HealthOmics for troubleshooting and creating support cases, see the HealthOmics Agentic generative AI tutorial on GitHub.

Warning

When working with Amazon Q CLI, review all generated content and proposed actions before proceeding. Provide feedback to improve response quality and to match your workflow's requirements. For more information, see Security considerations and best practices for Amazon Q.

Quotas for AWS HealthOmics

AWS populates your account with default values for the HealthOmics quotas. Unless otherwise noted, each quota value is the per-Region maximum value.

> **Important**
>
> You can request increases to most of the service quotas and API quotas. See the following topics for details.

Topics

- HealthOmics service quotas
- HealthOmics fixed 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.

> **Important**
>
> You can request an increase to an adjustable quota using the Service Quotas console.

For more information about service quotas, see Requesting a quota increase in the *Service Quotas User Guide*. For a quota that isn't available in Service Quotas console, use the quota increase form.

Name	Default	Adjuse	Description
Analytics - Maximum annotation stores	Each supported Region: 10	Yes	The maximum number of annotation stores in the current AWS region

Name	Default	Adjustable	Description
Analytics - Maximum concurrent variant or annotation store import jobs	Each supported Region: 5	Yes	The maximum number of concurrent import jobs in the current AWS region
Analytics - Maximum files per variant store import job	Each supported Region: 1,000	Yes	The maximum number of files per variant import job in the current AWS region
Analytics - Maximum shares per annotation store	Each supported Region: 10	Yes	The maximum number of shares per annotation store in the current AWS region
Analytics - Maximum shares per variant store	Each supported Region: 10	Yes	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	Yes	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	Yes	The maximum size of one file in an annotation import job in the current AWS region
Analytics - Maximum variant stores	Each supported Region: 10	Yes	The maximum number of variant stores in the current AWS region

Name	Default	Adjustable	Description
Analytics - Maximum versions per annotation store	Each supported Region: 10	Yes	The maximum number of versions per annotation store in the current AWS region
Configurations - Maximum configurations	Each supported Region: 10	Yes	The maximum number of configurations in the current AWS region.
Storage - Maximum concurrent read set activation jobs	Each supported Region: 25	Yes	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	Yes	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	Yes	The maximum number of concurrent import jobs for a sequence or reference store in the current AWS region
Storage - Maximum read sets per sequence store	Each supported Region: 1,000,000	Yes	The maximum number of read sets in a sequence store in the current AWS region
Storage - Maximum references per reference store	Each supported Region: 50	Yes	The maximum number of references in a reference store in the current AWS region

Name	Default	Adjustable	Description
Storage - Maximum sequence stores	Each supported Region: 20	Yes	The maximum number of sequence stores in the current AWS region
Workflows - Maximum active GPUs	Each supported Region: 12	Yes	The maximum number of concurrent active GPUs in the current AWS region. In us-east-1 and us-west-2, quota increase requests for values up to 500 are automatically approved.
Workflows - Maximum concurrent active runs using dynamic run storage	Each supported Region: 50	Yes	The maximum number of active runs using dynamic run storage in the current AWS region. Quota increase requests for values up to 200 are automatically approved.
Workflows - Maximum concurrent active runs using static run storage	Each supported Region: 10	Yes	The maximum number of active runs using static run storage in the current AWS region. Quota increase requests for values up to 50 are automatically approved.

Name	Default	Adjustable	Description
Workflows - Maximum concurrent tasks per run	Each supported Region: 25	Yes	The maximum number of concurrent tasks in each run in the current AWS region. In us-east-1 and us-west-2, quota increase requests for values up to 100 are automatically approved.
Workflows - Maximum run duration	Each supported Region: 604,800 Seconds	Yes	The maximum workflow run duration in the current AWS region.
Workflows - Maximum runs (active or inactive)	Each supported Region: 100,000	Yes	The maximum number of runs (active or inactive) in the current AWS region.
Workflows - Maximum shares per workflow	Each supported Region: 100	Yes	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	Yes	The maximum static run storage capacity in gibibytes (GiB) for each run in the current AWS region. In us-east-1 and us-west-2, quota increase requests for values up to 50,000 are automatically approved.
Workflows - Maximum workflows	Each supported Region: 1,000	Yes	The maximum number of workflows in the current AWS region.

Name	Default	Adjuse	Description
Workflows - Transactions per second (TPS) for the StartRun operation	Each supported Region: 1	Yes	The maximum transactions per second (TPS) for the StartRun operation in the current AWS region.

HealthOmics fixed size quotas

In addition to the HealthOmics service quotas, HealthOmics includes quotas that have fixed sizes. You cannot request an increase for these values.

Unless otherwise noted, each quota lists the maximum value per-Region.

Topics

- HealthOmics analytics fixed size quotas
- HealthOmics storage fixed size quotas
- HealthOmics workflow fixed size quotas
- HealthOmics Ready2Run workflow fixed size quotas

HealthOmics analytics fixed size quotas

The following table shows the maximum supported values for analytics quotas. These values aren't adjustable.

Name	Description	Maximum	Adjustable Yes/No
Analytics - Maximum files per annotation store import job	The maximum number of files per annotation import job.	1	No

HealthOmics storage fixed 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 S3 access resource policy size	The maximum size of the S3 access resource policy	15 KB	No
Storage - Maximum propagated set level tags	The maximum number of set level tag keys, per store, that propogate to the S3 object	5	No
Storage - Maximum read sets per activation job	The maximum number of read sets per activation job.	20	No
Storage - Maximum read sets per export job	The maximum number of read sets per export job.	100	No
Storage - Maximum read sets per import job	The maximum number of read sets per import job.	100	No
Storage - Maximum reference stores	The maximum number of reference stores.	1	No
Storage - Maximum part size for a direct upload	The maximum part size for direct upload to a sequence store.	100 MB	No

Name	Description	Maximum	Adjustable Yes/No
Storage - Maximum parts in file for direct upload	The maximum number of parts in a file for direct upload to a sequence store.	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

HealthOmics workflow fixed size quotas

The following table shows the maximum supported values for workflow quotas. These values aren't adjustable.

Name	Description	Maximum size	Adjustable Yes/No
Workflows - Maximum run groups	The maximum number of run groups.	1000	No
Workflows - Maximum run caches	The maximum number of run caches that you can create for one account. One or more runs can share the same run cache. There	1000	No

Name	Description	Maximum size	Adjustable Yes/No
	is no quota for the number of runs that HealthOmics can cache per account.		
Workflows - Maximum workflow versions	The maximum number of workflow versions per workflow.	1000	No
Workflows - CPU instance container size	The maximum container image size for a CPU instance.	45 GiB	No
Workflows - GPU instance container size	The maximum container image size for a GPU instance.	95 GiB	No
GPU instance /dev/shm shared memory	The maximum amount of shared memory per GPU instance.	8 GB per GPU	No
Workflows - Run parameter file	The maximum size of a run parameter file.	50,000 bytes	No
Workflows - Workflow parameters template file	The maximum number of entries and maximum file size for a workflow parameters template file. This quota applies to workflows that you create using the console or API.	1,000 entries, 400 KB	No

Name	Description	Maximum size	Adjustable Yes/No
Workflows - Workflow definition file size - API	The maximum size of the workflow definition file when you create the workflow using the API operation or an AWS SDK.	100 MB	No
Workflows - Workflow definitio n file size - Console (direct upload)	The maximum size of the workflow definition file that you can provide as a direct upload, when you create the workflow using the console.	4.4 MB	No
Workflows - Workflow definitio n file size - Console (upload from Amazon S3)	The maximum size of the workflow definition file that you can provide as an upload from Amazon S3, when you create the workflow using the console.	100 MB	No
Workflows - Repository size	The maximum size of an external code repository.	1 GiB	No
Workflows - Repository individual file size	The maximum size of an individual file from an external code repository.	100 MiB	No

Name	Description	Maximum size	Adjustable Yes/No
Workflows - README file size	The maximum size of a README file.	500 KiB	No

For suggestions on how to reduce the size of your run parameter file, see Managing run parameters size.

HealthOmics Ready2Run workflow fixed size quotas

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

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/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
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

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/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
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

Ready2Run workflow name	Maximum input file size (GiB)	Adjustable (Yes/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.

Topics

- General API quotas
- Storage API quotas
- Workflow API quotas
- Analytics API quotas

General API quotas

The following table lists general API operations that apply to more than one category (storage, workflows, and analytics).

API operation	Default maximum TPS	Adjustable (Yes/No)
AcceptShare, CreateShare, DeleteShare, GetShare, ListShares	1 TPS	Yes

Storage API quotas

The following table lists the storage API operations.

Storage API operation	Default maximum TPS	Adjustable (Yes/No)
CreateSequenceStore, UpdateSequenceStore, DeleteSequenceStore, CreateReferenceStore, DeleteReferenceStore	1 TPS	Yes
BatchDeleteReadSet, DeleteReference	1 TPS	Yes
CreateMultipartReadSetUpload, CompleteMultipartReadSetUpload, AbortMultipartReadSetUpload	1 TPS	No
GetS3AccessPolicy, PutS3AccessPolicy, DeleteS3AccessPolicy	1 TPS	Yes
GetReference	10 TPS	Yes
UploadReadSetPart	10 TPS	Yes
GetReadSet	30 TPS	Yes
GetSequenceStore, ListSequenceStores	5 TPS	Yes

Storage API operation	Default maximum TPS	Adjustable (Yes/No)
GetReadSetMetadata, ListReadSets	5 TPS	Yes
StartReadSetImportJob, GetReadSetImportJob, ListReadSetImportJobs	5 TPS	Yes
StartReadSetExportJob, GetReadSetExportJob, ListReadSetExportJobs	5 TPS	Yes
ListReferenceStores	5 TPS	Yes
StartReferencetImportJob, GetReferenceImportJob, ListReferenceImportJobs	5 TPS	Yes
ListReferences, GetReference Metadata	5 TPS	Yes
StartReadsetActivationJob	5 TPS	Yes
ListReadsetActivationJobs, GetReadSetActivationJob	5 TPS	Yes
ListMultipartReadSetUploads, ListReadSetUploadParts	5 TPS	Yes
TagResource, UntagResource, ListTagsForResource	5 TPS	Yes

Workflow API quotas

The following table lists the workflow API operations.

Workflow API operation	Default maximum TPS	Adjustable (Yes/No)
StartRun	1 TPS	Yes
CreateWorkflow	5 TPS	Yes
CancelRun, DeleteRun , GetRun, GetRunTask, ListRunTasks, ListRuns	10 TPS	Yes
CreateRunGroup, DeleteRun Group, GetRunGroup, ListRunGroups, UpdateRun Group	10 TPS	Yes
CreateRunCache, UpdateRun Cache, DeleteRunCache, GetRunCache, ListRunCaches	10 TPS	Yes
DeleteWorkflow, GetWorkfl ow, ListWorkflows, UpdateWorkflow	10 TPS	Yes

Analytics API quotas

The following table lists the analytics API operations.

Analytics API operation	Default maximum TPS	Adjustable (Yes/No)
CreateVariantStore, DeleteVar iantStore, GetVariantStore, ListVariantStores, UpdateVar iantStore	1 TPS	No
StartVariantImportJob, CancelVariantImportJob,	1 TPS	No

Analytics API operation	Default maximum TPS	Adjustable (Yes/No)
GetVariantImportJob, ListVariantImportJobs		
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
AWS HealthOmics variant stores and annotation stores are no longer open to new customers.	AWS HealthOmics variant stores and annotation stores are no longer open to new customers. For more informati on, see AWS HealthOmics variant store and annotation store availability change.	November 7, 2025
AWS HealthOmics variant stores and annotation stores will no longer be open to new customers starting November 7th, 2025.	AWS HealthOmics variant stores and annotation stores will no longer be open to new customers starting November 7th, 2025. If you would like to use variant stores or annotation stores, sign up prior to that date. Existing customers can continue to use the service as normal. For more information, see AWS HealthOmics variant store and annotation store availabil ity change.	October 7, 2025
New Features	HealthOmics added support for workflows to synchroni ze a private Amazon ECR repository with an upstream registry. To learn more, see Container images for private workflows in HealthOmics.	August 28, 2025

New README and repository integration features	Added support for creating workflows from external code repositories and README files.	July 24, 2025
New Features	HealthOmics added support for Nextflow automatic parameter interpolation. To learn more, see Parameter template files for HealthOmics workflows.	June 27, 2025
New Features	HealthOmics added support for workflows to interpolate the run parameters from a WDL workflow definition file. To learn more, see Parameter template files for HealthOmics workflows.	May 30, 2025
New Features	HealthOmics added support for workflow versioning. To learn more, see Workflow versioning in HealthOmics.	April 18, 2025
New Features	HealthOmics added elastic throughput for dynamic run storage. To learn more, see Run storage types in HealthOmics.	April 16, 2025

New Features	HealthOmics added attribute based access controls for Sequence Store S3 locations , and the abilty to synchroni ze up to five read-set tags to a Sequence Store S3 object. To learn more, see Creating a HealthOmics sequence store.	November 22, 2024
New Features	HealthOmics added support for call caching, also known as resume, for private workflows . To learn more, see Call caching.	November 20, 2024
New Features	HealthOmics added new API fields to help you map between sequence store input jobs and read sets.	August 29, 2024
New Features	HealthOmics added support for managing Nextflow versions. To learn more, see Nextflow versions.	August 14, 2024
New Features	HealthOmics added support for shared workflows and dynamic run storage.	April 30, 2024
New Features	HealthOmics added support for Amazon S3 access to reference and sequence stores, and support for SHA256 ETags.	April 15, 2024
New Features	HealthOmics added entity tags (ETags) for sequence stores.	October 6, 2023

New Features	HealthOmics added annotation store versioning and analytic store sharing.	August 15, 2023
New Features	HealthOmics added Common Workflow Language (CWL) as a supported language for HealthOmics workflows.	June 30, 2023
New Features	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