User Guide

Amazon Athena

Amazon Athena: User Guide

Table of Contents

What is Amazon Athena?

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard SQL. With a few actions in the AWS Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

For more information, see Get started.

Amazon Athena also makes it easy to interactively run data analytics using Apache Spark without having to plan for, configure, or manage resources. When you run Apache Spark applications on Athena, you submit Spark code for processing and receive the results directly. Use the simplified notebook experience in Amazon Athena console to develop Apache Spark applications using Python or Use Athena notebook APIs.

For more information, see Get started with Apache Spark on Amazon Athena.

Athena SQL and Apache Spark on Amazon Athena are serverless, so there is no infrastructure to set up or manage, and you pay only for the queries you run. Athena scales automatically—running queries in parallel—so results are fast, even with large datasets and complex queries.

Topics

- When should I use Athena?
- Client and programming tools for using Athena
- Set up, administrative, and programmatic access
- AWS service integrations with Athena

When should I use Athena?

Query services like Amazon Athena, data warehouses like Amazon Redshift, and sophisticated data processing frameworks like Amazon EMR all address different needs and use cases. The following guidance can help you choose one or more services based on your requirements.

Amazon Athena

Athena helps you analyze unstructured, semi-structured, and structured data stored in Amazon S3. Examples include CSV, JSON, or columnar data formats such as Apache Parquet and Apache ORC.

You can use Athena to run ad-hoc queries using ANSI SQL, without the need to aggregate or load the data into Athena.

Athena integrates with Amazon QuickSight for easy data visualization. You can use Athena to generate reports or to explore data with business intelligence tools or SQL clients connected with a JDBC or an ODBC driver. For more information, see What is Amazon QuickSight in the *Amazon QuickSight User Guide* and Connect to Amazon Athena with ODBC and JDBC drivers.

Athena integrates with the AWS Glue Data Catalog, which offers a persistent metadata store for your data in Amazon S3. This allows you to create tables and query data in Athena based on a central metadata store available throughout your Amazon Web Services account and integrated with the ETL and data discovery features of AWS Glue. For more information, see Use AWS Glue Data Catalog to connect to your data and What is AWS Glue in the *AWS Glue Developer Guide*.

Amazon Athena makes it easy to run interactive queries against data directly in Amazon S3 without having to format data or manage infrastructure. For example, Athena is useful if you want to run a quick query on web logs to troubleshoot a performance issue on your site. With Athena, you can get started fast: you just define a table for your data and start querying using standard SQL.

You should use Amazon Athena if you want to run interactive ad hoc SQL queries against data on Amazon S3, without having to manage any infrastructure or clusters. Amazon Athena provides the easiest way to run ad hoc queries for data in Amazon S3 without the need to setup or manage any servers.

For a list of AWS services that Athena leverages or integrates with, see the section called “AWS service integrations”.

SageMaker Unified Studio

Amazon SageMaker Unified Studio makes it simple to work with Amazon Athena and Amazon Redshift to run SQL queries on SageMaker Lakehouse data. With Unified Studio, you can develop SQL queries, work with query results, and collaborate with your team through an integrated notebook environment. You can also use Amazon Q generative SQL to generate SQL code from natural language input. To learn more, see SQL Analytics in the SageMaker Unified Studio user guide.

Amazon EMR

Amazon EMR makes it simple and cost effective to run highly distributed processing frameworks such as Hadoop, Spark, and Presto when compared to on-premises deployments. Amazon EMR is flexible – you can run custom applications and code, and define specific compute, memory, storage, and application parameters to optimize your analytic requirements.

In addition to running SQL queries, Amazon EMR can run a wide variety of scale-out data processing tasks for applications such as machine learning, graph analytics, data transformation, streaming data, and virtually anything you can code. You should use Amazon EMR if you use custom code to process and analyze extremely large datasets with the latest big data processing frameworks such as Spark, Hadoop, Presto, or Hbase. Amazon EMR gives you full control over the configuration of your clusters and the software installed on them.

You can use Amazon Athena to query data that you process using Amazon EMR. Amazon Athena supports many of the same data formats as Amazon EMR. Athena's data catalog is Hive metastore compatible. If you use EMR and already have a Hive metastore, you can run your DDL statements on Amazon Athena and query your data immediately without affecting your Amazon EMR jobs.

Amazon Redshift

A data warehouse like Amazon Redshift is your best choice when you need to pull together data from many different sources – like inventory systems, financial systems, and retail sales systems – into a common format, and store it for long periods of time. If you want to build sophisticated business reports from historical data, then a data warehouse like Amazon Redshift is the best choice. The query engine in Amazon Redshift has been optimized to perform especially well on running complex queries that join large numbers of very large database tables. When you need to run queries against highly structured data with lots of joins across lots of very large tables, choose Amazon Redshift.

For more information about when to use Athena, consult the following resources:

- Decision guide for analytics services on AWS in the *Getting Started Resource Center*
- When to use Athena vs other big data services in the *Amazon Athena FAQs*
- Amazon Athena overview
- Amazon Athena features
- Amazon Athena FAQs
- Amazon Athena blog posts

Client and programming tools for using Athena

You can access Athena using a variety of client and programming tools. These tools include the AWS Management Console, a JDBC or ODBC connection, the Athena API, the Athena CLI, the AWS SDK, or AWS Tools for Windows PowerShell.

- To get started using Athena SQL with the console, see Get started.
- To get started creating Jupyter compatible notebooks and Apache Spark applications that use Python, see Use Apache Spark in Amazon Athena.
- To learn how to use JDBC or ODBC drivers, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.
- To use the Athena API, see the Amazon Athena API Reference.
- To use the CLI, install the AWS CLI and then type `aws athena help` from the command line to see available commands. For information about available commands, see the Amazon Athena command line reference.
- To use the AWS SDK for Java 2.x, see the Athena section of the AWS SDK for Java 2.x API Reference, the Athena Java V2 examples on GitHub.com, and the AWS SDK for Java 2.x Developer Guide.
- To use the AWS SDK for .NET, see the `Amazon.Athena` namespace in the AWS SDK for .NET API Reference, the .NET Athena examples on GitHub.com, and the AWS SDK for .NET Developer Guide.
- To use AWS Tools for Windows PowerShell, see the AWS Tools for PowerShell - Amazon Athena cmdlet reference, the AWS Tools for PowerShell portal page, and the AWS Tools for Windows PowerShell User Guide.
- For information about Athena service endpoints that you can connect to programmatically, see Amazon Athena endpoints and quotas in the *Amazon Web Services General Reference*.

Set up, administrative, and programmatic access

If you've already signed up for Amazon Web Services, you can start using Amazon Athena immediately. If you haven't signed up for AWS or need assistance getting started, be sure to complete the following tasks.

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*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentia ls with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in

Which user needs programmatic access?	To	By
		the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Attach managed policies for Athena

Athena managed policies grant permissions to use Athena features. You can attach these managed policies to one or more IAM roles that users can assume in order to use Athena.

An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

For more information about roles, see IAM roles and Creating IAM roles in the *IAM User Guide*.

To create a role that grants access to Athena, you attach Athena managed policies to the role. There are two managed policies for Athena: `AmazonAthenaFullAccess` and `AWSQuicksightAthenaAccess`. These policies grant permissions to Athena to query Amazon S3 and to write the results of your queries to a separate bucket on your behalf. To see the contents of these policies for Athena, see AWS managed policies for Amazon Athena.

For steps to attach the Athena managed policies to a role, follow Adding IAM identity permissions (console) in the *IAM User Guide* and add the `AmazonAthenaFullAccess` and `AWSQuicksightAthenaAccess` managed policies to the role that you created.

> **Note**
>
> You may need additional permissions to access the underlying dataset in Amazon S3. If you are not the account owner or otherwise have restricted access to a bucket, contact the bucket owner to grant access using a resource-based bucket policy, or contact your account administrator to grant access using a role-based policy. For more information, see Control access to Amazon S3 from Athena. If the dataset or Athena query results are encrypted, you may need additional permissions. For more information, see Encryption at rest.

AWS service integrations with Athena

You can use Athena to query data from the AWS services listed in this section. To see the Regions that each service supports, see Regions and endpoints in the *Amazon Web Services General Reference*.

AWS services integrated with Athena

- AWS CloudFormation
- Amazon CloudFront
- AWS CloudTrail
- Amazon DataZone
- Elastic Load Balancing
- Amazon EMR Studio
- AWS Glue Data Catalog
- AWS Identity and Access Management (IAM)
- Amazon QuickSight
- Amazon S3 Inventory
- AWS Step Functions
- AWS Systems Manager Inventory
- Amazon Virtual Private Cloud

For information about each integration, see the following sections.

AWS CloudFormation

Capacity reservation

Reference topic: [AWS::Athena::CapacityReservation] in the *AWS CloudFormation User Guide*

Specifies a capacity reservation with the provided name and number of requested data processing units. For more information, see Manage query processing capacity in the *Amazon Athena User Guide* and CreateCapacityReservation in the *Amazon Athena API Reference*.

Data catalog

Reference topic: AWS::Athena::DataCatalog in the *AWS CloudFormation User Guide*

Specify an Athena data catalog, including a name, description, type, parameters, and tags. For more information, see Understanding tables, databases, and data catalogs in Athena in the *Amazon Athena User Guide* and CreateDataCatalog in the *Amazon Athena API Reference*.

Named query

Reference topic: AWS::Athena::NamedQuery in the *AWS CloudFormation User Guide*

Specify named queries with AWS CloudFormation and run them in Athena. Named queries allow you to map a query name to a query and then run it as a saved query from the Athena console. For more information, see Use saved queries in the *Amazon Athena User Guide* and CreateNamedQuery in the *Amazon Athena API Reference*.

Prepared statement

Reference topic: AWS::Athena::PreparedStatement in the *AWS CloudFormation User Guide*

Specifies a prepared statement for use with SQL queries in Athena. A prepared statement contains parameter placeholders whose values are supplied at execution time. For more information, see Use parameterized queries in the *Amazon Athena User Guide* and CreatePreparedStatement in the *Amazon Athena API Reference*.

Workgroup

Reference topic: AWS::Athena::WorkGroup in the *AWS CloudFormation User Guide*

Specify Athena workgroups using AWS CloudFormation. Use Athena workgroups to isolate queries for you or your group from other queries in the same account. For more information, see Use workgroups to control query access and costs in the *Amazon Athena User Guide* and CreateWorkGroup in the *Amazon Athena API Reference*.

Amazon CloudFront

Reference topic: Query Amazon CloudFront logs

Use Athena to query Amazon CloudFront logs. For more information about using CloudFront, see the Amazon CloudFront Developer Guide.

AWS CloudTrail

Reference topic: Query AWS CloudTrail logs

Using Athena with CloudTrail logs is a powerful way to enhance your analysis of AWS service activity. For example, you can use queries to identify trends and further isolate activity by attribute, such as source IP address or user. You can create tables for querying logs directly from the CloudTrail console, and use those tables to run queries in Athena. For more information, see Use the CloudTrail console to create an Athena table for CloudTrail logs .

Amazon DataZone

Reference topic: Use Amazon DataZone in Athena

Use Amazon DataZone to share, search, and discover data at scale across organizational boundaries. DataZone simplifies your experience across AWS analytics services like Athena, AWS Glue, and AWS Lake Formation. If you have large amounts of data in different data sources, you can use Amazon DataZone to build business use case based groupings of people, data and tools.

In Athena, you can use the query editor to access and query DataZone environments. For more information, see Use Amazon DataZone in Athena.

Elastic Load Balancing

Reference topic: Query Application Load Balancer logs

Querying Application Load Balancer logs allows you to see the source of traffic, latency, and bytes transferred to and from Elastic Load Balancing instances and backend applications. For more information, see Query Application Load Balancer logs.

Reference topic: Query Classic Load Balancer logs

Query Classic Load Balancer logs to analyze and understand traffic patterns to and from Elastic Load Balancing instances and backend applications. You can see the source of traffic, latency, and bytes transferred. For more information, see Query Classic Load Balancer logs.

Amazon EMR Studio

Reference topic: Use the Amazon Athena SQL editor in EMR Studio

You can use Athena in an EMR Studio to develop and run interactive queries. This makes it possible for you to use EMR Studio for SQL analytics on Athena from the same Amazon EMR interface that you use for your Spark, Scala, and other workloads. With the Athena integration in EMR Studio, you can perform the following tasks:

- Perform Athena SQL queries
- View query results
- View query history
- View saved queries
- Perform parameterized queries
- View databases, tables, and views for a data catalog

The following Athena features are not available in Amazon EMR Studio:

- Admin features like creating or updating Athena workgroups, data sources, or capacity reservations
- Athena for Spark or Spark notebooks
- DataZone integration
- Step Functions

EMR Studio integration with Athena is available in all AWS Regions where EMR Studio and Athena are available. For more information about using Athena in EMR Studio, see Use the Amazon Athena SQL editor in EMR Studio in the *Amazon EMR Management Guide*.

AWS Glue Data Catalog

Reference topic: Use AWS Glue Data Catalog to connect to your data

Athena integrates with the AWS Glue Data Catalog, which offers a persistent metadata store for your data in Amazon S3. This allows you to create tables and query data in Athena based on a central metadata store available throughout your Amazon Web Services account and integrated with the ETL and data discovery features of AWS Glue. For more information, see Use AWS Glue Data Catalog to connect to your data and What is AWS Glue in the *AWS Glue Developer Guide*.

AWS Identity and Access Management (IAM)

Reference topic: Actions for Amazon Athena

You can use Athena API actions in IAM permission policies. For more information, see Actions for Amazon Athena and Identity and access management in Athena.

Amazon QuickSight

Reference topic: Connect to Amazon Athena with ODBC and JDBC drivers

Athena integrates with Amazon QuickSight for easy data visualization. You can use Athena to generate reports or to explore data with business intelligence tools or SQL clients connected with a JDBC or an ODBC driver. For more information about QuickSight, see What is Amazon QuickSight in the *Amazon QuickSight User Guide*. For information about using JDBC and ODBC drivers with Athena, see Connecting to Amazon Athena with ODBC and JDBC Drivers.

Amazon S3 Inventory

Reference topic: Querying inventory with Athena in the *Amazon Simple Storage Service User Guide*

You can use Amazon Athena to query Amazon S3 inventory using standard SQL. You can use Amazon S3 inventory to audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs. For more information, see Amazon S3 inventory in the *Amazon Simple Storage Service User Guide*.

AWS Step Functions

Reference topic: Call Athena with Step Functions in the *AWS Step Functions Developer Guide*

Call Athena with AWS Step Functions. AWS Step Functions can control select AWS services directly using the Amazon States Language. You can use Step Functions with Athena to start and stop query execution, get query results, run ad-hoc or scheduled data queries, and retrieve results from data lakes in Amazon S3. The Step Functions role must have permissions to use Athena. For more information, see the AWS Step Functions Developer Guide.

Video: Orchestrate Amazon Athena Queries using AWS Step Functions

The following video demonstrates how to use Amazon Athena and AWS Step Functions to run a regularly scheduled Athena query and generate a corresponding report.

Orchestrate Amazon Athena queries using AWS Step Functions

For an example that uses Step Functions and Amazon EventBridge to orchestrate AWS Glue DataBrew, Athena, and Amazon QuickSight, see Orchestrating an AWS Glue DataBrew job and Amazon Athena query with AWS Step Functions in the AWS Big Data Blog.

AWS Systems Manager Inventory

Reference topic: Querying inventory data from multiple regions and accounts in the *AWS Systems Manager User Guide*

AWS Systems Manager Inventory integrates with Amazon Athena to help you query inventory data from multiple AWS Regions and accounts. For more information, see the AWS Systems Manager User Guide.

Amazon Virtual Private Cloud

Reference topic: Query Amazon VPC flow logs

Amazon Virtual Private Cloud flow logs capture information about the IP traffic going to and from network interfaces in a VPC. Query the logs in Athena to investigate network traffic patterns and identify threats and risks across your Amazon VPC network. For more information about Amazon VPC, see the Amazon VPC User Guide.

Use Athena SQL

You can use Athena SQL to query your data in-place in Amazon S3 using the AWS Glue Data Catalog, an external Hive metastore, or federated queries using a variety of prebuilt connectors to other data sources.

You can also:

- Connect to business intelligence tools and other applications using Athena's JDBC and ODBC drivers.
- Query AWS service logs.
- Query Apache Iceberg tables, including time travel queries, and Apache Hudi datasets.
- Query geospatial data.
- Query using machine learning inference from Amazon SageMaker AI.
- Query using your own user-defined functions.
- Speed up query processing of highly-partitioned tables and automate partition management by using partition projection.

Topics

- SQL reference for Athena
- Troubleshoot issues in Athena
- Code samples

Understanding tables, databases, and data catalogs in Athena

In Athena, catalogs, databases, and tables are containers for the metadata definitions that define a schema for underlying source data.

Athena uses the following terms to refer to hierarchies of data objects:

- **Data source** – a group of databases
- **Database** – a group of tables
- **Table** – data organized as a group of rows or columns

Sometimes these objects are also referred to with alternate but equivalent names such as the following:

- A data source is sometimes referred to as a *catalog*.
- A database is sometimes referred to as a *schema*.

> **Note**
>
> This terminology can vary in the federated data sources that you use with Athena. For more information, see Understand federated table name qualifiers.

For each dataset, a table needs to exist in Athena. The metadata in the table tells Athena where the data is located in Amazon S3, and specifies the structure of the data, for example, column names, data types, and the name of the table. Databases are a logical grouping of tables, and also hold only metadata and schema information for a dataset.

For each dataset that you'd like to query, Athena must have an underlying table it will use for obtaining and returning query results. Therefore, before querying data, a table must be registered in Athena. The registration occurs when you either create tables automatically or manually.

You can create a table automatically using an AWS Glue crawler. For more information about AWS Glue and crawlers, see Use AWS Glue Data Catalog to connect to your data. When AWS Glue creates a table, it registers it in its own AWS Glue Data Catalog. Athena uses the AWS Glue Data Catalog to store and retrieve this metadata, using it when you run queries to analyze the underlying dataset.

Regardless of how the tables are created, the table creation process registers the dataset with Athena. This registration occurs in the AWS Glue Data Catalog and enables Athena to run queries on the data. In the Athena query editor, this catalog (or data source) is referred to with the label `AwsDataCatalog`.

After you create a table, you can use SQL SELECT statements to query it, including getting specific file locations for your source data. Your query results are stored in Amazon S3 in the query result location that you specify.

The AWS Glue Data Catalog is accessible throughout your Amazon Web Services account. Other AWS services can share the AWS Glue Data Catalog, so you can see databases and tables created throughout your organization using Athena and vice versa.

- To create a table manually:
  - Use the Athena console to run the **Create Table Wizard**.
  - Use the Athena console to write Hive DDL statements in the Query Editor.
  - Use the Athena API or CLI to run a SQL query string with DDL statements.
  - Use the Athena JDBC or ODBC driver.

When you create tables and databases manually, Athena uses HiveQL data definition language (DDL) statements such as `CREATE TABLE`, `CREATE DATABASE`, and `DROP TABLE` under the hood to create tables and databases in the AWS Glue Data Catalog.

To get started, you can use a tutorial in the Athena console or work through a step-by-step guide in the Athena documentation.

- To use the tutorial in the Athena console, choose the information icon on the upper right of the console, and then choose the **Tutorial** tab.
- For a step-by-step tutorial on creating a table and writing queries in the Athena query editor, see Get started.

Get started

This tutorial walks you through using Amazon Athena to query data. You'll create a table based on sample data stored in Amazon Simple Storage Service, query the table, and check the results of the query.

The tutorial uses live resources, so you are charged for the queries that you run. You aren't charged for the sample data in the location that this tutorial uses, but if you upload your own data files to Amazon S3, charges do apply.

Prerequisites

- If you have not already done so, sign up for an AWS account.
- Using the same AWS Region (for example, US West (Oregon)) and account that you are using for Athena, follow the steps to create a bucket in Amazon S3 to hold your Athena query results. You will configure this bucket to be your query output location.

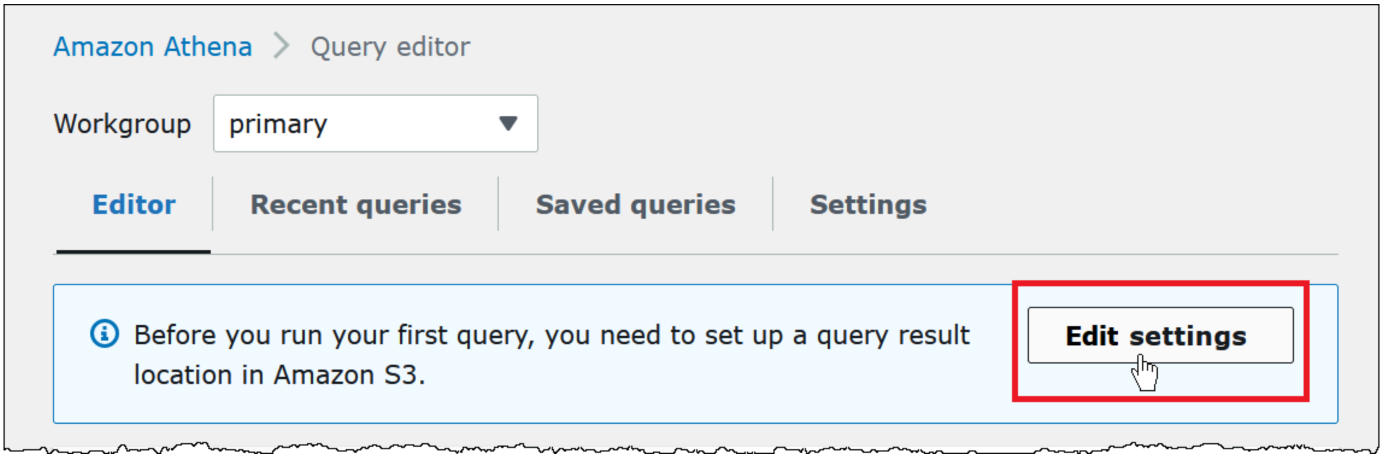
Topics

Step 1: Create a database

You first need to create a database in Athena.

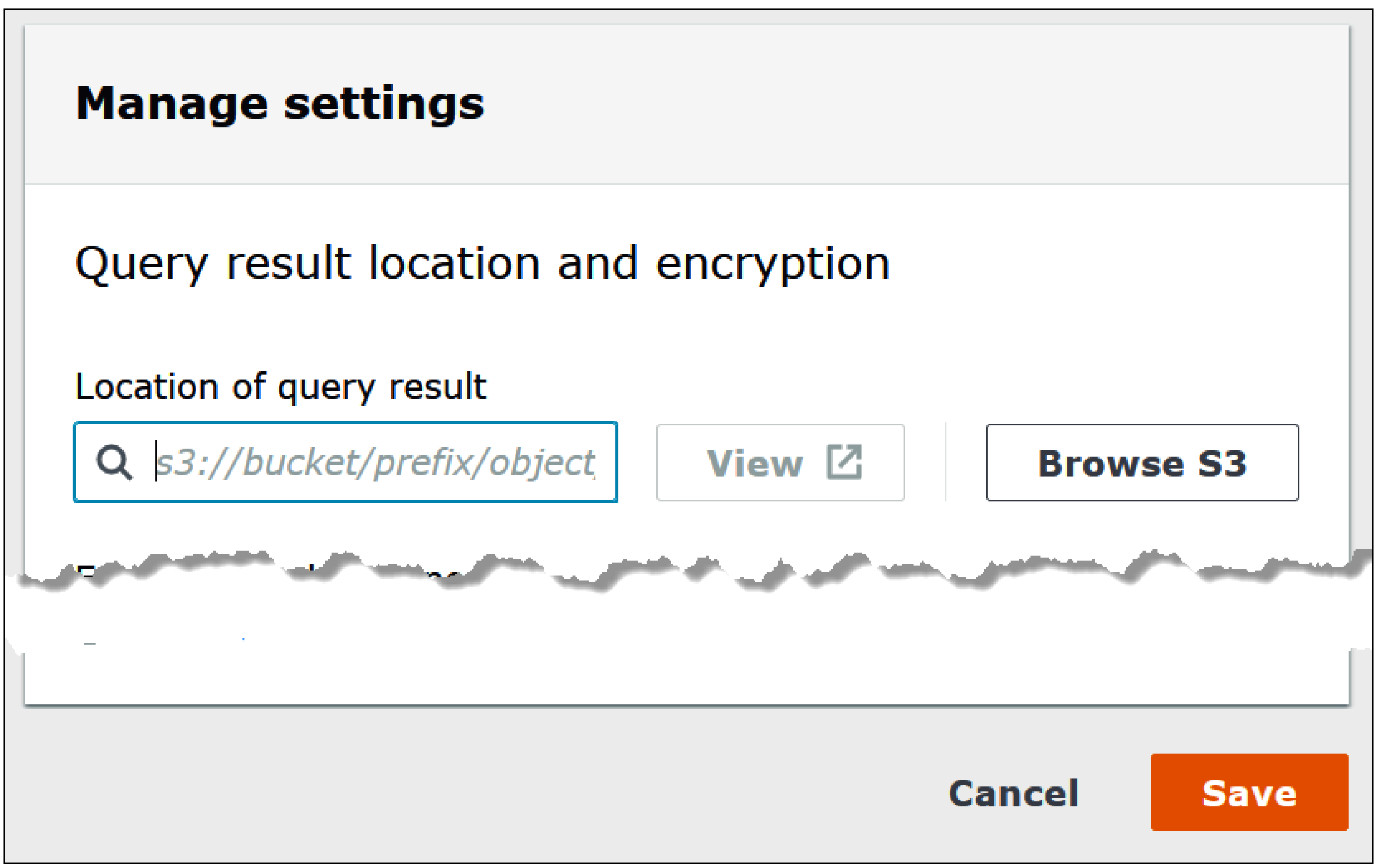
To create an Athena database

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If this is your first time to visit the Athena console in your current AWS Region, choose **Explore the query editor** to open the query editor. Otherwise, Athena opens in the query editor.
3. Choose **Edit Settings** to set up a query result location in Amazon S3.



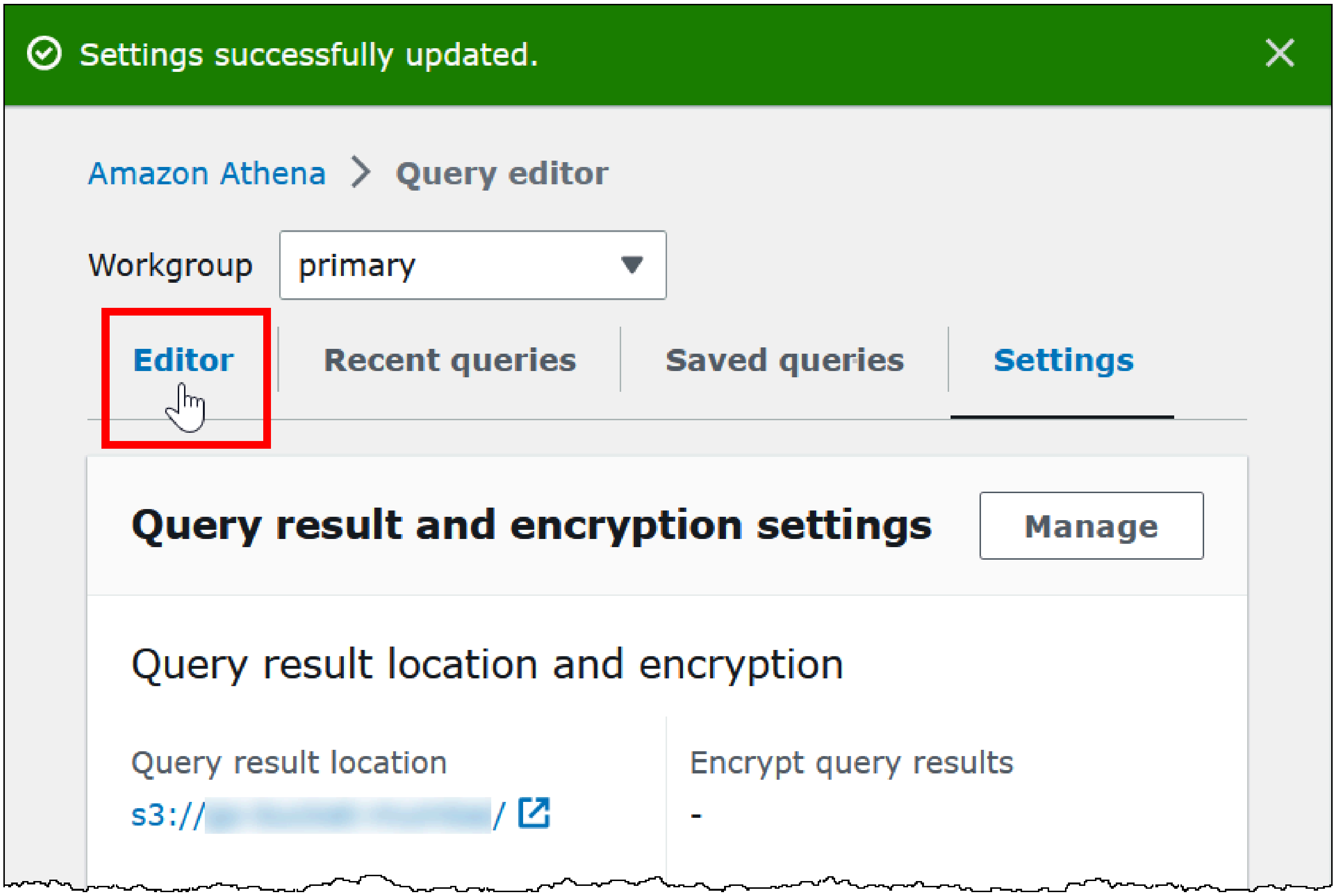
4. For **Manage settings**, do one of the following:

   - In the **Location of query result** box, enter the path to the bucket that you created in Amazon S3 for your query results. Prefix the path with `s3://`.
   - Choose **Browse S3**, choose the Amazon S3 bucket that you created for your current Region, and then choose **Choose**.



5. Choose **Save**.

6. Choose **Editor** to switch to the query editor.



7. On the right of the navigation pane, you can use the Athena query editor to enter and run queries and statements.
8. To create a database named `mydatabase`, enter the following CREATE DATABASE statement.

```
CREATE DATABASE mydatabase
```

9. Choose **Run** or press **`Ctrl+ENTER`**.
10. From the **Database** list on the left, choose `mydatabase` to make it your current database.

Step 2: Create a table

Now that you have a database, you can create an Athena table for it. The table that you create will be based on sample Amazon CloudFront log data in the location `s3://athena-examples-`*`myregion`*`/cloudfront/plaintext/`, where *`myregion`* is your current AWS Region.

The sample log data is in tab-separated values (TSV) format, which means that a tab character is used as a delimiter to separate the fields. The data looks like the following example. For readability, the tabs in the excerpt have been converted to spaces and the final field shortened.

```
2014-07-05 20:00:09 DFW3 4260 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-1.jpeg 200 - Mozilla/5.0[...]
2014-07-05 20:00:09 DFW3 4252 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-2.jpeg 200 - Mozilla/5.0[...]
2014-07-05 20:00:10 AMS1 4261 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-3.jpeg 200 - Mozilla/5.0[...]
```

To enable Athena to read this data, you could create a straightforward `CREATE EXTERNAL TABLE` statement like the following. The statement that creates the table defines columns that map to the data, specifies how the data is delimited, and specifies the Amazon S3 location that contains the sample data. Note that because Athena expects to scan all of the files in a folder, the `LOCATION` clause specifies an Amazon S3 folder location, not a specific file.

Do not use this example just yet as it has an important limitation that will be explained shortly.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  ClientInfo STRING
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LINES TERMINATED BY '\n'
  LOCATION 's3://athena-examples-my-region/cloudfront/plaintext/';
```

The example creates a table called `cloudfront_logs` and specifies a name and data type for each field. These fields become the columns in the table. Because `date` is a [reserved word](), it is escaped with backtick (`) characters. `ROW FORMAT DELIMITED` means that Athena will use a default library called [LazySimpleSerDe]() to do the actual work of parsing the data. The example also specifies that the fields are tab separated (`FIELDS TERMINATED BY '\t'`) and that each record in the file ends in a newline character (`LINES TERMINATED BY '\n`). Finally, the `LOCATION` clause specifies the path in Amazon S3 where the actual data to be read is located.

If you have your own tab or comma-separated data, you can use a `CREATE TABLE` statement like the example just presented—as long as your fields do not contain nested information. However, if you have a column like `ClientInfo` that contains nested information that uses a different delimiter, a different approach is required.

Extracting data from the ClientInfo field

Looking at the sample data, here is a full example of the final field `ClientInfo`:

```
Mozilla/5.0%20(Android;%20U;%20Windows%20NT%205.1;%20en-US;
%20rv:1.9.0.9)%20Gecko/2009040821%20IE/3.0.9
```

As you can see, this field is multivalued. Because the example `CREATE TABLE` statement just presented specifies tabs as field delimiters, it can't break out the separate components inside the `ClientInfo` field into separate columns. So, a new `CREATE TABLE` statement is required.

To create columns from the values inside the `ClientInfo` field, you can use a regular expression (regex) that contains regex groups. The regex groups that you specify become separate table columns. To use a regex in your `CREATE TABLE` statement, use syntax like the following. This syntax instructs Athena to use the Regex SerDe library and the regular expression that you specify.

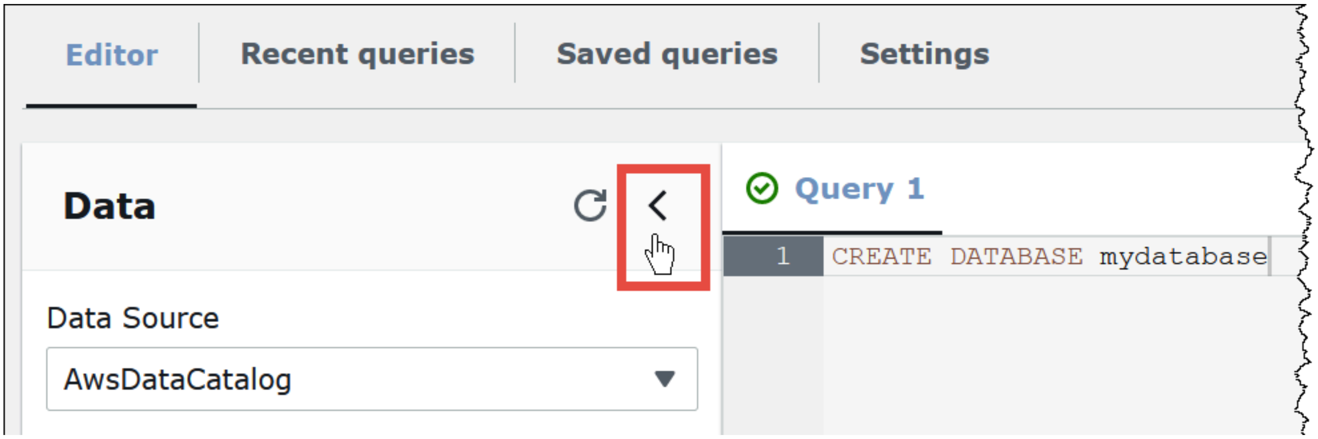
```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES ("input.regex" = "regular_expression")
```

Regular expressions can be useful for creating tables from complex CSV or TSV data but can be difficult to write and maintain. Fortunately, there are other libraries that you can use for formats like JSON, Parquet, and ORC. For more information, see Choose a SerDe for your data.

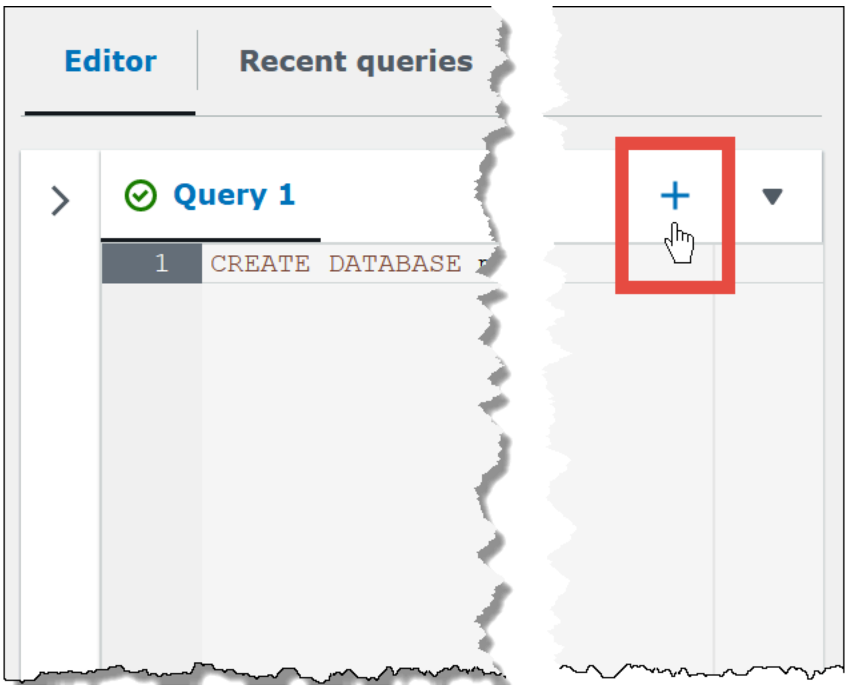
Now you are ready to create the table in the Athena query editor. The `CREATE TABLE` statement and regex are provided for you.

To create a table in Athena

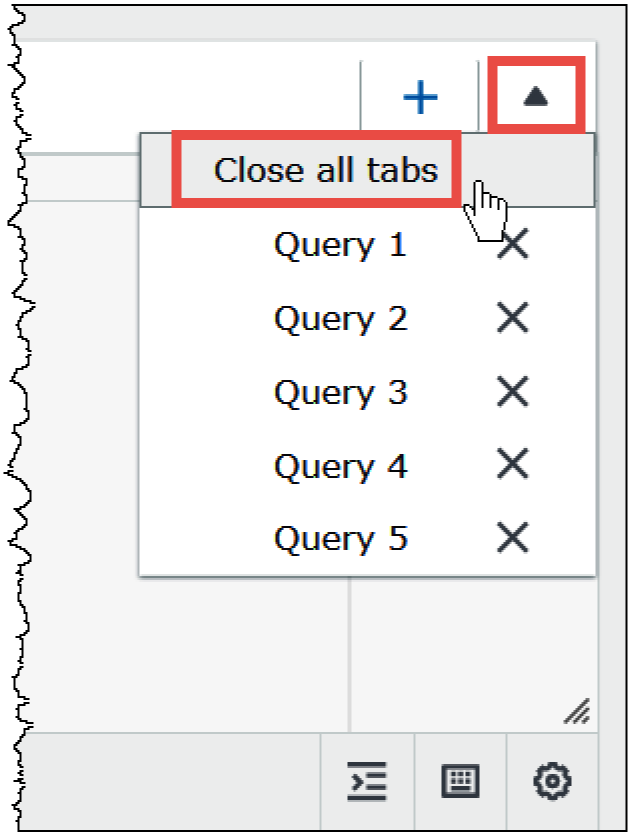
1. In the navigation pane, for **Database**, make sure that `mydatabase` is selected.
2. To give yourself more room in the query editor, you can choose the arrow icon to collapse the navigation pane.



3. To create a tab for a new query, choose the plus (**+**) sign in the query editor. You can have up to ten query tabs open at once.



4. To close one or more query tabs, choose the arrow next to the plus sign. To close all tabs at once, choose the arrow, and then choose **Close all tabs**.



5. In the query pane, enter the following `CREATE EXTERNAL TABLE` statement. The regex breaks out the operating system, browser, and browser version information from the `ClientInfo` field in the log data.

> **Note**
>
> The regex used in the following example is designed to work with the publicly available sample CloudFront log data in the `athena-examples` Amazon S3 location and is illustrative only. For more up-to-date regexes that query both standard and real-time CloudFront log files, see [Query Amazon CloudFront logs](#).

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
```

```
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
  BrowserVersion STRING
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES (
  "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s
+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^
\/]+)[\/](.*)$"
  ) LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

6. In the `LOCATION` statement, replace *`myregion`* with the AWS Region that you are currently using (for example, `us-west-1`).

7. Choose **Run**.

   The table `cloudfront_logs` is created and appears under the list of **Tables** for the `mydatabase` database.

Step 3: Query data

Now that you have the `cloudfront_logs` table created in Athena based on the data in Amazon S3, you can run SQL queries on the table and see the results in Athena. For more information about using SQL in Athena, see SQL reference for Athena.

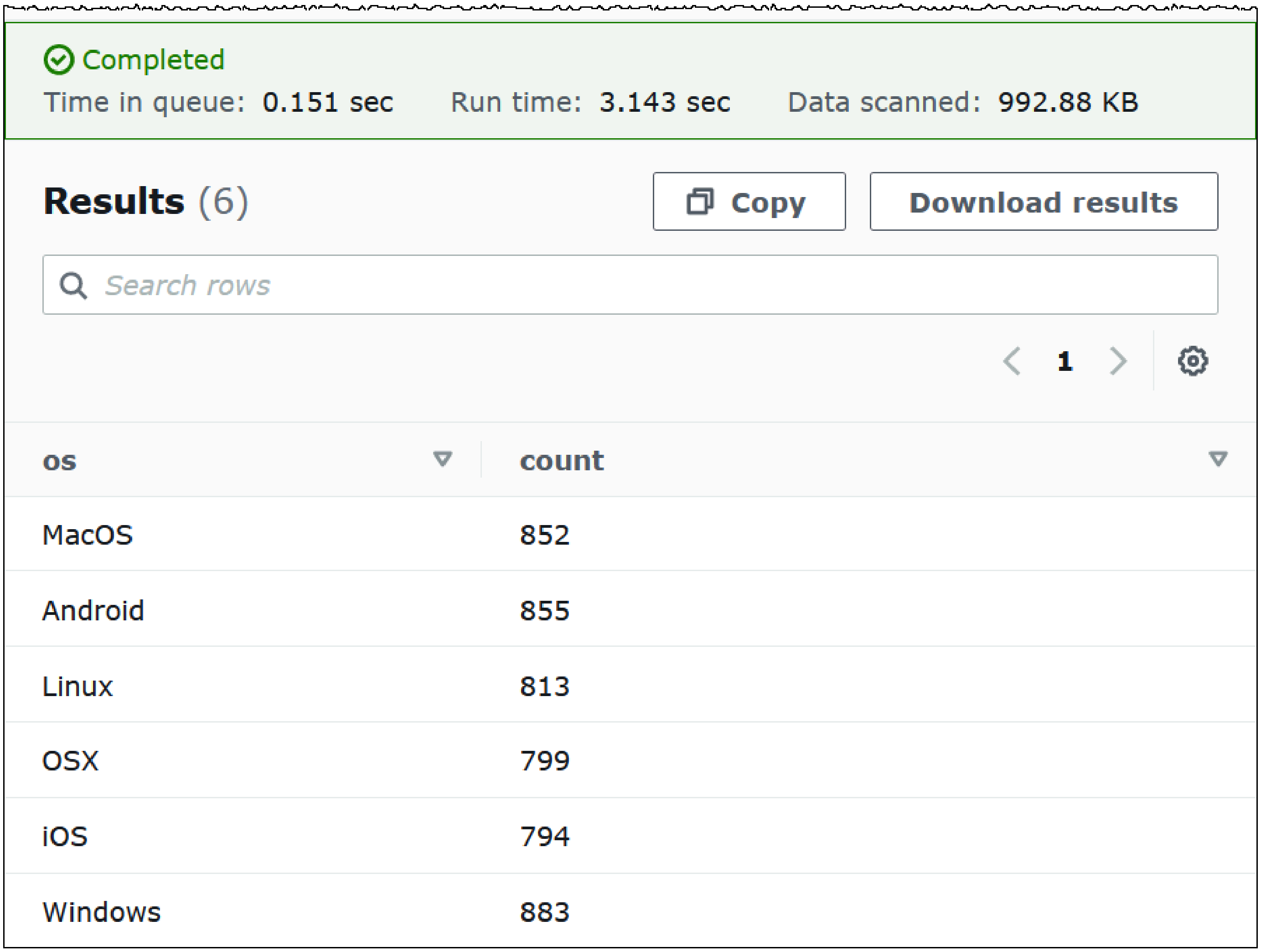
To run a query

1. Choose the plus (**+**) sign to open a new query tab and enter the following SQL statement in the query pane.

```
SELECT os, COUNT(*) count
FROM cloudfront_logs
WHERE date BETWEEN date '2014-07-05' AND date '2014-08-05'
GROUP BY os
```

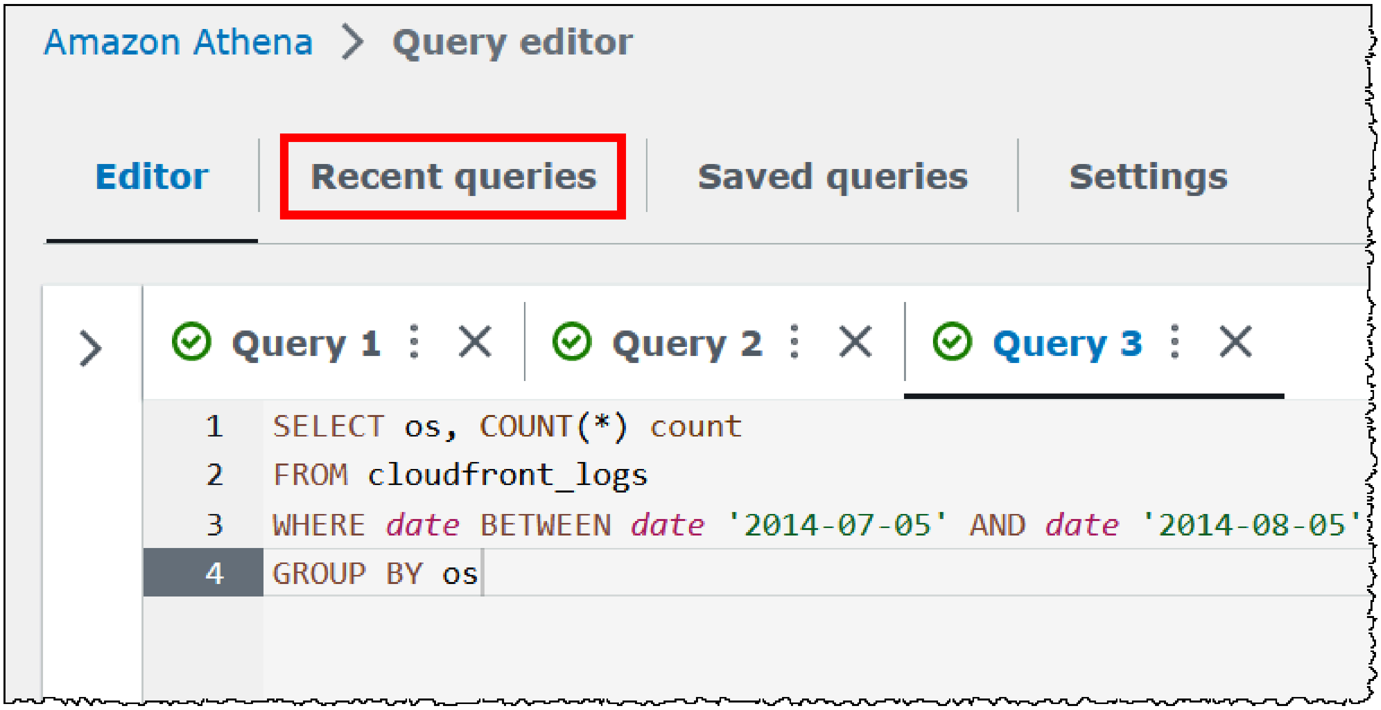
2. Choose **Run**.

   The results look like the following:

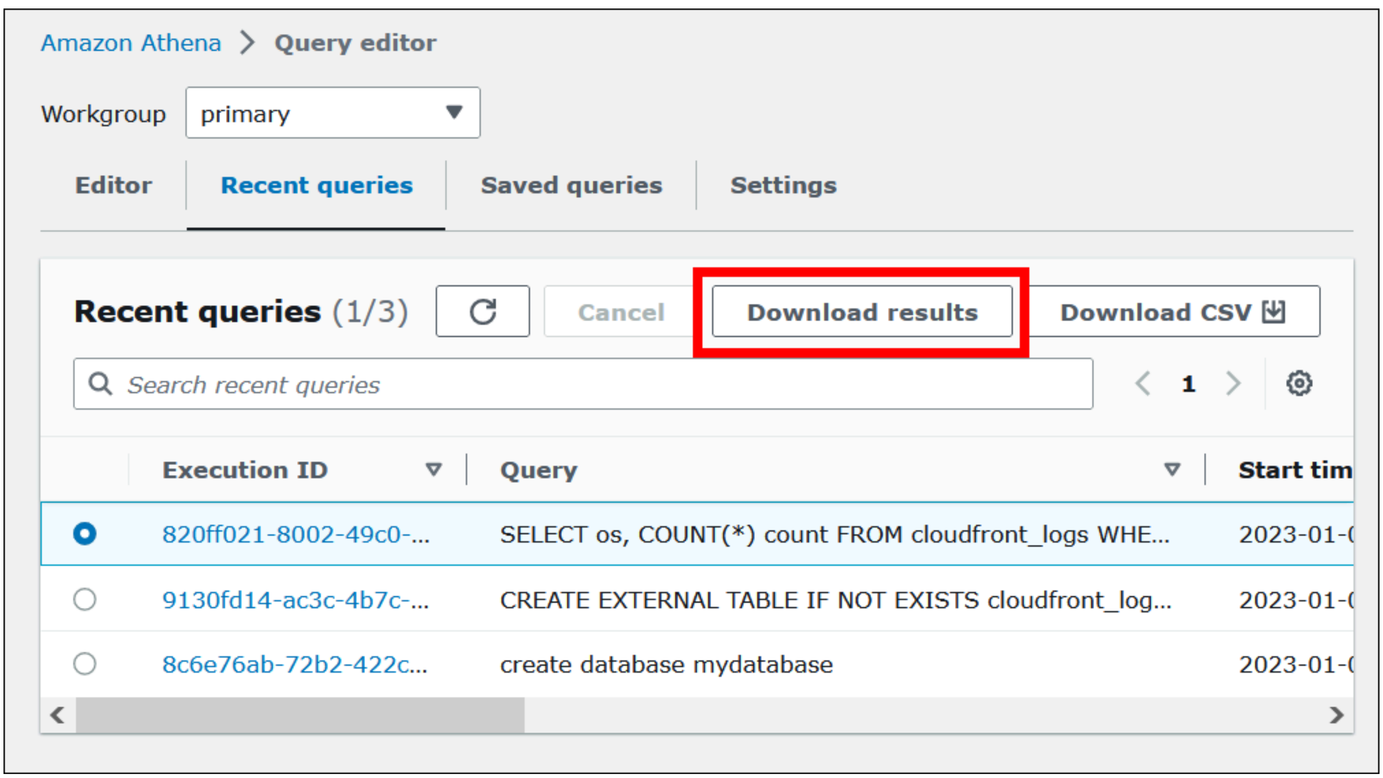


3. To save the results of the query to a `.csv` file, choose **Download results**.

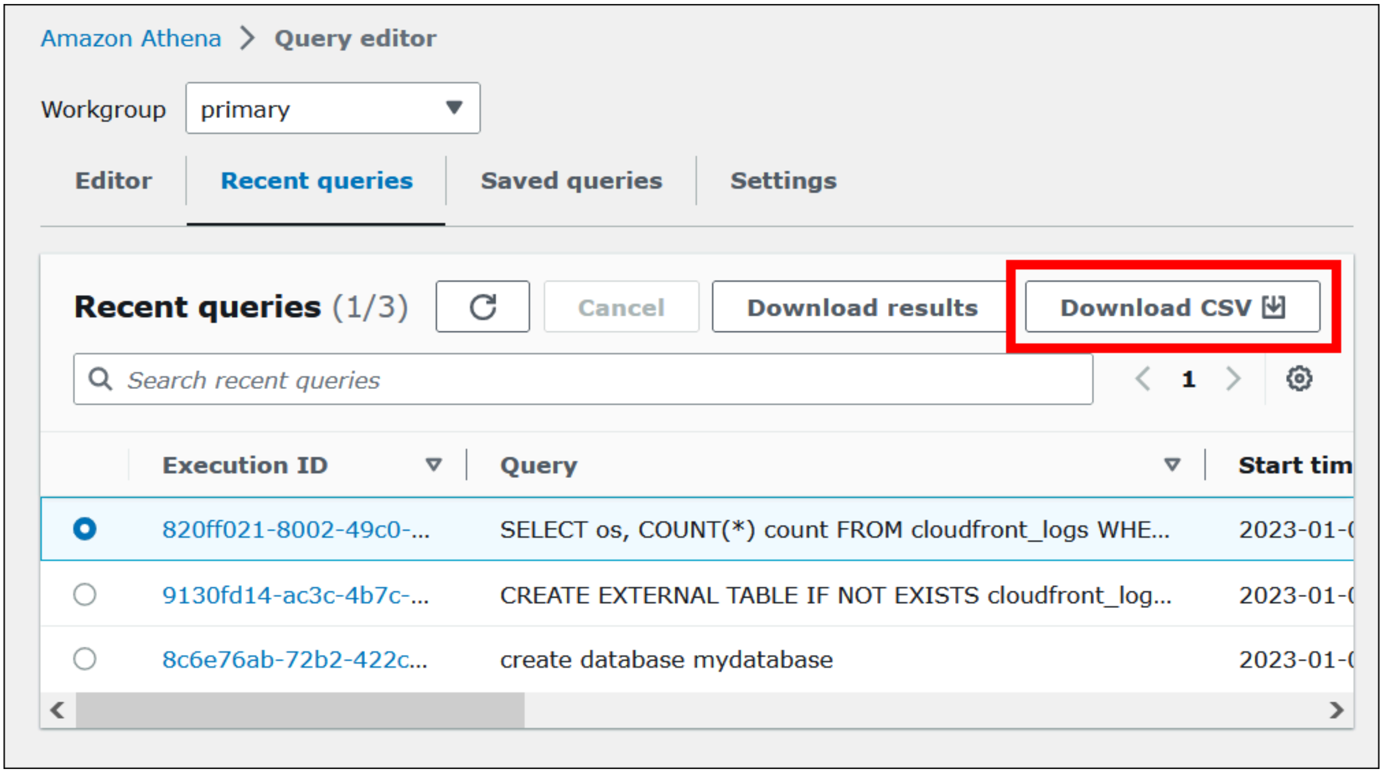
4. To view or run previous queries, choose the **Recent queries** tab.



5. To download the results of a previous query from the **Recent queries** tab, select the query, and then choose **Download results**. Queries are retained for 45 days.



6. To download one or more recent SQL query strings to a CSV file, choose **Download CSV**.



For more information, see Work with query results and recent queries.

Step 4: Use named queries

You can save the queries that you create or edit in the query editor with a name. Athena stores these queries on the **Saved queries** tab. You can use the **Saved queries** tab to recall, run, rename, or delete your saved queries. For more information, see Use saved queries.

Step 5: Use keyboard shortcuts and typeahead suggestions

The Athena query editor provides numerous keyboard shortcuts for actions like running a query, formatting a query, line operations, and find and replace. For more information and a complete list of shortcuts, see Improve productivity by using keyboard shortcuts in Amazon Athena query editor in the *AWS Big Data Blog*.

The Athena query editor supports typeahead code suggestions for a faster query authoring experience. To help you write SQL queries with enhanced accuracy and increased efficiency, it offers the following features:

- As you type, suggestions appear in real time for keywords, local variables, snippets, and catalog items.

- When you type a database name or table name followed by a dot, the editor conveniently displays a list of tables or columns to choose from.
- When you hover over a snippet suggestion, a synopsis shows a brief overview of the snippet's syntax and usage.
- To improve code readability, keywords and their highlighting rules have also been updated to align with latest syntax of Trino and Hive.

This feature is enabled by default. To enable or disable the feature, use the **Code editor preferences** (gear icon) at the bottom right of the query editor window.

Step 6: Connect to other data sources

This tutorial used a data source in Amazon S3 in CSV format. For information about using Athena with AWS Glue, see Use AWS Glue Data Catalog to connect to your data. You can also connect Athena to a variety of data sources by using ODBC and JDBC drivers, external Hive metastores, and Athena data source connectors. For more information, see Connect to data sources.

Connect to data sources

You can use Amazon Athena to query data stored in different locations and formats in a *dataset*. This dataset might be in CSV, JSON, Avro, Parquet, or some other format.

The tables and databases that you work with in Athena to run queries are based on *metadata*. Metadata is data about the underlying data in your dataset. How that metadata describes your dataset is called the *schema*. For example, a table name, the column names in the table, and the data type of each column are schema, saved as metadata, that describe an underlying dataset. In Athena, we call a system for organizing metadata a *data catalog* or a *metastore*. The combination of a dataset and the data catalog that describes it is called a *data source*.

The relationship of metadata to an underlying dataset depends on the type of data source that you work with. Relational data sources like MySQL, PostgreSQL, and SQL Server tightly integrate the metadata with the dataset. In these systems, the metadata is most often written when the data is written. Other data sources, like those built using Hive, allow you to define metadata on-the-fly when you read the dataset. The dataset can be in a variety of formats—for example, CSV, JSON, Parquet, or Avro.

Athena natively supports the AWS Glue Data Catalog. The AWS Glue Data Catalog is a data catalog built on top of other datasets and data sources such as Amazon S3, Amazon Redshift, and Amazon DynamoDB. You can also connect Athena to other data sources by using a variety of connectors.

Topics

Use AWS Glue Data Catalog to connect to your data

Athena uses the AWS Glue Data Catalog to store metadata such as table and column names for your data stored in Amazon S3. This metadata information becomes the databases, tables, and views that you see in the Athena query editor.

When using Athena with the AWS Glue Data Catalog, you can use AWS Glue to create databases and tables (schema) to be queried in Athena, or you can use Athena to create schema and then use them in AWS Glue and related services.

To define schema information for AWS Glue, you can use a form in the Athena console, use the query editor in Athena, or create an AWS Glue crawler in the AWS Glue console. AWS Glue crawlers automatically infer database and table schema from your data in Amazon S3. Using a form offers more customization. Writing your own `CREATE TABLE` statements requires more effort, but offers the most control. For more information, see CREATE TABLE.

Additional Resources

- For more information about the AWS Glue Data Catalog, see Data Catalog and crawlers in AWS Glue in the *AWS Glue Developer Guide*.
- For an illustrative article showing how to use AWS Glue and Athena to process XML data, see Process and analyze highly nested and large XML files using AWS Glue and Amazon Athena in the AWS Big Data Blog.
- Separate charges apply to AWS Glue. For more information, see AWS Glue pricing.

Topics

Register and use data catalogs in Athena

Athena supports mounting and connecting to multiple data catalogs.

- You can mount Amazon Redshift data in the AWS Glue Data Catalog and query it from Athena without having to copy or move data. For more information, see Bringing Amazon Redshift data into the AWS Glue Data Catalog.
- Connect the AWS Glue Data Catalog to external data sources using AWS Glue connections, and create federated catalogs to centrally manage permissions to the data with fine-grained access control using Lake Formation. For more information, see Register your connection as a Glue Data Catalog.
- Create catalogs from Amazon S3 table buckets, and use Lake Formation to centrally manage access permissions and restrict user access to objects within the table bucket. For more information, see Working with Amazon S3 Tables and table buckets in the Amazon S3 User Guide.

> **Note**
>
> For any Glue catalog, you can only register a multi-level catalog like `123412341234:my_catalog/my_child`. You cannot register a single-level catalog like `123412341234:linkcontainer` or `my_catalog`. Single-level catalogs can only be

queried by using the Glue data catalog directly in the Athena query. For more information, see Query AWS Glue data catalogs in Athena.

Topics

Register Redshift data catalogs in Athena

Athena can read and write data stored in Redshift clusters or serverless namespaces that have been registered in the AWS Glue Data Catalog. This works in tandem with AWS Lake Formation, which provides centralized security and governance, ensuring that data access is managed consistently across different query engines and maintaining fine-grained access controls for the shared Redshift data.

Considerations and limitations

- **Materialized views** – Amazon Redshift materialized views are queryable from Athena but creating materialized views using Athena or Spark is not supported.
- DDL operations, including setting AWS Glue Data Catalog configuration and operations on Amazon Redshift managed storage tables, are not supported.

Prerequisites

Before you can query a AWS Glue data catalog from Athena, complete the following tasks:

1. Create and register an Amazon Redshift cluster or serverless namespace to the AWS Glue Data Catalog. For more information, see Registering a cluster to the AWS Glue Data Catalog or Registering namespaces to the AWS Glue Data Catalog in the Amazon Redshift Management guide.
2. Create a data catalog in AWS Lake Formation from the registered namespace. For more information, see Creating Amazon Redshift federated catalogs in the AWS Lake Formation Developer Guide.

3. (Optional) Use Lake Formation to set fine-grained access controls on the catalog. For more information, see [Bringing your data into the AWS Glue Data Catalog] in the AWS Lake Formation Developer Guide.

Register a Redshift data catalog with the Athena console

To register a Redshift data catalog with the Athena console, perform the following steps.

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Data sources and catalogs**.
3. On the **Data sources and catalogs** page, choose **Create data source**.
4. For **Choose a data source**, choose **Amazon S3 - AWS Glue Data Catalog**.
5. In the **AWS Glue Data Catalog** section, for **Data source account**, choose **AWS Glue Data Catalog in this account**.
6. For **Create a table or register a catalog**, choose **Register a new AWS Glue Catalog**.
7. In the **Data source details** section, for **Data source name**, enter the name that you want to use to specify the data source in your SQL queries or use the default name that is generated.
8. For **Catalog**, choose **Browse** to search for a list of AWS Glue catalogs in the same account. If you don't see any existing catalogs, create one in AWS Glue console.
9. In the **Browse AWS Glue catalogs** dialog box, select the catalog that you want to use, and then choose **Choose**.
10. (Optional) For **Tags**, enter any key/value pairs that you want to associate with the data source.
11. Choose **Next**.
12. On the **Review and create** page, verify that the information that you entered is correct, and then choose **Create data source**.

Register federated catalogs in Athena

After you create connections to federated data sources, you can register them as federated data catalogs for simplified data discovery and manage data access with fine-grained permissions using Lake Formation. For more information, see Register your connection as a Glue Data Catalog.

Considerations and limitations

- DDL operations are not supported on federated catalogs.

- You can register the following connectors to integrate with AWS Glue for fine-grained access control:
  - Redshift
  - DynamoDB
  - BigQuery
  - Snowflake
  - MySQL
  - PostgreSQL
  - Timestream
  - CMDB
- When you create a resource link for Glue connection federation, the name of resource link must be same as the database name of the producer.
- Currently, only lowercase table and column names are recognized even if the data source is case insensitive.

Register S3 table bucket catalogs and query Tables from Athena

Amazon S3 table buckets are a bucket type in Amazon S3 that is purpose-built to store tabular data in Apache Iceberg tables. Table buckets automate table management tasks such as compaction, snapshot management, and garbage collection to continuously optimize query performance and minimize cost. Whether you're just starting out, or have thousands of tables in your Iceberg environment, table buckets simplify data lakes at any scale. For more information, see Table buckets.

Considerations and limitations

- All DDL operations supported for Iceberg tables are supported for S3 Tables with the following exceptions:
  - `ALTER TABLE RENAME`, `CREATE VIEW`, and `ALTER DATABASE` are not supported.
  - `CREATE TABLE AS SELECT` (CTAS) – You can still perform a `CREATE TABLE` DDL and then run `INSERT INTO <s3_table> SELECT * FROM source_table` to seed an S3 Table from an existing table.
  - `OPTIMIZE` and `VACUUM` – You can manage compaction and snapshot management in S3. For more information, see S3 Tables maintenance documentation.
- DDL queries on S3 Tables registered as Athena data sources are not supported.

- Query result reuse is not supported.
- Querying Iceberg table metadata is not supported.
- In workgroups with SSE-KMS, CSE-KMS encryption enabled, you can't run write operations like `INSERT`, `UPDATE`, `DELETE`, or `MERGE` on S3 Tables.
- In workgroups with S3 Requester Pays option enabled, you can't run DML operations on S3 Tables.

Query S3 Tables from Athena

Complete these prerequisite steps before you query S3 Tables in Athena

1. Create an S3 table bucket. For more information, see Creating a table bucket in Amazon Simple Storage Service User Guide.
2. Make sure that the integration of your table buckets with AWS Glue Data Catalog and AWS Lake Formation is successful by following Prerequisites for integration and Integrating table buckets with AWS analytics services in Amazon Simple Storage Service User Guide.

   > **Note**
   >
   > If you enabled the integration while creating an S3 table bucket from the S3 console in Step 1, then you can skip this step.

3. For the principal you use to run queries with Athena, grant Lake Formation permissions on the S3 Table catalog, either through the Lake Formation console or AWS CLI.

   AWS Management Console

   1. Open the AWS Lake Formation console at https://console.aws.amazon.com/lakeformation/ and sign in as a data lake administrator. For more information on how to create a data lake administrator, see Create a data lake administrator.
   2. In the navigation pane, choose **Data permissions** and then choose **Grant**.
   3. On the **Grant Permissions** page, under **Principals**, choose the principal that you want to use to submit query from Athena.
   4. Under **LF-Tags or catalog resources**, choose **Named Data Catalog resources**.
   5. For **Catalogs**, choose a glue data catalog that you created from the integration of your table bucket. For example, *<accoundID>*:s3tablescatalog/*amzn-s3-demo-bucket*.

6. For **Catalog permissions**, choose **Super**.
7. Choose **Grant**.

AWS CLI

Run the following command with Lake Formation data lake administrator role to grant access to the principal that you use to submit query from Athena.

```
aws lakeformation grant-permissions \
--region <region (Example,us-east-1)> \
--cli-input-json \
'{
    "Principal": {
        "DataLakePrincipalIdentifier": "<user or role ARN (Example,
 arn:aws:iam::<Account ID>:role/ExampleRole>"
    },
    "Resource": {
        "Catalog": {
            "Id":"<Account ID>:s3tablescatalog/amzn-s3-demo-bucket"
        }
    },
    "Permissions": ["ALL"]
}'
```

Submit queries for S3 Tables

1. Submit a `CREATE DATABASE` query from Athena with the above granted user/role. In this example, `s3tablescatalog` is the parent Glue Data Catalog created from the integration and `s3tablescatalog/amzn-s3-demo-bucket` is the child Glue Data Catalog created for each S3 table bucket. There are two ways in which you can query.

   Option 1

   Specify the child Glue Data Catalog (`s3tablescatalog/amzn-s3-demo-bucket`) directly from console or AWS CLI.

   Using AWS Management Console

   1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. In the left navigation, for **Data source name**, choose **AwsDataCatalog**.
3. For **Catalog**, choose **s3tablescatalog/*amzn-s3-demo-bucket***.
4. In the query editor, enter a query like `CREATE DATABASE test_namespace`.

Using AWS CLI

Run the following command.

```
aws athena start-query-execution \
--query-string 'CREATE DATABASE `test_namespace`' \
--query-execution-context '{"Catalog": "s3tablescatalog/amzn-s3-demo-bucket"}' \
--work-group "primary"
```

Option 2

Create Athena data catalog from the child Glue Data Catalog in the Athena console and specify it as a catalog in the query. For more information, see Register S3 table bucket catalogs as Athena data sources.

2. With the database that you created in previous step, use `CREATE TABLE` to create a table. The following example creates a table in the *`test_namespace`* database that you previously created in the `s3tablescatalog/amzn-s3-demo-bucket` Glue catalog.

AWS Management Console

1. In the left navigation, for **Data source name**, choose **AwsDataCatalog**.
2. For **Catalog**, choose **s3tablescatalog/*amzn-s3-demo-bucket***.
3. For **Database**, choose **test_namespace**.
4. In the query editor, run the following query.

```
CREATE TABLE daily_sales (
        sale_date date,
        product_category
        string, sales_amount double)
PARTITIONED BY (month(sale_date))
TBLPROPERTIES ('table_type' = 'iceberg')
```

AWS CLI

Run the following command.

```
aws athena start-query-execution \
--query-string "CREATE TABLE daily_sales (
        sale_date date,
        product_category
        string, sales_amount double)
PARTITIONED BY (month(sale_date))
TBLPROPERTIES ('table_type' = 'iceberg')" \
--query-execution-context '{"Catalog": "s3tablescatalog/amzn-s3-demo-bucket",
 "Database":"test_namespace"}' \
--work-group "primary"
```

3. Insert data into the table that you created in the previous step.

AWS Management Console

1. In the left navigation, for **Data source name**, choose **AwsDataCatalog**.
2. For **Catalog**, choose **s3tablescatalog/*amzn-s3-demo-bucket***.
3. For **Database**, choose **test_namespace**.
4. In the query editor, run the following query.

```
INSERT INTO daily_sales
VALUES
    (DATE '2024-01-15', 'Laptop', 900.00),
    (DATE '2024-01-15', 'Monitor', 250.00),
    (DATE '2024-01-16', 'Laptop', 1350.00),
    (DATE '2024-02-01', 'Monitor', 300.00);
```

AWS CLI

Run the following command.

```
aws athena start-query-execution \
--query-string "INSERT INTO \"s3tablescatalog/amzn-s3-demo-
bucket\".test_namespace.daily_sales
VALUES
```

```
(DATE '2024-01-15', 'Laptop', 900.00),
(DATE '2024-01-15', 'Monitor', 250.00),
(DATE '2024-01-16', 'Laptop', 1350.00),
(DATE '2024-02-01', 'Monitor', 300.00)"\
--work-group "primary"
```

4. After inserting data into the table, you can query it.

   AWS Management Console

   1. In the left navigation, for **Data source name**, choose **AwsDataCatalog**.
   2. For **Catalog**, choose **s3tablescatalog/*amzn-s3-demo-bucket***.
   3. For **Database**, choose **test_namespace**.
   4. In the query editor, run the following query.

   ```
   SELECT
       product_category,
       COUNT(*) AS units_sold,
       SUM(sales_amount) AS total_revenue,
       AVG(sales_amount) AS average_price
   FROM
       daily_sales
   WHERE
       sale_date BETWEEN DATE '2024-02-01'
                     AND DATE '2024-02-29'
   GROUP BY
       product_category
   ORDER BY
       total_revenue DESC
   ```

   AWS CLI

   Run the following command.

   ```
   aws athena start-query-execution \
   --query-string "SELECT product_category,
       COUNT(*) AS units_sold,
       SUM(sales_amount) AS total_revenue,
       AVG(sales_amount) AS average_price
   FROM \"s3tablescatalog/amzn-s3-demo-bucket\".test_namespace.daily_sales
   WHERE sale_date BETWEEN DATE '2024-02-01' AND DATE '2024-02-29'
   ```

```
GROUP BY product_category
ORDER BY total_revenue DESC"\
--work-group "primary"
```

Create S3 Tables in Athena

Athena supports creating tables in existing S3 Table namespaces or namespaces created in Athena with `CREATE DATABASE` statements. To create an S3 Table from Athena, the syntax is the same as when you create a regular Iceberg table except you don't specify the `LOCATION`, as shown in the following example.

```
CREATE TABLE
[db_name.]table_name (col_name data_type [COMMENT col_comment] [, ...] )
[PARTITIONED BY (col_name | transform, ... )]
[TBLPROPERTIES ([, property_name=property_value] )]
```

Note

`TBLPROPERTIES` is optional and you are not required to set the table type as `Iceberg` when you create a table in a S3 Table namespace.

Register S3 table bucket catalogs as Athena data sources

To register S3 table bucket catalogs with the Athena console, perform the following steps.

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Data sources and catalogs**.
3. On the **Data sources and catalogs** page, choose **Create data source**.
4. For **Choose a data source**, choose **Amazon S3 - AWS Glue Data Catalog**.
5. In the **AWS Glue Data Catalog** section, for **Data source account**, choose **AWS Glue Data Catalog in this account**.
6. For **Create a table or register a catalog**, choose **Register a new AWS Glue Catalog**.
7. In the **Data source details** section, for **Data source name**, enter the name that you want to use to specify the data source in your SQL queries or use the default name that is generated.
8. For **Catalog**, choose **Browse** to search for a list of AWS Glue catalogs in the same account. If you don't see any existing catalogs, create one in AWS Glue console.

9. In the **Browse AWS Glue catalogs** dialog box, select the catalog that you want to use, and then choose **Choose**.
10. (Optional) For **Tags**, enter any key/value pairs that you want to associate with the data source.
11. Choose **Next**.
12. On the **Review and create** page, verify that the information that you entered is correct, and then choose **Create data source**.

Query AWS Glue data catalogs in Athena

To query data catalogs from Athena, do one of the following.

- Register the catalog in Athena as a data source, then use the data source name to query the catalog. In this usage, the following queries are equivalent.

```
SELECT * FROM my_data_source.my_database.my_table
```

- If you are querying a catalog that has not been registered as an Athena data source, you can supply the full path to the catalog in your `SELECT` queries, as in the following example.

```
SELECT * FROM "my_catalog/my_subcatalog".my_database.my_table
```

- You can also do this through the AWS Management Console.
  1. Open the Athena console at https://console.aws.amazon.com/athena/
  2. In the query editor, for **Data source**, choose **AwsDataCatalog**.
  3. For **Catalog**, choose the name of the catalog that you want to use.
  4. For **Database**, choose the database that contains the table that you want to query.
  5. Enter a query like `SELECT * FROM` *`my_table`*, and then choose **Run**.

Register a Data Catalog from another account

You can use Athena's cross-account AWS Glue catalog feature to register an AWS Glue catalog from an account other than your own. After you configure the required IAM permissions for AWS Glue and register the catalog as an Athena `DataCatalog` resource, you can use Athena to run cross-account queries. For information about configuring the required permissions, see [Configure cross-account access to AWS Glue data catalogs](#).

The following procedure shows you how to use the Athena to configure an AWS Glue Data Catalog in an Amazon Web Services account other than your own as a data source.

Register from console

1. Follow the steps in [Configure cross-account access to AWS Glue data catalogs]() to ensure that you have permissions to query the data catalog in the other account.
2. Open the Athena console at https://console.aws.amazon.com/athena/.
3. If the console navigation pane is not visible, choose the expansion menu on the left.

4. Choose **Data sources and catalogs**.
5. On the upper right, choose **Create data source**.
6. On the **Choose a data source** page, for **Data sources**, choose **S3 - AWS Glue Data Catalog**, and then choose **Next**.
7. On the **Enter data source details** page, in the **AWS Glue Data Catalog** section, for **Choose an AWS Glue Data Catalog**, choose **AWS Glue Data Catalog in another account**.
8. For **Data source details**, enter the following information:

   - **Data source name** – Enter the name that you want to use in your SQL queries to refer to the data catalog in the other account.
   - **Description** – (Optional) Enter a description of the data catalog in the other account.
   - **Catalog ID** – Enter the 12-digit Amazon Web Services account ID of the account to which the data catalog belongs. The Amazon Web Services account ID is the catalog ID.

9. (Optional) For **Tags**, enter key-value pairs that you want to associate with the data source. For more information about tags, see [Tag Athena resources](#).
10. Choose **Next**.
11. On the **Review and create** page, review the information that you provided, and then choose **Create data source**. The **Data source details** page lists the databases and tags for the data catalog that you registered.
12. Choose **Data sources and catalogs**. The data catalog that you registered is listed in the **Data source name** column.
13. To view or edit information about the data catalog, choose the catalog, and then choose **Actions**, **Edit**.
14. To delete the new data catalog, choose the catalog, and then choose **Actions**, **Delete**.

Register using API operations

1. The following `CreateDataCatalog` request body registers an AWS Glue catalog for cross-account access:

```
# Example CreateDataCatalog request to register a cross-account Glue catalog:
{
    "Description": "Cross-account Glue catalog",
    "Name": "ownerCatalog",
    "Parameters": {"catalog-id" : "<catalogid>"  # Owner's account ID
    },
    "Type": "GLUE"
}
```

2. The following sample code uses a Java client to create the `DataCatalog` object.

```
# Sample code to create the DataCatalog through Java client
CreateDataCatalogRequest request = new CreateDataCatalogRequest()
    .withName("ownerCatalog")
    .withType(DataCatalogType.GLUE)
    .withParameters(ImmutableMap.of("catalog-id", "<catalogid>"));

athenaClient.createDataCatalog(request);
```

After these steps, the borrower should see *`ownerCatalog`* when it calls the [ListDataCatalogs](#) API operation.

Register using AWS CLI

Use the followig example CLI command to register an AWS Glue Data Catalog from another account

```
aws athena create-data-catalog \
  --name cross_account_catalog \
  --type GLUE \
  --description "Cross Account Catalog" \
  --parameters catalog-id=<catalogid>
```

For more information, see Query cross-account AWS Glue Data Catalogs using Amazon Athena in the *AWS Big Data Blog*.

Control access to data catalogs with IAM policies

To control access to data catalogs, use resource-level IAM permissions or identity-based IAM policies.

The following procedure is specific to Athena.

For IAM-specific information, see the links listed at the end of this section. For information about example JSON data catalog policies, see Data Catalog example policies.

To use the visual editor in the IAM console to create a data catalog policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**, and then choose **Create policy**.
3. On the **Visual editor** tab, choose **Choose a service**. Then choose Athena to add to the policy.
4. Choose **Select actions**, and then choose the actions to add to the policy. The visual editor shows the actions available in Athena. For more information, see Actions, resources, and condition keys for Amazon Athena in the *Service Authorization Reference*.
5. Choose **add actions** to type a specific action or use wildcards (*) to specify multiple actions.

   By default, the policy that you are creating allows the actions that you choose. If you chose one or more actions that support resource-level permissions to the `datacatalog` resource in Athena, then the editor lists the `datacatalog` resource.
6. Choose **Resources** to specify the specific data catalogs for your policy. For example JSON data catalog policies, see Data Catalog example policies.

7. Specify the `datacatalog` resource as follows:

```
arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>
```

8. Choose **Review policy**, and then type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy summary to make sure that you granted the intended permissions.
9. Choose **Create policy** to save your new policy.
10. Attach this identity-based policy to a user, a group, or role and specify the `datacatalog` resources they can access.

For more information, see the following topics in the *Service Authorization Reference* and the *IAM User Guide*:

- Actions, resources, and condition keys for Amazon Athena
- Creating policies with the visual editor
- Adding and removing IAM policies
- Controlling access to resources

For example JSON data catalog policies, see Data Catalog example policies.

For information about AWS Glue permissions and AWS Glue crawler permissions, see Setting up IAM permissions for AWS Glue and Crawler prerequisites in the *AWS Glue Developer Guide*.

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference.

Data Catalog example policies

This section includes example policies you can use to enable various actions on data catalogs.

A data catalog is an IAM resource managed by Athena. Therefore, if your data catalog policy uses actions that take `datacatalog` as an input, you must specify the data catalog's ARN as follows:

```
"Resource": [arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>]
```

The `<datacatalog-name>` is the name of your data catalog. For example, for a data catalog named `test_datacatalog`, specify it as a resource as follows:

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:datacatalog/test_datacatalog"]
```

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference. For more information about IAM policies, see Creating policies with the visual editor in the *IAM User Guide*. For more information about creating IAM policies for workgroups, see Control access to data catalogs with IAM policies.

- Example Policy for Full Access to All Data Catalogs
- Example Policy for Full Access to a Specified Data Catalog
- Example Policy for Querying a Specified Data Catalog
- Example Policy for Management Operations on a Specified Data Catalog
- Example Policy for Listing Data Catalogs
- Example Policy for Metadata Operations on Data Catalogs

Example Example policy for full access to all data catalogs

The following policy allows full access to all data catalog resources that might exist in the account. We recommend that you use this policy for those users in your account that must administer and manage data catalogs for all other users.

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

Example Example policy for full access to a specified Data Catalog

The following policy allows full access to the single specific data catalog resource, named `datacatalogA`. You could use this policy for users with full control over a particular data catalog.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:ListDataCatalogs",
            "athena:ListWorkGroups",
            "athena:GetDatabase",
            "athena:ListDatabases",
            "athena:ListTableMetadata",
            "athena:GetTableMetadata"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution",
            "athena:GetQueryResults",
            "athena:DeleteNamedQuery",
            "athena:GetNamedQuery",
            "athena:ListQueryExecutions",
            "athena:StopQueryExecution",
            "athena:GetQueryResultsStream",
            "athena:ListNamedQueries",
            "athena:CreateNamedQuery",
            "athena:GetQueryExecution",
            "athena:BatchGetNamedQuery",
            "athena:BatchGetQueryExecution",
            "athena:DeleteWorkGroup",
            "athena:UpdateWorkGroup",
            "athena:GetWorkGroup",
            "athena:CreateWorkGroup"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
      },
      {
        "Effect":"Allow",
        "Action":[
            "athena:CreateDataCatalog",
```

```
                "athena:DeleteDataCatalog",
                "athena:GetDataCatalog",
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDatabases",
                "athena:ListTableMetadata",
                "athena:UpdateDataCatalog"
            ],
            "Resource":"arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
        }
    ]
}
```

Example Example policy for querying a specified Data Catalog

In the following policy, a user is allowed to run queries on the specified `datacatalogA`. The user is not allowed to perform management tasks for the data catalog itself, such as updating or deleting it.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:GetDataCatalog"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
         ]
      }
   ]
}
```

Example Example policy for management operations on a specified Data Catalog

In the following policy, a user is allowed to create, delete, obtain details, and update a data catalog `datacatalogA`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:UpdateDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
            ]
        }
    ]
}
```

Example Example policy for listing data catalogs

The following policy allows all users to list all data catalogs:

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

Example Example policy for metadata operations on data catalogs

The following policy allows metadata operations on data catalogs:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": "*"
        }
    ]
}
```

Use a form in the Athena console to add an AWS Glue table

The following procedure shows you how to use the Athena console to add a table using the **Create Table From S3 bucket data** form.

To add a table and enter schema information using a form

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the query editor, next to **Tables and views**, choose **Create**, and then choose **S3 bucket data.**
3. On the **Create Table From S3 bucket data** form, for **Table name**, enter a name for the table. For information about acceptable characters for database names, table names, and column names in Athena, see Name databases, tables, and columns.
4. For **Database configuration**, choose an existing database, or create a new one.
5. For **Location of Input Data Set**, specify the path in Amazon S3 to the folder that contains the dataset that you want to process. Do not include a file name in the path. Athena scans all files in the folder that you specify. If your data is already partitioned (for example,

   s3://amzn-s3-demo-bucket/logs/year=2004/month=12/day=11/), enter the base path only (for example, s3://amzn-s3-demo-bucket/logs/).
6. For **Data Format**, choose among the following options:

   - For **Table type**, choose **Apache Hive**, **Apache Iceberg**, or **Delta Lake**. Athena uses the Apache Hive table type as the default. For information about querying Apache Iceberg tables in Athena, see Query Apache Iceberg tables. For information about using Delta Lake tables in Athena, see Query Linux Foundation Delta Lake tables.

- For **File format**, choose the file or log format that your data is in.
  - For the **Text File with Custom Delimiters** option, specify a **Field terminator** (that is, a column delimiter). Optionally, you can specify a **Collection terminator** that marks the end of an array type or a **Collection terminator** that marks the end of a map data type.
- **SerDe library** – A SerDe (serializer-deserializer) library parses a particular data format so that Athena can create a table for it. For most formats, a default SerDe library is chosen for you. For the following formats, choose a library according to your requirements:
  - **Apache Web Logs** – Choose either the **RegexSerDe** or **GrokSerDe** library. For RegexSerDe, provide a regular expression in the **Regex definition** box. For GrokSerDe, provide a series of named regular expressions for the `input.format` SerDe property. Named regular expressions are easier to read and maintain than regular expressions. For more information, see Query Apache logs stored in Amazon S3.
  - **CSV** – Choose **LazySimpleSerDe** if your comma-separated data does not contain values enclosed in double quotes or if it uses the `java.sql.Timestamp` format. Choose **OpenCSVSerDe** if your data includes quotes or uses the UNIX numeric format for `TIMESTAMP` (for example, `1564610311`). For more information, see Lazy Simple SerDe for CSV, TSV, and custom-delimited files and Open CSV SerDe for processing CSV.
  - **JSON** – Choose either the **OpenX** or **Hive** JSON SerDe library. Both formats expect each JSON document to be on a single line of text and that fields not be separated by newline characters. The OpenX SerDe offers some additional properties. For more information about these properties, see OpenX JSON SerDe. For information about the Hive SerDe, see Hive JSON SerDe.

  For more information about using SerDe libraries in Athena, see Choose a SerDe for your data.

7. For **SerDe properties**, add, edit, or remove properties and values according to the SerDe library that you are using and your requirements.
   - To add a SerDe property, choose **Add SerDe property**.
   - In the **Name** field, enter the name of the property.
   - In the **Value** field, enter a value for the property.
   - To remove a SerDe property, choose **Remove**.
8. For **Table properties**, choose or edit the table properties according to your requirements.

- For **Write compression**, choose a compression option. The availability of the write compression option and of the compression options available depends on the data format. For more information, see Use compression in Athena.
- For **Encryption**, select **Encrypted data set** if the underlying data is encrypted in Amazon S3. This option sets the `has_encrypted_data` table property to true in the `CREATE TABLE` statement.

9. For **Column details**, enter the names and data types of the columns that you want to add to the table.

   - To add more columns one at a time, choose **Add a column**.
   - To quickly add more columns, choose **Bulk add columns**. In the text box, enter a comma separated list of columns in the format *`column_name data_type`*, *`column_name data_type`*[, …], and then choose **Add**.

10. (Optional) For **Partition details**, add one or more column names and data types. Partitioning keeps related data together based on column values and can help reduce the amount of data scanned per query. For information about partitioning, see Partition your data.
11. (Optional) For **Bucketing**, you can specify one or more columns that have rows that you want to group together, and then put those rows into multiple buckets. This allows you to query only the bucket that you want to read when the bucketed columns value is specified.

    - For **Buckets**, select one or more columns that have a large number of unique values (for example, a primary key) and that are frequently used to filter the data in your queries.
    - For **Number of buckets**, enter a number that permits files to be of optimal size. For more information, see Top 10 Performance Tuning Tips for Amazon Athena in the AWS Big Data Blog.
    - To specify your bucketed columns, the `CREATE TABLE` statement will use the following syntax:

```
CLUSTERED BY (bucketed_columns) INTO number_of_buckets BUCKETS
```

> **Note**
>
> The **Bucketing** option is not available for the **Iceberg** table type.

12. The **Preview table query** box shows the `CREATE TABLE` statement generated by the information that you entered into the form. The preview statement cannot be edited directly. To change the statement, modify the form fields above the preview, or create the statement directly in the query editor instead of using the form.

13. Choose **Create table** to run the generated statement in the query editor and create the table.

Use a crawler to add a table

AWS Glue crawlers help discover the schema for datasets and register them as tables in the AWS Glue Data Catalog. The crawlers go through your data and determine the schema. In addition, the crawler can detect and register partitions. For more information, see Defining crawlers in the *AWS Glue Developer Guide*. Tables from data that were successfully crawled can be queried from Athena.

> **Note**
>
> Athena does not recognize exclude patterns that you specify for an AWS Glue crawler. For example, if you have an Amazon S3 bucket that contains both `.csv` and `.json` files and you exclude the `.json` files from the crawler, Athena queries both groups of files. To avoid this, place the files that you want to exclude in a different location.

Create an AWS Glue crawler

You can create a crawler by starting in the Athena console and then using the AWS Glue console in an integrated way. When you create the crawler, you specify a data location in Amazon S3 to crawl.

To create a crawler in AWS Glue starting from the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. In the query editor, next to **Tables and views**, choose **Create**, and then choose **AWS Glue crawler**.

3. On the **AWS Glue** console **Add crawler** page, follow the steps to create a crawler. For more information, see Using AWS Glue Crawlers in this guide and Populating the AWS Glue Data Catalog in the *AWS Glue Developer Guide*.

> **Note**
>
> Athena does not recognize exclude patterns that you specify for an AWS Glue crawler. For example, if you have an Amazon S3 bucket that contains both `.csv` and `.json` files and you exclude the `.json` files from the crawler, Athena queries both groups of files. To avoid this, place the files that you want to exclude in a different location.

After a crawl, the AWS Glue crawler automatically assigns certain table metadata to help make it compatible with other external technologies like Apache Hive, Presto, and Spark. Occasionally, the crawler may incorrectly assign metadata properties. Manually correct the properties in AWS Glue before querying the table using Athena. For more information, see Viewing and editing table details in the *AWS Glue Developer Guide*.

AWS Glue may mis-assign metadata when a CSV file has quotes around each data field, getting the `serializationLib` property wrong. For more information, see Handling CSV data enclosed in quotes.

Use multiple data sources with a crawler

When an AWS Glue crawler scans Amazon S3 and detects multiple directories, it uses a heuristic to determine where the root for a table is in the directory structure, and which directories are partitions for the table. In some cases, where the schema detected in two or more directories is similar, the crawler may treat them as partitions instead of separate tables. One way to help the crawler discover individual tables is to add each table's root directory as a data store for the crawler.

The following partitions in Amazon S3 are an example:

```
s3://amzn-s3-demo-bucket/folder1/table1/partition1/file.txt
s3://amzn-s3-demo-bucket/folder1/table1/partition2/file.txt
s3://amzn-s3-demo-bucket/folder1/table1/partition3/file.txt
s3://amzn-s3-demo-bucket/folder1/table2/partition4/file.txt
s3://amzn-s3-demo-bucket/folder1/table2/partition5/file.txt
```

If the schema for `table1` and `table2` are similar, and a single data source is set to `s3://amzn-s3-demo-bucket/folder1/` in AWS Glue, the crawler may create a single table with two partition columns: one partition column that contains `table1` and `table2`, and a second partition column that contains `partition1` through `partition5`.

To have the AWS Glue crawler create two separate tables, set the crawler to have two data sources, `s3://amzn-s3-demo-bucket/folder1/table1/` and `s3://amzn-s3-demo-bucket/folder1/table2`, as shown in the following procedure.

To add an S3 data store to an existing crawler in AWS Glue

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the navigation pane, choose **Crawlers**.
3. Choose the link to your crawler, and then choose **Edit**.
4. For **Step 2: Choose data sources and classifiers**, choose **Edit**.
5. For **Data sources and catalogs**, choose **Add a data source**.
6. In the **Add data source** dialog box, for **S3 path**, choose **Browse**.
7. Select the bucket that you want to use, and then choose **Choose**.

   The data source that you added appears in the **Data sources** list.
8. Choose **Next**.
9. On the **Configure security settings** page, create or choose an IAM role for the crawler, and then choose **Next**.
10. Make sure that the S3 path ends in a trailing slash, and then choose **Add an S3 data source**.
11. On the **Set output and scheduling** page, for **Output configuration**, choose the target database.
12. Choose **Next**.
13. On the **Review and update** page, review the choices that you made. To edit a step, choose **Edit**.
14. Choose **Update**.

Schedule a crawler to keep the AWS Glue Data Catalog and Amazon S3 in sync

AWS Glue crawlers can be set up to run on a schedule or on demand. For more information, see [Time-based schedules for jobs and crawlers](#) in the *AWS Glue Developer Guide*.

If you have data that arrives for a partitioned table at a fixed time, you can set up an AWS Glue crawler to run on schedule to detect and update table partitions. This can eliminate the need to run a potentially long and expensive `MSCK REPAIR` command or manually run an `ALTER TABLE`

ADD PARTITION command. For more information, see Table partitions in the *AWS Glue Developer Guide*.

Optimize queries with AWS Glue partition indexing and filtering

When Athena queries partitioned tables, it retrieves and filters the available table partitions to the subset relevant to your query. As new data and partitions are added, more time is required to process the partitions, and query runtime can increase. If you have a table with a large number of partitions that grows over time, consider using AWS Glue partition indexing and filtering. Partition indexing allows Athena to optimize partition processing and improve query performance on highly partitioned tables. Setting up partition filtering in a table's properties is a two-step process:

1. Creating a partition index in AWS Glue.
2. Enabling partition filtering for the table.

Creating a partition index

For steps on creating a partition index in AWS Glue, see Working with partition indexes in the AWS Glue Developer Guide. For the limitations on partition indexes in AWS Glue, see the About partition indexes section on that page.

Enabling partition filtering

To enable partition filtering for the table, you must set a new table property in AWS Glue. For steps on how to set table properties in AWS Glue, refer to the Setting up partition projection page. When you edit the table details in AWS Glue, add the following key-value pair to the **Table properties** section:

- For **Key**, add `partition_filtering.enabled`
- For **Value**, add `true`

You can disable partition filtering on this table at any time by setting the `partition_filtering.enabled` value to `false`.

After you complete the above steps, you can return to the Athena console to query the data.

For more information about using partition indexing and filtering, see Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes in the *AWS Big Data Blog*.

Use the AWS CLI to recreate an AWS Glue database and its tables

Renaming a AWS Glue database directly is not possible, but you can copy its definition, modify the definition, and use the definition to recreate the database with a different name. Similarly, you can copy the definitions of the tables in the old database, modify the definitions, and use the modified definitions to recreate the tables in the new database.

> **Note**
>
> The method presented does not copy table partitioning.

The following procedure for Windows assumes that your AWS CLI is configured for JSON output. To change the default output format in the AWS CLI, run `aws configure`.

To copy an AWS Glue Database using the AWS CLI

1. At a command prompt, run the following AWS CLI command to retrieve the definition of the AWS Glue database that you want to copy.

   ```
   aws glue get-database --name database_name
   ```

   For more information about the `get-database` command, see get-database.
2. Save the JSON output to a file with the name of the new database (for example, *`new_database_name`*`.json`) to your desktop.
3. Open the *`new_database_name`*`.json` file in a text editor.
4. In the JSON file, perform the following steps:

   a. Remove the outer `{ "Database":` entry and the corresponding closing brace `}` at the end of the file.

   b. Change the `Name` entry to the new database name.

   c. Remove the `CatalogId` field.
5. Save the file.
6. At a command prompt, run the following AWS CLI command to use the modified database definition file to create the database with the new name.

   ```
   aws glue create-database --database-input "file://~/Desktop\new_database_name.json"
   ```

For more information about the `create-database` command, see [create-database](). For information about loading AWS CLI parameters from a file, see [Loading AWS CLI parameters from a file]() in the *AWS Command Line Interface User Guide*.

7. To verify that the new database has been created in AWS Glue, run the following command:

```
aws glue get-database --name new_database_name
```

Now you are ready to get the definition for a table that you want to copy to the new database, modify the definition, and use the modified definition to recreate the table in the new database. This procedure does not change the table name.

To copy an AWS Glue table using the AWS CLI

1. At a command prompt, run the following AWS CLI command.

```
aws glue get-table --database-name database_name --name table_name
```

   For more information about the `get-table` command, see [get-table]().

2. Save the JSON output to a file with the name of the table (for example, *table_name*.json) to your Windows desktop.
3. Open the file in a text editor.
4. In the JSON file, remove the outer `{"Table":` entry and the corresponding closing brace `}` at the end of the file.
5. In the JSON file, remove the following entries and their values:
   - `DatabaseName` – This entry is not required because the `create-table` CLI command uses the `--database-name` parameter.
   - `CreateTime`
   - `UpdateTime`
   - `CreatedBy`
   - `IsRegisteredWithLakeFormation`
   - `CatalogId`
   - `VersionId`
6. Save the table definition file.

7. At a command prompt, run the following AWS CLI command to recreate the table in the new database:

```
aws glue create-table --database-name new_database_name --table-input "file://~/
Desktop\table_name.json"
```

   For more information about the `create-table` command, see create-table.

   The table now appears in the new database in AWS Glue and can be queried from Athena.

8. Repeat the steps to copy each additional table to the new database in AWS Glue.

Create tables for ETL jobs

You can use Athena to create tables that AWS Glue can use for ETL jobs. AWS Glue jobs perform ETL operations. An AWS Glue job runs a script that extracts data from sources, transforms the data, and loads it into targets. For more information, see Authoring Jobs in AWS Glue in the *AWS Glue Developer Guide*.

Creating Athena tables for AWS Glue ETL jobs

Tables that you create in Athena must have a table property added to them called a `classification`, which identifies the format of the data. This allows AWS Glue to use the tables for ETL jobs. The classification values can be `avro`, `csv`, `json`, `orc`, `parquet`, or `xml`. An example `CREATE TABLE` statement in Athena follows:

```
CREATE EXTERNAL TABLE sampleTable (
  column1 INT,
  column2 INT
  ) STORED AS PARQUET
  TBLPROPERTIES (
  'classification'='parquet')
```

If the `classification` table property was not added when the table was created, you can add it using the AWS Glue console.

To add the classification table property using the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.

2. In the console navigation pane, choose **Tables**.
3. Choose the link for the table that you want to edit, and then choose **Actions**, **Edit table**.
4. Scroll down to the **Table properties** section.
5. Choose **Add**.
6. For **Key**, enter **`classification`**.
7. For **Value**, enter a data type (for example, **`json`**).
8. Choose **Save**.

   In the **Table details** section, the data type that you entered appears in the **Classification** field for the table.

For more information, see Working with tables in the *AWS Glue Developer Guide*.

Use ETL jobs to optimize query performance

AWS Glue jobs can help you transform data to a format that optimizes query performance in Athena. Data formats have a large impact on query performance and query costs in Athena.

AWS Glue supports writing to the Parquet and ORC data formats. You can use this feature to transform your data for use in Athena. For more information about using Parquet and ORC, and other ways to improve performance in Athena, see Top 10 performance tuning tips for Amazon Athena.

> **Note**
>
> To reduce the likelihood that Athena is unable to read the `SMALLINT` and `TINYINT` data types produced by an AWS Glue ETL job, convert `SMALLINT` and `TINYINT` to `INT` when you create an ETL job that converts data to ORC.

Automate AWS Glue jobs for ETL

You can configure AWS Glue ETL jobs to run automatically based on triggers. This feature is ideal when data from outside AWS is being pushed to an Amazon S3 bucket in an otherwise suboptimal format for querying in Athena. For more information, see Triggering AWS Glue jobs in the *AWS Glue Developer Guide*.

Work with CSV data in AWS Glue

This page describes how to use AWS Glue to create schema from CSV files that have quotes around the data values for each column or from CSV files that include header values.

Handling CSV data enclosed in quotes

Suppose a CSV file has data fields enclosed in double quotes, as in the following example.

```
"John","Doe","123-555-1231","John said \"hello\""
"Jane","Doe","123-555-9876","Jane said \"hello\""
```

To run a query in Athena on a table created from a CSV file that has quoted values, you must modify the table properties in AWS Glue to use the OpenCSVSerDe. For more information about the OpenCSV SerDe, see Open CSV SerDe for processing CSV.

To edit table properties in the AWS Glue console

1. In the AWS Glue console navigation pane, choose **Tables**.
2. Choose the link for the table that you want to edit, and then choose **Actions**, **Edit table**.
3. On the **Edit table** page, make the following changes:
   - For **Serialization lib**, enter `org.apache.hadoop.hive.serde2.OpenCSVSerde`.
   - For **Serde parameters**, enter the following values for the keys `escapeChar`, `quoteChar`, and `separatorChar`:
     - For `escapeChar`, enter a backslash (****).
     - For `quoteChar`, enter a double quote (**"**).
     - For `separatorChar`, enter a comma (**,**).
4. Choose **Save**.

For more information, see Viewing and editing table details in the *AWS Glue Developer Guide*.

You can also update AWS Glue table properties programmatically. Use the AWS Glue UpdateTable API operation or the update-table AWS CLI command to modify the `SerDeInfo` block in the table definition, as in the following JSON example.

```
"SerDeInfo": {
```

```
    "name": "",
    "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
    "parameters": {
       "separatorChar": ","
       "quoteChar": "\""
       "escapeChar": "\\"
       }
},
```

Handling CSV files with headers

When you define a table in Athena with a `CREATE TABLE` statement, you can use the `skip.header.line.count` table property to ignore headers in your CSV data, as in the following example.

```
...
STORED AS TEXTFILE
LOCATION 's3://amzn-s3-demo-bucket/csvdata_folder/';
TBLPROPERTIES ("skip.header.line.count"="1")
```

Alternatively, you can remove the CSV headers beforehand so that the header information is not included in Athena query results. One way to achieve this is to use AWS Glue jobs, which perform extract, transform, and load (ETL) work. You can write scripts in AWS Glue using a language that is an extension of the PySpark Python dialect. For more information, see Authoring Jobs in AWS Glue in the *AWS Glue Developer Guide*.

The following example shows a function in an AWS Glue script that writes out a dynamic frame using `from_options`, and sets the `writeHeader` format option to false, which removes the header information:

```
glueContext.write_dynamic_frame.from_options(frame = applymapping1, connection_type =
 "s3", connection_options = {"path": "s3://amzn-s3-demo-bucket/MYTABLEDATA/"}, format =
 "csv", format_options = {"writeHeader": False}, transformation_ctx = "datasink2")
```

Work with geospatial data in AWS Glue

AWS Glue does not natively support Well-known Text (WKT), Well-Known Binary (WKB), or other PostGIS data types. The AWS Glue classifier parses geospatial data and classifies them using supported data types for the format, such as `varchar` for CSV. As with other AWS Glue tables, you may need to update the properties of tables created from geospatial data to allow Athena to parse these data types as-is. For more information, see Use a crawler to add a table and Work with CSV data in AWS Glue. Athena may not be able to parse some geospatial data types in AWS Glue tables as-is. For more information about working with geospatial data in Athena, see Query geospatial data.

Use Amazon Athena Federated Query

If you have data in sources other than Amazon S3, you can use Athena Federated Query to query the data in place or build pipelines that extract data from multiple data sources and store them in Amazon S3. With Athena Federated Query, you can run SQL queries across data stored in relational, non-relational, object, and custom data sources.

Athena uses *data source connectors* that run on AWS Lambda to run federated queries. A data source connector is a piece of code that can translate between your target data source and Athena. You can think of a connector as an extension of Athena's query engine. Prebuilt Athena data source connectors exist for data sources like Amazon CloudWatch Logs, Amazon DynamoDB, Amazon DocumentDB, and Amazon RDS, and JDBC-compliant relational data sources such MySQL, and PostgreSQL under the Apache 2.0 license. You can also use the Athena Query Federation SDK to write custom connectors. To choose, configure, and deploy a data source connector to your account, you can use the Athena and Lambda consoles or the AWS Serverless Application Repository. After you deploy data source connectors, the connector is associated with a catalog that you can specify in SQL queries. You can combine SQL statements from multiple catalogs and span multiple data sources with a single query.

When a query is submitted against a data source, Athena invokes the corresponding connector to identify parts of the tables that need to be read, manages parallelism, and pushes down filter predicates. Based on the user submitting the query, connectors can provide or restrict access to specific data elements. Connectors use Apache Arrow as the format for returning data requested in a query, which enables connectors to be implemented in languages such as C, C++, Java, Python, and Rust. Since connectors are processed in Lambda, they can be used to access data from any data source on the cloud or on-premises that is accessible from Lambda.

To write your own data source connector, you can use the Athena Query Federation SDK to customize one of the prebuilt connectors that Amazon Athena provides and maintains. You can modify a copy of the source code from the GitHub repository and then use the Connector publish tool to create your own AWS Serverless Application Repository package.

Note

Third party developers may have used the Athena Query Federation SDK to write data source connectors. For support or licensing issues with these data source connectors, please work with your connector provider. These connectors are not tested or supported by AWS.

For a list of data source connectors written and tested by Athena, see Available data source connectors.

For information about writing your own data source connector, see Example Athena connector on GitHub.

Considerations and limitations

- **Engine versions** – Athena Federated Query is supported only on Athena engine version 2 and later. For information about Athena engine versions, see Athena engine versioning.
- **Views** – You can create and query views on federated data sources. Federated views are stored in AWS Glue, not the underlying data source. For more information, see Query federated views.
- **Delimited identifiers** – Delimited identifiers (also known as quoted identifiers) begin and end with double quotation marks ("). Currently, delimited identifiers are not supported for federated queries in Athena.
- **Write operations** – Write operations like INSERT INTO are not supported. Attempting to do so may result in the error message This operation is currently not supported for external catalogs.
- **Pricing** – For pricing information, see Amazon Athena pricing.
- **JDBC driver** – To use the JDBC driver with federated queries or an external Hive metastore, include `MetadataRetrievalMethod=ProxyAPI` in your JDBC connection string. For information about the JDBC driver, see Connect to Amazon Athena with JDBC.
- **Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, you must configure an Amazon VPC private endpoint for Secrets Manager. For more information, see Create a Secrets Manager VPC private endpoint in the *AWS Secrets Manager User Guide*.

Permissions required

Data source connectors might require access to the following resources to function correctly. If you use a prebuilt connector, check the information for the connector to ensure that you have configured your VPC correctly. Also, ensure that IAM principals running queries and creating connectors have privileges to required actions. For more information, see Allow access to Athena Federated Query: Example policies .

- **Amazon S3 –** In addition to writing query results to the Athena query results location in Amazon S3, data connectors also write to a spill bucket in Amazon S3. Connectivity and permissions to this Amazon S3 location are required. We recommend using spill to disk encryption for each connector and S3 lifecycle configuration to expire spilled data that is no longer needed.
- **Athena –** Data sources need connectivity to Athena and vice versa for checking query status and preventing overscan.
- **AWS Glue Data Catalog –** Connectivity and permissions are required if your connector uses Data Catalog for supplemental or primary metadata.
- **Amazon ECR –** Data source connector Lambda functions use an Amazon ECR image from an Amazon ECR repository. The user that deploys the connector must have the permissions `ecr:BatchGetImage` and `ecr:GetDownloadUrlForLayer`. For more information, see Amazon ECR permissions in the *AWS Lambda Developer Guide*.

Videos

Watch the following videos to learn more about using Athena Federated Query.

Video: Analyze Results of Federated Query in Amazon Athena in QuickSight

The following video demonstrates how to analyze results of an Athena federated query in QuickSight.

Analyze results of federated query in Amazon Athena in QuickSight

Video: Game Analytics Pipeline

The following video shows how to deploy a scalable serverless data pipeline to ingest, store, and analyze telemetry data from games and services using Amazon Athena federated queries.

Game analytics pipeline

Available data source connectors

This section lists prebuilt Athena data source connectors that you can use to query a variety of data sources external to Amazon S3. To use a connector in your Athena queries, configure it and deploy it to your account.

Considerations and limitations

- Some prebuilt connectors require that you create a VPC and a security group before you can use the connector. For information about creating VPCs, see Create a VPC for a data source connector or AWS Glue connection.
- To use the Athena Federated Query feature with AWS Secrets Manager, you must configure an Amazon VPC private endpoint for Secrets Manager. For more information, see Create a Secrets Manager VPC private endpoint in the *AWS Secrets Manager User Guide*.
- For connectors that do not support predicate pushdown, queries that include a predicate take longer to execute. For small datasets, very little data is scanned, and queries take an average of about 2 minutes. However, for large datasets, many queries can time out.
- Some federated data sources use terminology to refer data objects that is different from Athena. For more information, see Understand federated table name qualifiers.
- We update our connectors periodically based on upgrades from the database or data source provider. We do not support data sources that are at end-of-life for support.
- For connectors that do not support pagination when you list tables, the web service can time out if your database has many tables and metadata. The following connectors provide pagination support for listing tables:
  - DocumentDB
  - DynamoDB
  - MySQL
  - OpenSearch
  - Oracle
  - PostgreSQL
  - Redshift
  - SQL Server

Additional information

- For information about deploying an Athena data source connector, see Use Amazon Athena Federated Query.
- For information about queries that use Athena data source connectors, see Run federated queries.

Athena data source connectors

- Amazon Athena Azure Data Lake Storage (ADLS) Gen2 connector
- Amazon Athena Azure Synapse connector
- Amazon Athena Cloudera Hive connector
- Amazon Athena Cloudera Impala connector
- Amazon Athena CloudWatch connector
- Amazon Athena CloudWatch Metrics connector
- Amazon Athena AWS CMDB connector
- Amazon Athena IBM Db2 connector
- Amazon Athena IBM Db2 AS/400 (Db2 iSeries) connector
- Amazon Athena DocumentDB connector
- Amazon Athena DynamoDB connector
- Amazon Athena Google BigQuery connector
- Amazon Athena Google Cloud Storage connector
- Amazon Athena HBase connector
- Amazon Athena Hortonworks connector
- Amazon Athena Apache Kafka connector
- Amazon Athena MSK connector
- Amazon Athena MySQL connector
- Amazon Athena Neptune connector
- Amazon Athena OpenSearch connector
- Amazon Athena Oracle connector
- Amazon Athena PostgreSQL connector
- Amazon Athena Redis OSS connector
- Amazon Athena Redshift connector
- Amazon Athena SAP HANA connector
- Amazon Athena Snowflake connector
- Amazon Athena Microsoft SQL Server connector
- Amazon Athena Teradata connector
- Amazon Athena Timestream connector

- Amazon Athena TPC benchmark DS (TPC-DS) connector
- Amazon Athena Vertica connector

> **Note**
>
> The AthenaJdbcConnector (latest version 2022.4.1) has been deprecated. Instead, use a database-specific connector like those for MySQL, Redshift, or PostgreSQL.

Amazon Athena Azure Data Lake Storage (ADLS) Gen2 connector

The Amazon Athena connector for Azure Data Lake Storage (ADLS) Gen2 enables Amazon Athena to run SQL queries on data stored on ADLS. Athena cannot access stored files in the data lake directly.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

- **Workflow** – The connector implements the JDBC interface, which uses the `com.microsoft.sqlserver.jdbc.SQLServerDriver` driver. The connector passes queries to the Azure Synapse engine, which then accesses the data lake.
- **Data handling and S3** – Normally, the Lambda connector queries data directly without transfer to Amazon S3. However, when data returned by the Lambda function exceeds Lambda limits, the data is written to the Amazon S3 spill bucket that you specify so that Athena can read the excess.
- **AAD authentication** – AAD can be used as an authentication method for the Azure Synapse connector. In order to use AAD, the JDBC connection string that the connector uses must contain the URL parameters `authentication=ActiveDirectoryServicePrincipal`, `AADSecurePrincipalId`, and `AADSecurePrincipalSecret`. These parameters can either be passed in directly or by Secrets Manager.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Date and timestamp data types in filter conditions must be cast to appropriate data types.

Terms

The following terms relate to the Azure Data Lake Storage Gen2 connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Azure Data Lake Storage Gen2 connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
datalakegentwo://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`DataLakeGen2MuxCompositeHandler`
Metadata handler	`DataLakeGen2MuxMetadataHandler`
Record handler	`DataLakeGen2MuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mydatalakegentwocatalog` , then the environment variable name is `mydatalakegentwocatalog_connection_string` .
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a DataLakeGen2 MUX Lambda function that supports two database instances: `datalakegentwo1` (the default), and `datalakegentwo2`.

Property	Value
`default`	`datalakegentwo://jdbc:sqlserver://adlsgentwo1.`*`hostname`*`:`*`port`*`;databaseName=`*`database_name`*`;${`*`secret1_name`*`}`
`datalakegentwo_catalog1_connection_string`	`datalakegentwo://jdbc:sqlserver://adlsgentwo1.`*`hostname`*`:`*`port`*`;databaseName=`*`database_name`*`;${`*`secret1_name`*`}`
`datalakegentwo_catalog2_connection_string`	`datalakegentwo://jdbc:sqlserver://adlsgentwo2.`*`hostname`*`:`*`port`*`;databaseName=`*`database_name`*`;${`*`secret2_name`*`}`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

  > **Important**
  >
  > As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${secret1_name}`.

```
datalakegentwo://jdbc:sqlserver://hostname:port;databaseName=database_name;
${secret1_name}
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
datalakegentwo://
jdbc:sqlserver://
hostname:port;databaseName=database_name;user=user_name;password=password
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Azure Data Lake Storage Gen2 instance.

Handler type	Class
Composite handler	`DataLakeGen2CompositeHandler`
Metadata handler	`DataLakeGen2MetadataHandler`
Record handler	`DataLakeGen2RecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Azure Data Lake Storage Gen2 instance supported by a Lambda function.

Property	Value
`default`	`datalakegentwo://jdbc:sqlserver:// hostname:port;databaseName=;${ secret_name }`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for ADLS Gen2 and Arrow.

ADLS Gen2	Arrow
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)

ADLS Gen2	Arrow
char[n]	VARCHAR
varchar[n/max]	VARCHAR

Partitions and splits

Azure Data Lake Storage Gen2 uses Hadoop compatible Gen2 blob storage for storing data files. The data from these files is queried from the Azure Synapse engine. The Azure Synapse engine treats Gen2 data stored in file systems as external tables. The partitions are implemented based on the type of data. If the data has already been partitioned and distributed within the Gen 2 storage system, the connector retrieves the data as single split.

Performance

The Azure Data Lake Storage Gen2 connector shows slower query performance when running multiple queries at once, and is subject to throttling.

The Athena Azure Data Lake Storage Gen2 connector performs predicate pushdown to decrease the data scanned by the query. Simple predicates and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Azure Data Lake Storage Gen2 connector can combine these expressions and push them directly to Azure Data Lake Storage Gen2 for enhanced functionality and to reduce the amount of data scanned.

The following Athena Azure Data Lake Storage Gen2 connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

Passthrough queries

The Azure Data Lake Storage Gen2 connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Azure Data Lake Storage Gen2, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Azure Data Lake Storage Gen2. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Azure Data Lake Storage Gen2 connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Azure Synapse connector

The Amazon Athena connector for Azure Synapse analytics enables Amazon Athena to run SQL queries on your Azure Synapse databases using JDBC.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- In filter conditions, you must cast the `Date` and `Timestamp` data types to the appropriate data type.
- To search for negative values of type `Real` and `Float`, use the <= or >= operator.
- The `binary`, `varbinary`, `image`, and `rowversion` data types are not supported.

Terms

The following terms relate to the Synapse connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.

- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Synapse connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
synapse://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	SynapseMuxCompositeHandler
Metadata handler	SynapseMuxMetadataHandler
Record handler	SynapseMuxRecordHandler

Multiplexing handler parameters

Parameter	Description
$*catalog*_connection_string	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is mysynapsecatalog , then the environment variable name is mysynapsecatalog_connection_string .
default	Required. The default connection string. This string is used when the catalog is lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }.

The following example properties are for a Synapse MUX Lambda function that supports two database instances: synapse1 (the default), and synapse2.

Property	Value
default	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog1_connection_string	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog2_c	synapse://jdbc:synapse://synapse2.hostname:port;databaseName= *<database_name>* ;${*secret2_name* }

Property	Value
connection_string	

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name ${secret_name}.

```
synapse://jdbc:synapse://hostname:port;databaseName=<database_name>;${secret_name}
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
synapse://jdbc:synapse://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Synapse instance.

Handler type	Class
Composite handler	`SynapseCompositeHandler`
Metadata handler	`SynapseMetadataHandler`
Record handler	`SynapseRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Synapse instance supported by a Lambda function.

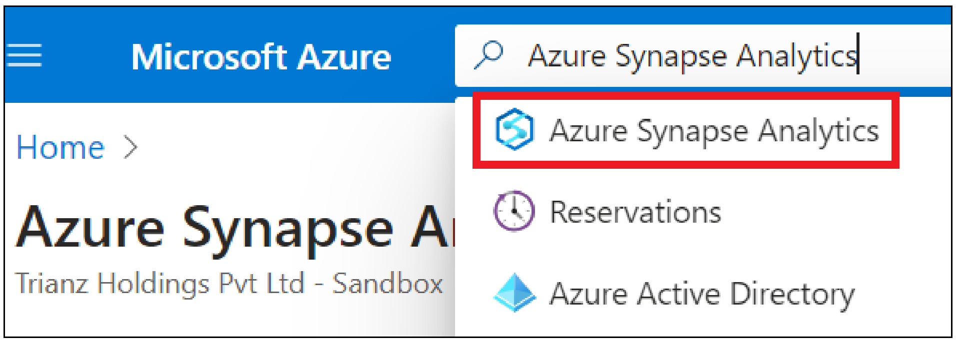
Prope	Value
defa	`synapse://jdbc:sqlserver://hostname:port;databaseName= <database_name> ;${secret_name }`

Configuring Active Directory authentication

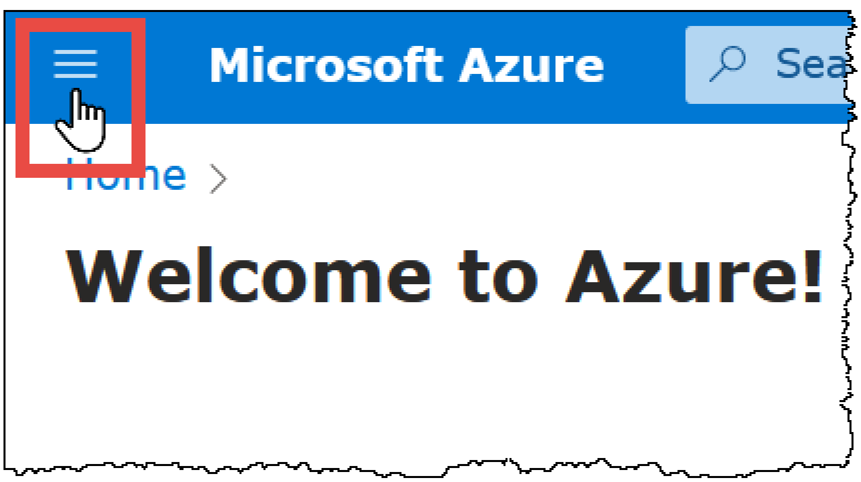
The Amazon Athena Azure Synapse connector supports Microsoft Active Directory Authentication. Before you begin, you must configure an administrative user in the Microsoft Azure portal and then use AWS Secrets Manager to create a secret.

To set the Active Directory administrative user

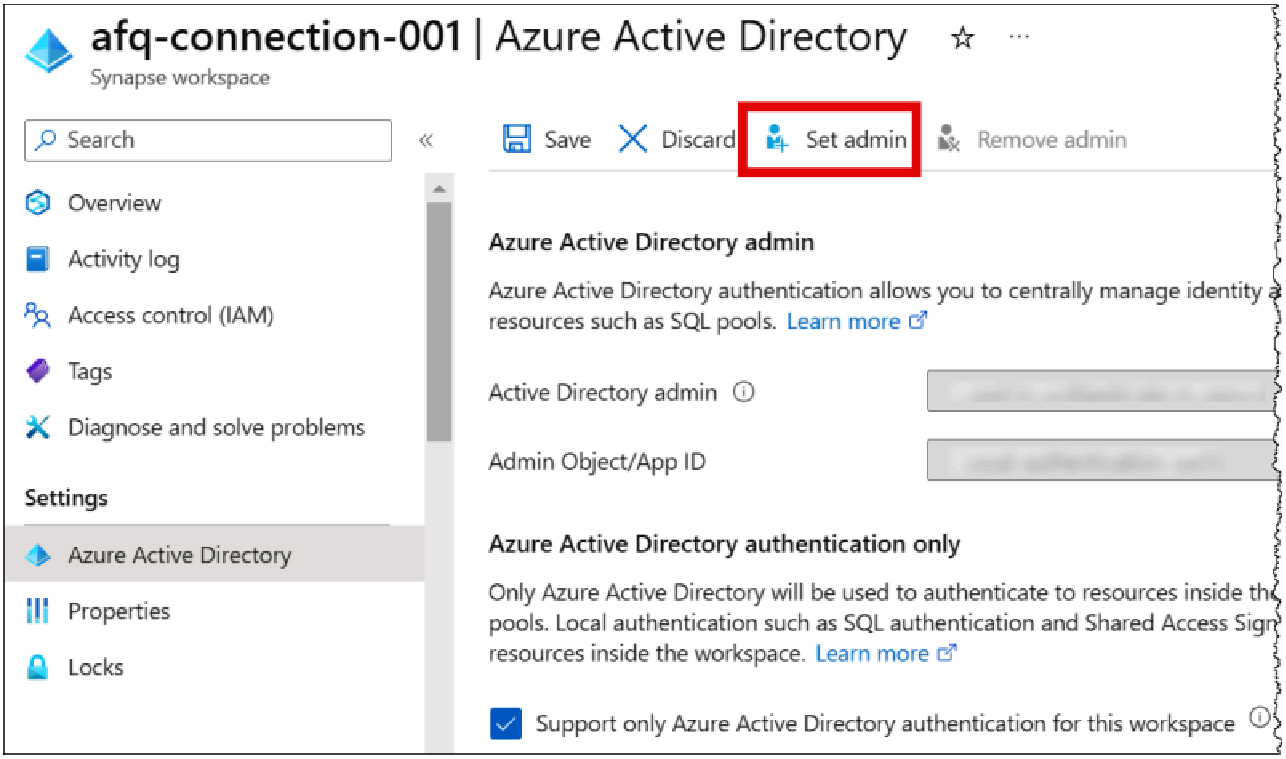
1. Using an account that has administrative privileges, sign in to the Microsoft Azure portal at https://portal.azure.com/.
2. In the search box, enter **Azure Synapse Analytics**, and then choose **Azure Synapse Analytics**.



3. Open the menu on the left.



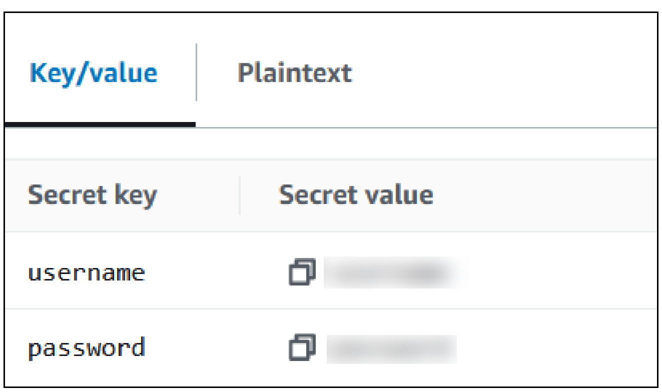
4. In the navigation pane, choose **Azure Active Directory**.
5. On the **Set admin** tab, set **Active Directory admin** to a new or existing user.



6. In AWS Secrets Manager, store the admin username and password credentials. For information on creating a secret in Secrets Manager, see [Create an AWS Secrets Manager secret](#).

To view your secret in Secrets Manager

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the navigation pane, choose **Secrets**.
3. On the **Secrets** page, choose the link to your secret.
4. On the details page for your secret, choose **Retrieve secret value**.



Modifying the connection string

To enable Active Directory Authentication for the connector, modify the connection string using the following syntax:

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryPassword;
{secret_name}
```

Using ActiveDirectoryServicePrincipal

The Amazon Athena Azure Synapse connector also supports `ActiveDirectoryServicePrincipal`. To enable this, modify the connection string as follows.

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryServicePrincipal;
{secret_name}
```

For `secret_name`, specify the application or client ID as the username and the secret of a service principal identity in the password.

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see [PutObject](#) in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for Synapse and Apache Arrow.

Synapse	Arrow
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)

Synapse	Arrow
char[n]	VARCHAR
varchar[n/max]	VARCHAR
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR

Partitions and splits

A partition is represented by a single partition column of type `varchar`. Synapse supports range partitioning, so partitioning is implemented by extracting the partition column and partition range from Synapse metadata tables. These range values are used to create the splits.

Performance

Selecting a subset of columns significantly slows down query runtime. The connector shows significant throttling due to concurrency.

The Athena Synapse connector performs predicate pushdown to decrease the data scanned by the query. Simple predicates and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Synapse connector can combine these expressions and push them directly to Synapse for enhanced functionality and to reduce the amount of data scanned.

The following Athena Synapse connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

Passthrough queries

The Synapse connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Synapse, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Synapse. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

- For an article that shows how to use QuickSight and Amazon Athena Federated Query to build dashboards and visualizations on data stored in Microsoft Azure Synapse databases, see Perform multi-cloud analytics using QuickSight, Amazon Athena Federated Query, and Microsoft Azure Synapse in the *AWS Big Data Blog*.

- For the latest JDBC driver version information, see the pom.xml file for the Synapse connector on GitHub.com.
- For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Cloudera Hive connector

The Amazon Athena connector for Cloudera Hive enables Athena to run SQL queries on the Cloudera Hive Hadoop distribution. The connector transforms your Athena SQL queries to their equivalent HiveQL syntax.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.

Terms

The following terms relate to the Cloudera Hive connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.

- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Cloudera Hive connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
hive://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`HiveMuxCompositeHandler`
Metadata handler	`HiveMuxMetadataHandler`
Record handler	`HiveMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `myhivecatalog`, then the environment variable name is `myhivecatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Hive MUX Lambda function that supports two database instances: `hive1` (the default), and `hive2`.

Property	Value
`default`	`hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}`
`hive2_catalog1_connection_string`	`hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}`
`hive2_catalog2_connection_string`	`hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, the Cloudera Hive connector requires a secret from AWS Secrets Manager. To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

Put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

Example connection string with secret name

The following string has the secret name `${Test/RDS/hive1}`.

```
hive://jdbc:hive2://hive1:10000/default?...&${Test/RDS/hive1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
hive://jdbc:hive2://hive1:10000/default?...&UID=sample2&PWD=sample2&...
```

Currently, the Cloudera Hive connector recognizes the `UID` and `PWD` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Cloudera Hive instance.

Handler type	Class
Composite handler	`HiveCompositeHandler`
Metadata handler	`HiveMetadataHandler`
Record handler	`HiveRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Cloudera Hive instance supported by a Lambda function.

Property	Value
default	`hive://jdbc:hive2://hive1:10000/default?secret=${Test/RDS/hive1}`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC, Cloudera Hive, and Arrow.

JDBC	Cloudera Hive	Arrow
Boolean	Boolean	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint
Integer	INT	Int

JDBC	Cloudera Hive	Arrow
Long	BIGINT	Bigint
float	float4	Float4
Double	float8	Float8
Date	date	DateDay
Timestamp	timestamp	DateMilli
String	VARCHAR	Varchar
Bytes	bytes	Varbinary
BigDecimal	Decimal	Decimal
ARRAY	N/A (see note)	List

Note

Currently, Cloudera Hive does not support the aggregate types `ARRAY`, `MAP`, `STRUCT`, or `UNIONTYPE`. Columns of aggregate types are treated as `VARCHAR` columns in SQL.

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

Cloudera Hive supports static partitions. The Athena Cloudera Hive connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, static partitioning is highly recommended. The Cloudera Hive connector is resilient to throttling due to concurrency.

The Athena Cloudera Hive connector performs predicate pushdown to decrease the data scanned by the query. LIMIT clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A LIMIT N statement reduces the data scanned by the query. With LIMIT N pushdown, the connector returns only N rows to Athena.

Predicates

A predicate is an expression in the WHERE clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Cloudera Hive connector can combine these expressions and push them directly to Cloudera Hive for enhanced functionality and to reduce the amount of data scanned.

The following Athena Cloudera Hive connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The Cloudera Hive connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Cloudera Hive, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Cloudera Hive. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Cloudera Hive connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Cloudera Impala connector

The Amazon Athena Cloudera Impala connector enables Athena to run SQL queries on the Cloudera Impala distribution. The connector transforms your Athena SQL queries to the equivalent Impala syntax.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.

Terms

The following terms relate to the Cloudera Impala connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Cloudera Impala connector.

Connection string

Use a JDBC connection string in the following format to connect to an Impala cluster.

```
impala://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`ImpalaMuxCompositeHandler`
Metadata handler	`ImpalaMuxMetadataHandler`
Record handler	`ImpalaMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. An Impala cluster connection string for an Athena catalog. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `myimpalacatalog`, then the environment variable name is `myimpalacatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Impala MUX Lambda function that supports two database instances: `impala1` (the default), and `impala2`.

Property	Value
default	impala://jdbc:impala://some.impala.host.name:21050/?${Test/impala1}
impala_catalog1_connection_string	impala://jdbc:impala://someother.impala.host.name:21050/?${Test/impala1}
impala_catalog2_connection_string	impala://jdbc:impala://another.impala.host.name:21050/?UID=sample&PWD=sample

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/impala1host}`.

```
impala://jdbc:impala://Impala1host:21050/?...&${Test/impala1host}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
impala://jdbc:impala://Impala1host:21050/?...&UID=sample2&PWD=sample2&...
```

Currently, Cloudera Impala recognizes the `UID` and `PWD` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Cloudera Impala instance.

Handler type	Class
Composite handler	`ImpalaCompositeHandler`
Metadata handler	`ImpalaMetadataHandler`
Record handler	`ImpalaRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Cloudera Impala instance supported by a Lambda function.

Property	Value
`default`	`impala://jdbc:impala://Impala1host:21050/?secret=${Test/impala1host}`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC, Cloudera Impala, and Arrow.

JDBC	Cloudera Impala	Arrow
Boolean	Boolean	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint

JDBC	Cloudera Impala	Arrow
Integer	INT	Int
Long	BIGINT	Bigint
float	float4	Float4
Double	float8	Float8
Date	date	DateDay
Timestamp	timestamp	DateMilli
String	VARCHAR	Varchar
Bytes	bytes	Varbinary
BigDecimal	Decimal	Decimal
ARRAY	N/A (see note)	List

Note

Currently, Cloudera Impala does not support the aggregate types `ARRAY`, `MAP`, `STRUCT`, or `UNIONTYPE`. Columns of aggregate types are treated as `VARCHAR` columns in SQL.

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

Cloudera Impala supports static partitions. The Athena Cloudera Impala connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, static partitioning is highly recommended. The Cloudera Impala connector is resilient to throttling due to concurrency.

The Athena Cloudera Impala connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Cloudera Impala connector can combine these expressions and push them directly to Cloudera Impala for enhanced functionality and to reduce the amount of data scanned.

The following Athena Cloudera Impala connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The Cloudera Impala connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Cloudera Impala, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Cloudera Impala. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Cloudera Impala connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena CloudWatch connector

The Amazon Athena CloudWatch connector enables Amazon Athena to communicate with CloudWatch so that you can query your log data with SQL.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

The connector maps your LogGroups as schemas and each LogStream as a table. The connector also maps a special `all_log_streams` view that contains all LogStreams in the LogGroup. This view enables you to query all the logs in a LogGroup at once instead of searching through each LogStream individually.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the CloudWatch connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.

The connector also supports AIMD congestion control for handling throttling events from CloudWatch through the Amazon Athena Query Federation SDK `ThrottlingInvoker` construct. You can tweak the default throttling behavior by setting any of the following optional environment variables:

- **throttle_initial_delay_ms** – The initial call delay applied after the first congestion event. The default is 10 milliseconds.
- **throttle_max_delay_ms** – The maximum delay between calls. You can derive TPS by dividing it into 1000ms. The default is 1000 milliseconds.

- **throttle_decrease_factor** – The factor by which Athena reduces the call rate. The default is 0.5
- **throttle_increase_ms** – The rate at which Athena decreases the call delay. The default is 10 milliseconds.

Databases and tables

The Athena CloudWatch connector maps your LogGroups as schemas (that is, databases) and each LogStream as a table. The connector also maps a special `all_log_streams` view that contains all LogStreams in the LogGroup. This view enables you to query all the logs in a LogGroup at once instead of searching through each LogStream individually.

Every table mapped by the Athena CloudWatch connector has the following schema. This schema matches the fields provided by CloudWatch Logs.

- **log_stream** – A `VARCHAR` that contains the name of the LogStream that the row is from.
- **time** – An `INT64` that contains the epoch time of when the log line was generated.
- **message** – A `VARCHAR` that contains the log message.

Examples

The following example shows how to perform a `SELECT` query on a specified LogStream.

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."log_stream_name"
LIMIT 100
```

The following example shows how to use the `all_log_streams` view to perform a query on all LogStreams in a specified LogGroup.

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."all_log_streams"
LIMIT 100
```

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-cloudwatch.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **CloudWatch Logs Read/Write** – The connector uses this permission to read your log data and to write its diagnostic logs.

Performance

The Athena CloudWatch connector attempts to optimize queries against CloudWatch by parallelizing scans of the log streams required for your query. For certain time period filters, predicate pushdown is performed both within the Lambda function and within CloudWatch Logs.

For best performance, use only lowercase for your log group names and log stream names. Using mixed casing causes the connector to perform a case insensitive search that is more computationally intensive.

Passthrough queries

The CloudWatch connector supports passthrough queries that use CloudWatch Logs Insights query syntax. For more information about CloudWatch Logs Insights, see Analyzing log data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*.

To create passthrough queries with CloudWatch, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => 'start_time',
            ENDTIME => 'end_time',
            QUERYSTRING => 'query_string',
            LOGGROUPNAMES => 'log_group-names',
            LIMIT => 'max_number_of_results'
        ))
```

The following example CloudWatch passthrough query filters for the `duration` field when it does not equal 1000.

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => '1710918615308',
```

```
            ENDTIME => '1710918615972',
            QUERYSTRING => 'fields @duration | filter @duration != 1000',
            LOGGROUPNAMES => '/aws/lambda/cloudwatch-test-1',
            LIMIT => '2'
            ))
```

License information

The Amazon Athena CloudWatch connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena CloudWatch Metrics connector

The Amazon Athena CloudWatch Metrics connector enables Amazon Athena to query CloudWatch Metrics data with SQL.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

For information on publishing query metrics to CloudWatch from Athena itself, see Use CloudWatch and EventBridge to monitor queries and control costs.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the CloudWatch Metrics connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-`

side-encryption" : "AES256"}). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.

The connector also supports AIMD congestion control for handling throttling events from CloudWatch through the Amazon Athena Query Federation SDK `ThrottlingInvoker` construct. You can tweak the default throttling behavior by setting any of the following optional environment variables:

- **throttle_initial_delay_ms** – The initial call delay applied after the first congestion event. The default is 10 milliseconds.
- **throttle_max_delay_ms** – The maximum delay between calls. You can derive TPS by dividing it into 1000ms. The default is 1000 milliseconds.
- **throttle_decrease_factor** – The factor by which Athena reduces the call rate. The default is 0.5
- **throttle_increase_ms** – The rate at which Athena decreases the call delay. The default is 10 milliseconds.

Databases and tables

The Athena CloudWatch Metrics connector maps your namespaces, dimensions, metrics, and metric values into two tables in a single schema called `default`.

The metrics table

The `metrics` table contains the available metrics as uniquely defined by a combination of namespace, set, and name. The `metrics` table contains the following columns.

- **namespace** – A VARCHAR containing the namespace.
- **metric_name** – A VARCHAR containing the metric name.

- **dimensions** – A LIST of STRUCT objects composed of dim_name (VARCHAR) and dim_value (VARCHAR).
- **statistic** – A LIST of VARCH statistics (for example, p90, AVERAGE, ...) available for the metric.

The metric_samples table

The metric_samples table contains the available metric samples for each metric in the metrics table. The metric_samples table contains the following columns.

- **namespace** – A VARCHAR that contains the namespace.
- **metric_name** – A VARCHAR that contains the metric name.
- **dimensions** – A LIST of STRUCT objects composed of dim_name (VARCHAR) and dim_value (VARCHAR).
- **dim_name** – A VARCHAR convenience field that you can use to easily filter on a single dimension name.
- **dim_value** – A VARCHAR convenience field that you can use to easily filter on a single dimension value.
- **period** – An INT field that represents the "period" of the metric in seconds (for example, a 60 second metric).
- **timestamp** – A BIGINT field that represents the epoch time in seconds that the metric sample is for.
- **value** – A FLOAT8 field that contains the value of the sample.
- **statistic** – A VARCHAR that contains the statistic type of the sample (for example, AVERAGE or p90).

Required Permissions

For full details on the IAM policies that this connector requires, review the Policies section of the athena-cloudwatch-metrics.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **CloudWatch Metrics ReadOnly** – The connector uses this permission to query your metrics data.

- **CloudWatch Logs Write** – The connector uses this access to write its diagnostic logs.

Performance

The Athena CloudWatch Metrics connector attempts to optimize queries against CloudWatch Metrics by parallelizing scans of the log streams required for your query. For certain time period, metric, namespace, and dimension filters, predicate pushdown is performed both within the Lambda function and within CloudWatch Logs.

License information

The Amazon Athena CloudWatch Metrics connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena AWS CMDB connector

The Amazon Athena AWS CMDB connector enables Athena to communicate with various AWS services so that you can query them with SQL.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the AWS CMDB connector.

Glue connections (recommended)

We recommended that you configure a AWS CMDB connector by using a Glue connections object. To do this, set the `glue_connection` environment variable of the AWS CMDB connector Lambda to the name of the Glue connection to use.

Glue connections properties

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type CMDB
```

Lambda environment properties

glue_connection – Specifies the name of the Glue connection associated with the federated connector.

Legacy connections

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Lambda environment properties

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.

- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **default_ec2_image_owner** – (Optional) When set, controls the default Amazon EC2 image owner that filters Amazon Machine Images (AMI). If you do not set this value and your query against the EC2 images table does not include a filter for owner, your results will include all public images.

Databases and tables

The Athena AWS CMDB connector makes the following databases and tables available for querying your AWS resource inventory. For more information on the columns available in each table, run a `DESCRIBE database.table` statement using the Athena console or API.

- **ec2** – This database contains Amazon EC2 related resources, including the following.

- **ebs_volumes** – Contains details of your Amazon EBS volumes.
- **ec2_instances** – Contains details of your EC2 Instances.
- **ec2_images** – Contains details of your EC2 Instance images.
- **routing_tables** – Contains details of your VPC Routing Tables.
- **security_groups** – Contains details of your security groups.
- **subnets** – Contains details of your VPC Subnets.
- **vpcs** – Contains details of your VPCs.

- **emr** – This database contains Amazon EMR related resources, including the following.

- **emr_clusters** – Contains details of your EMR Clusters.

- **rds** – This database contains Amazon RDS related resources, including the following.

- **rds_instances** – Contains details of your RDS Instances.

- **s3 –** This database contains RDS related resources, including the following.

- **buckets** – Contains details of your Amazon S3 buckets.
- **objects** – Contains details of your Amazon S3 objects, excluding their contents.

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-aws-cmdb.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **S3 List** – The connector uses this permission to list your Amazon S3 buckets and objects.
- **EC2 Describe** – The connector uses this permission to describe resources such as your Amazon EC2 instances, security groups, VPCs, and Amazon EBS volumes.
- **EMR Describe / List** – The connector uses this permission to describe your EMR clusters.
- **RDS Describe** – The connector uses this permission to describe your RDS Instances.

Performance

Currently, the Athena AWS CMDB connector does not support parallel scans. Predicate pushdown is performed within the Lambda function. Where possible, partial predicates are pushed to the services being queried. For example, a query for the details of a specific Amazon EC2 instance calls the EC2 API with the specific instance ID to run a targeted describe operation.

License information

The Amazon Athena AWS CMDB connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena IBM Db2 connector

The Amazon Athena connector for Db2 enables Amazon Athena to run SQL queries on your IBM Db2 databases using JDBC.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Date and timestamp data types in filter conditions must be cast to appropriate data types.

Terms

The following terms relate to the Db2 connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.

- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Db2 connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding [AWS Glue connection properties](). Starting December 3, 2024, use the parameters below only when you [manually deploy]() an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
dbtwo://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`Db2MuxCompositeHandler`
Metadata handler	`Db2MuxMetadataHandler`
Record handler	`Db2MuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mydbtwocatalog`, then the environment variable name is `mydbtwocatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Db2 MUX Lambda function that supports two database instances: `dbtwo1` (the default), and `dbtwo2`.

Property	Value
`default`	`dbtwo://jdbc:db2://dbtwo1.hostname:port/ database_name :${secret1_name }`
`dbtwo_catalog1_connection_string`	`dbtwo://jdbc:db2://dbtwo1. hostname:port/ database_name :${secret1_name }`
`dbtwo_catalog2_connection_string`	`dbtwo://jdbc:db2://dbtwo2. hostname:port/ database_name :${secret2_name }`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

  ```
  {"username": "${username}", "password": "${password}"}
  ```

Example connection string with secret name

The following string has the secret name `${secret_name}`.

```
dbtwo://jdbc:db2://hostname:port/database_name:${secret_name}
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
dbtwo://jdbc:db2://hostname:port/database_name:user=user_name;password=password;
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Db2 instance.

Handler type	Class
Composite handler	`Db2CompositeHandler`
Metadata handler	`Db2MetadataHandler`
Record handler	`Db2RecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Db2 instance supported by a Lambda function.

Property	Value
`default`	dbtwo://jdbc:db2://hostname:port/*database_name* :${secret_name}

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Arrow.

Db2	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR
DATE	DATEDAY
TIME	VARCHAR
TIMESTAMP	DATEMILLI
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT

Db2	Arrow
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8
DECFLOAT	FLOAT8

Partitions and splits

A partition is represented by one or more partition columns of type `varchar`. The Db2 connector creates partitions using the following organization schemes.

- Distribute by hash
- Partition by range
- Organize by dimensions

The connector retrieves partition details such as the number of partitions and column name from one or more Db2 metadata tables. Splits are created based upon the number of partitions identified.

Performance

The Athena Db2 connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Db2 connector can combine these expressions and push them directly to Db2 for enhanced functionality and to reduce the amount of data scanned.

The following Athena Db2 connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The Db2 connector supports [passthrough queries](). Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Db2, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Db2. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
```

```
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Db2 connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena IBM Db2 AS/400 (Db2 iSeries) connector

The Amazon Athena connector for Db2 AS/400 enables Amazon Athena to run SQL queries on your IBM Db2 AS/400 (Db2 iSeries) databases using JDBC.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Date and timestamp data types in filter conditions must be cast to appropriate data types.

Terms

The following terms relate to the Db2 AS/400 connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Db2 AS/400 connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
db2as400://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`Db2MuxCompositeHandler`
Metadata handler	`Db2MuxMetadataHandler`
Record handler	`Db2MuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mydb2as400catalog`, then the environment variable name is `mydb2as400catalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Db2 MUX Lambda function that supports two database instances: `db2as4001` (the default), and `db2as4002`.

Property	Value
`default`	`db2as400://jdbc:as400:// <ip_address> ;<properties> ;:${<secret name>};`

Property	Value
db2as400_catalog1_connection_string	db2as400://jdbc:as400://db2as4001. hostname/ :${ *secret1_name* }
db2as400_catalog2_connection_string	db2as400://jdbc:as400://db2as4002. hostname/ :${ *secret2_name* }
db2as400_catalog3_connection_string	db2as400://jdbc:as400:// *<ip_address>* ;user=*<username>* ;password= *<password>* ;*<properties>* ;

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see [Move hardcoded secrets to AWS Secrets Manager]() in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have [internet access]() or a [VPC endpoint]() to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${secret_name}`.

```
db2as400://jdbc:as400://<ip_address>;<properties>;:${<secret_name>};
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
db2as400://jdbc:as400://<ip_address>;user=<username>;password=<password>;<properties>;
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Db2 AS/400 instance.

Handler type	Class
Composite handler	`Db2CompositeHandler`
Metadata handler	`Db2MetadataHandler`
Record handler	`Db2RecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Db2 AS/400 instance supported by a Lambda function.

Property	Value
`default`	`db2as400://jdbc:as400:// <ip_address> ;<properties> ;:${<secret_name> };`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Apache Arrow.

Db2 AS/400	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR

Db2 AS/400	Arrow
DATE	DATEDAY
TIME	VARCHAR
TIMESTAMP	DATEMILLI
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8
DECFLOAT	FLOAT8

Partitions and splits

A partition is represented by one or more partition columns of type `varchar`. The Db2 AS/400 connector creates partitions using the following organization schemes.

- Distribute by hash
- Partition by range
- Organize by dimensions

The connector retrieves partition details such as the number of partitions and column name from one or more Db2 AS/400 metadata tables. Splits are created based upon the number of partitions identified.

Performance

For improved performance, use predicate pushdown to query from Athena, as in the following examples.

```
SELECT * FROM "lambda:<LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE integercol = 2147483647
```

```
SELECT * FROM "lambda: <LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

Passthrough queries

The Db2 AS/400 connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Db2 AS/400, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Db2 AS/400. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Db2 AS/400 connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena DocumentDB connector

The Amazon Athena DocumentDB connector enables Athena to communicate with your DocumentDB instances so that you can query your DocumentDB data with SQL. The connector also works with any endpoint that is compatible with MongoDB.

Unlike traditional relational data stores, Amazon DocumentDB collections do not have set schema. DocumentDB does not have a metadata store. Each entry in a DocumentDB collection can have different fields and data types.

The DocumentDB connector supports two mechanisms for generating table schema information: basic schema inference and AWS Glue Data Catalog metadata.

Schema inference is the default. This option scans a small number of documents in your collection, forms a union of all fields, and coerces fields that have non-overlapping data types. This option works well for collections that have mostly uniform entries.

For collections with a greater variety of data types, the connector supports retrieving metadata from the AWS Glue Data Catalog. If the connector sees a AWS Glue database and table that match your DocumentDB database and collection names, it gets its schema information from the corresponding AWS Glue table. When you create your AWS Glue table, we recommend that you make it a superset of all fields that you might want to access from your DocumentDB collection.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the DocumentDB connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **disable_glue** – (Optional) If present and set to true, the connector does not attempt to retrieve supplemental metadata from AWS Glue.
- **glue_catalog** – (Optional) Use this option to specify a cross-account AWS Glue catalog. By default, the connector attempts to get metadata from its own AWS Glue account.
- **default_docdb** – If present, specifies a DocumentDB connection string to use when no catalog-specific environment variable exists.

- **disable_projection_and_casing** – (Optional) Disables projection and casing. Use if you want to query Amazon DocumentDB tables that use case sensitive column names. The `disable_projection_and_casing` parameter uses the following values to specify the behavior of casing and column mapping:
  - **false** – This is the default setting. Projection is enabled, and the connector expects all column names to be in lower case.
  - **true** – Disables projection and casing. When using the `disable_projection_and_casing` parameter, keep in mind the following points:
    - Use of the parameter can result in higher bandwidth usage. Additionally, if your Lambda function is not in the same AWS Region as your data source, you will incur higher standard AWS cross-region transfer costs as a result of the higher bandwidth usage. For more information about cross-region transfer costs, see [AWS Data Transfer Charges for Server and Serverless Architectures] in the AWS Partner Network Blog.
    - Because a larger number of bytes is transferred and because the larger number of bytes requires a higher deserialization time, overall latency can increase.
- **enable_case_insensitive_match** – (Optional) When `true`, performs case insensitive searches against schema and table names in Amazon DocumentDB. The default is `false`. Use if your query contains uppercase schema or table names.

Specifying connection strings

You can provide one or more properties that define the DocumentDB connection details for the DocumentDB instances that you use with the connector. To do this, set a Lambda environment variable that corresponds to the catalog name that you want to use in Athena. For example, suppose you want to use the following queries to query two different DocumentDB instances from Athena:

```
SELECT * FROM "docdb_instance_1".database.table
```

```
SELECT * FROM "docdb_instance_2".database.table
```

Before you can use these two SQL statements, you must add two environment variables to your Lambda function: `docdb_instance_1` and `docdb_instance_2`. The value for each should be a DocumentDB connection string in the following format:

```
mongodb://:@:/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

Using secrets

You can optionally use AWS Secrets Manager for part or all of the value for your connection string details. To use the Athena Federated Query feature with Secrets Manager, the VPC connected to your Lambda function should have [internet access](https://aws.amazon.com/premiumsupport/knowledge-center/internet-access-lambda-function/) or a [VPC endpoint](https://docs.aws.amazon.com/secretsmanager/latest/userguide/vpc-endpoint-overview.html) to connect to Secrets Manager.

If you use the syntax `${my_secret}` to put the name of a secret from Secrets Manager in your connection string, the connector replaces `${my_secret}` with its plain text value from Secrets Manager exactly. Secrets should be stored as a plain text secret with value *<username>*`:`*<password>*. Secrets stored as `{username:`*<username>*`,password:`*<password>*`}` will not be passed to the connection string properly.

Secrets can also be used for the entire connection string entirely, and the username and password can be defined within the secret.

For example, suppose you set the Lambda environment variable for `docdb_instance_1` to the following value:

```
mongodb://${docdb_instance_1_creds}@myhostname.com:123/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

The Athena Query Federation SDK automatically attempts to retrieve a secret named `docdb_instance_1_creds` from Secrets Manager and inject that value in place of `${docdb_instance_1_creds}`. Any part of the connection string that is enclosed by the `${ }` character combination is interpreted as a secret from Secrets Manager. If you specify a secret name that the connector cannot find in Secrets Manager, the connector does not replace the text.

Setting up databases and tables in AWS Glue

Because the connector's built-in schema inference capability scans a limited number of documents and supports only a subset of data types, you might want to use AWS Glue for metadata instead.

To enable an AWS Glue table for use with Amazon DocumentDB, you must have a AWS Glue database and table for the DocumentDB database and collection that you want to supply supplemental metadata for.

To use an AWS Glue table for supplemental metadata

1. Use the AWS Glue console to create an AWS Glue database that has the same name as your Amazon DocumentDB database name.
2. Set the URI property of the database to include **docdb-metadata-flag**.
3. (Optional) Add the **sourceTable** table property. This property defines the source table name in Amazon DocumentDB. Use this property if your AWS Glue table has a different name from the table name in Amazon DocumentDB. Differences in naming rules between AWS Glue and Amazon DocumentDB can make this necessary. For example, capital letters are not permitted in AWS Glue table names, but they are permitted in Amazon DocumentDB table names.
4. (Optional) Add the **columnMapping** table property. This property defines column name mappings. Use this property if AWS Glue column naming rules prevent you from creating an AWS Glue table that has the same column names as those in your Amazon DocumentDB table. This can be useful because capital letters are permitted in Amazon DocumentDB column names but are not permitted in AWS Glue column names.

   The `columnMapping` property value is expected to be a set of mappings in the format `col1=Col1,col2=Col2`.

   > **Note**
   >
   > Column mapping applies only to top level column names and not to nested fields.

   After you add the AWS Glue `columnMapping` table property, you can remove the `disable_projection_and_casing` Lambda environment variable.
5. Make sure that you use the data types appropriate for AWS Glue as listed in this document.

Data type support

This section lists the data types that the DocumentDB connector uses for schema inference, and the data types when AWS Glue metadata is used.

Schema inference data types

The schema inference feature of the DocumentDB connector attempts to infer values as belonging to one of the following data types. The table shows the corresponding data types for Amazon DocumentDB, Java, and Apache Arrow.

Apache Arrow	Java or DocDB
VARCHAR	String
INT	Integer
BIGINT	Long
BIT	Boolean
FLOAT4	Float
FLOAT8	Double
TIMESTAMPSEC	Date
VARCHAR	ObjectId
LIST	List
STRUCT	Document

AWS Glue data types

If you use AWS Glue for supplemental metadata, you can configure the following data types. The table shows the corresponding data types for AWS Glue and Apache Arrow.

AWS Glue	Apache Arrow
int	INT
bigint	BIGINT
double	FLOAT8
float	FLOAT4
boolean	BIT
binary	VARBINARY

AWS Glue	Apache Arrow
string	VARCHAR
List	LIST
Struct	STRUCT

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-docdb.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **AWS Glue Data Catalog** – The DocumentDB connector requires read only access to the AWS Glue Data Catalog to obtain schema information.
- **CloudWatch Logs** – The connector requires access to CloudWatch Logs for storing logs.
- **AWS Secrets Manager read access** – If you choose to store DocumentDB endpoint details in Secrets Manager, you must grant the connector access to those secrets.
- **VPC access** – The connector requires the ability to attach and detach interfaces to your VPC so that it can connect to it and communicate with your DocumentDB instances.

Performance

The Athena Amazon DocumentDB connector does not currently support parallel scans but attempts to push down predicates as part of its DocumentDB queries, and predicates against indexes on your DocumentDB collection result in significantly less data scanned.

The Lambda function performs projection pushdown to decrease the data scanned by the query. However, selecting a subset of columns sometimes results in a longer query execution runtime. `LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect `SELECT` queries with a `LIMIT` clause to scan at least 16 MB of data.

Passthrough queries

The Athena Amazon DocumentDB connector supports passthrough queries and is NoSQL based. For information about querying Amazon DocumentDB, see Querying in the *Amazon DocumentDB Developer Guide*.

To use passthrough queries with Amazon DocumentDB, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

The following example queries the `example` database within the TPCDS collection, filtering on all books with the title *Bill of Rights*.

```
SELECT * FROM TABLE(
        system.query(
            database => 'example',
            collection => 'tpcds',
            filter => '{title: "Bill of Rights"}'
        ))
```

Additional resources

- For an article on using Amazon Athena Federated Query to connect a MongoDB database to QuickSight to build dashboards and visualizations, see Visualize MongoDB data from QuickSight using Amazon Athena Federated Query in the *AWS Big Data Blog*.
- For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena DynamoDB connector

The Amazon Athena DynamoDB connector enables Amazon Athena to communicate with DynamoDB so that you can query your tables with SQL. Write operations like INSERT INTO are not supported.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

If you migrate your DynamoDB connections to Glue Catalog and Lake Formation, only the lowercase table and column names will be recognized.

Parameters

Use the parameters in this section to configure the DynamoDB connector.

Glue connections (recommended)

We recommended that you configure a DynamoDB connector by using a Glue connections object. To do this, set the `glue_connection` environment variable of the DynamoDB connector Lambda to the name of the Glue connection to use.

Glue connections properties

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type DYNAMODB
```

Lambda environment properties

glue_connection – Specifies the name of the Glue connection associated with the federated connector.

Legacy connections

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Lambda environment properties

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **disable_glue** – (Optional) If present and set to true, the connector does not attempt to retrieve supplemental metadata from AWS Glue.
- **glue_catalog** – (Optional) Use this option to specify a cross-account AWS Glue catalog. By default, the connector attempts to get metadata from its own AWS Glue account.

- **disable_projection_and_casing** – (Optional) Disables projection and casing. Use if you want to query DynamoDB tables that have casing in their column names and you do not want to specify a `columnMapping` property on your AWS Glue table.

  The `disable_projection_and_casing` parameter uses the following values to specify the behavior of casing and column mapping:

  - **auto** – Disables projection and casing when a previously unsupported type is detected and column name mapping is not set on the table. This is the default setting.
  - **always** – Disables projection and casing unconditionally. This is useful when you have casing in your DynamoDB column names but do not want to specify any column name mapping.

  When using the `disable_projection_and_casing` parameter, keep in mind the following points:

  - Use of the parameter can result in higher bandwidth usage. Additionally, if your Lambda function is not in the same AWS Region as your data source, you will incur higher standard AWS cross-region transfer costs as a result of the higher bandwidth usage. For more information about cross-region transfer costs, see [AWS Data Transfer Charges for Server and Serverless Architectures] in the AWS Partner Network Blog.
  - Because a larger number of bytes is transferred and because the larger number of bytes requires a higher deserialization time, overall latency can increase.

Setting up databases and tables in AWS Glue

Because the connector's built-in schema inference capability is limited, you might want to use AWS Glue for metadata. To do this, you must have a database and table in AWS Glue. To enable them for use with DynamoDB, you must edit their properties.

To edit database properties in the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the navigation pane, expand **Data Catalog**, and then choose **Databases**.

   On the **Databases** page, you can edit an existing database, or choose **Add database** to create one.
3. In the list of databases, choose the link for the database that you want to edit.

4. Choose **Edit**.
5. On the **Update a database** page, under **Database settings**, for **Location**, add the string **`dynamo-db-flag`**. This keyword indicates that the database contains tables that the Athena DynamoDB connector is using for supplemental metadata and is required for AWS Glue databases other than `default`. The `dynamo-db-flag` property is useful for filtering out databases in accounts with many databases.
6. Choose **Update Database**.

To edit table properties in the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the navigation pane, expand **Data Catalog**, and then choose **Tables**.
3. On the **Tables** page, in the list of tables, choose the linked name of the table that you want to edit.
4. Choose **Actions**, **Edit table**.
5. On the **Edit table** page, in the **Table properties** section, add the following table properties as required. If you use the AWS Glue DynamoDB crawler, these properties are automatically set.

   - **dynamodb** – String that indicates to the Athena DynamoDB connector that the table can be used for supplemental metadata. Enter the `dynamodb` string in the table properties under a field called **classification** (exact match).

     > **Note**
     >
     > The **Set table properties** page that is part of the table creation process in the AWS Glue console has a **Data format** section with a **Classification** field. You cannot enter or choose `dynamodb` here. Instead, after you create your table, follow the steps to edit the table and to enter `classification` and `dynamodb` as a key-value pair in the **Table properties** section.

   - **sourceTable** – Optional table property that defines the source table name in DynamoDB. Use this if AWS Glue table naming rules prevent you from creating a AWS Glue table with the same name as your DynamoDB table. For example, capital letters are not permitted in AWS Glue table names, but they are permitted in DynamoDB table names.

- **columnMapping** – Optional table property that defines column name mappings. Use this if AWS Glue column naming rules prevent you from creating a AWS Glue table with the same column names as your DynamoDB table. For example, capital letters are not permitted in AWS Glue column names but are permitted in DynamoDB column names. The property value is expected to be in the format col1=Col1,col2=Col2. Note that column mapping applies only to top level column names and not to nested fields.
- **defaultTimeZone** – Optional table property that is applied to `date` or `datetime` values that do not have an explicit time zone. Setting this value is a good practice to avoid discrepancies between the data source default time zone and the Athena session time zone.
- **datetimeFormatMapping** – Optional table property that specifies the `date` or `datetime` format to use when parsing data from a column of the AWS Glue `date` or `timestamp` data type. If this property is not specified, the connector attempts to infer an ISO-8601 format. If the connector cannot infer the `date` or `datetime` format or parse the raw string, then the value is omitted from the result.

  The `datetimeFormatMapping` value should be in the format `col1=someformat1,col2=someformat2`. Following are some example formats:

  ```
  yyyyMMdd'T'HHmmss
  ddMMyyyy'T'HH:mm:ss
  ```

  If your column has `date` or `datetime` values without a time zone and you want to use the column in the `WHERE` clause, set the `datetimeFormatMapping` property for the column.

6. If you define your columns manually, make sure that you use the appropriate data types. If you used a crawler, validate the columns and types that the crawler discovered.
7. Choose **Save**.

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-dynamodb.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.

- **AWS Glue Data Catalog –** The DynamoDB connector requires read only access to the AWS Glue Data Catalog to obtain schema information.
- **CloudWatch Logs –** The connector requires access to CloudWatch Logs for storing logs.
- **DynamoDB read access –** The connector uses the `DescribeTable`, `ListSchemas`, `ListTables`, `Query`, and `Scan` API operations.

Performance

The Athena DynamoDB connector supports parallel scans and attempts to push down predicates as part of its DynamoDB queries. A hash key predicate with `X` distinct values results in `X` query calls to DynamoDB. All other predicate scenarios result in `Y` number of scan calls, where `Y` is heuristically determined based on the size of your table and its provisioned throughput. However, selecting a subset of columns sometimes results in a longer query execution runtime.

`LIMIT` clauses and simple predicates are pushed down and can reduce the amount of data scanned and will lead to decreased query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. For enhanced functionality, and to reduce the amount of data scanned, the Athena DynamoDB connector can combine these expressions and push them directly to DynamoDB.

The following Athena DynamoDB connector operators support predicate pushdown:

- **Boolean:** AND
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10 and col_b < 10
LIMIT 10
```

For an article on using predicate pushdown to improve performance in federated queries, including DynamoDB, see [Improve federated queries with predicate pushdown in Amazon Athena]() in the *AWS Big Data Blog*.

Passthrough queries

The DynamoDB connector supports [passthrough queries]() and uses PartiQL syntax. The DynamoDB [GetItem]() API operation is not supported. For information about querying DynamoDB using PartiQL, see [PartiQL select statements for DynamoDB]() in the *Amazon DynamoDB Developer Guide*.

To use passthrough queries with DynamoDB, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query_string'
        ))
```

The following DynamoDB passthrough query example uses PartiQL to return a list of Fire TV Stick devices that have a `DateWatched` property later than 12/24/22.

```
SELECT * FROM TABLE(
        system.query(
           query => 'SELECT Devices
                       FROM WatchList
                       WHERE Devices.FireStick.DateWatched[0] > '12/24/22''
        ))
```

Troubleshooting

Multiple filters on a sort key column

Error message: KeyConditionExpressions must only contain one condition per key

Cause: This issue can occur in Athena engine version 3 in queries that have both a lower and upper bounded filter on a DynamoDB sort key column. Because DynamoDB does not support more than one filter condition on a sort key, an error is thrown when the connector attempts to push down a query that has both conditions applied.

Solution: Update the connector to version 2023.11.1 or later. For instructions on updating a connector, see Update a data source connector.

Costs

The costs for use of the connector depends on the underlying AWS resources that are used. Because queries that use scans can consume a large number of read capacity units (RCUs), consider the information for Amazon DynamoDB pricing carefully.

Additional resources

- For an introduction to using the Amazon Athena DynamoDB connector, see Access, query, and join Amazon DynamoDB tables using Athena in the *AWS Prescriptive Guidance Patterns* guide.
- For an article on how to use the Athena DynamoDB connector to query data in DynamoDB with SQL and visualize insights in QuickSight, see the *AWS Big Data Blog* post Visualize Amazon DynamoDB insights in QuickSight using the Amazon Athena DynamoDB connector and AWS Glue.
- For an article on using the Amazon Athena DynamoDB connector with Amazon DynamoDB, Athena, and QuickSight to create a simple governance dashboard, see the *AWS Big Data Blog* post Query cross-account Amazon DynamoDB tables using Amazon Athena Federated Query.
- For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Google BigQuery connector

The Amazon Athena connector for Google BigQuery enables Amazon Athena to run SQL queries on your Google BigQuery data.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Lambda functions have a maximum timeout value of 15 minutes. Each split executes a query on BigQuery and must finish with enough time to store the results for Athena to read. If the Lambda function times out, the query fails.
- Google BigQuery is case sensitive. The connector attempts to correct the case of dataset names, table names, and project IDs. This is necessary because Athena lower cases all metadata. These corrections make many extra calls to Google BigQuery.
- Binary data types are not supported.
- Because of Google BigQuery concurrency and quota limits, the connector may encounter Google quota limit issues. To avoid these issues, push as many constraints to Google BigQuery as feasible. For information about BigQuery quotas, see Quotas and limits in the Google BigQuery documentation.

Parameters

Use the parameters in this section to configure the Google BigQuery connector.

Glue connections (recommended)

We recommended that you configure a Google BigQuery connector by using a Glue connections object. To do this, set the `glue_connection` environment variable of the Google BigQuery connector Lambda to the name of the Glue connection to use.

Glue connections properties

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type BIGQUERY
```

Lambda environment properties

glue_connection – Specifies the name of the Glue connection associated with the federated connector.

Legacy connections

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Lambda environment properties

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **gcp_project_id** – The project ID (not project name) that contains the datasets that the connector should read from (for example, `semiotic-primer-1234567`).
- **secret_manager_gcp_creds_name** – The name of the secret within AWS Secrets Manager that contains your BigQuery credentials in JSON format (for example, `GoogleCloudPlatformCredentials`).

- **big_query_endpoint** – (Optional) The URL of a BigQuery private endpoint. Use this parameter when you want to access BigQuery over a private endpoint.

Splits and views

Because the BigQuery connector uses the BigQuery Storage Read API to query tables, and the BigQuery Storage API does not support views, the connector uses the BigQuery client with a single split for views.

Performance

To query tables, the BigQuery connector uses the BigQuery Storage Read API, which uses an RPC-based protocol that provides fast access to BigQuery managed storage. For more information about the BigQuery Storage Read API, see [Use the BigQuery Storage Read API to read table data](#) in the Google Cloud documentation.

Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The connector is subject to query failures as concurrency increases, and generally is a slow connector.

The Athena Google BigQuery connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, `ORDER BY` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Top N queries

A top `N` query specifies an ordering of the result set and a limit on the number of rows returned. You can use this type of query to determine the top `N` max values or top `N` min values for your datasets. With top `N` pushdown, the connector returns only `N` ordered rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Google BigQuery connector can combine these expressions and push them directly to Google BigQuery for enhanced functionality and to reduce the amount of data scanned.

The following Athena Google BigQuery connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
ORDER BY col_a DESC
LIMIT 10;
```

Passthrough queries

The Google BigQuery connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Google BigQuery, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Google BigQuery. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

The Amazon Athena Google BigQuery connector project is licensed under the Apache-2.0 License.

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Google Cloud Storage connector

The Amazon Athena Google Cloud Storage connector enables Amazon Athena to run queries on Parquet and CSV files stored in a Google Cloud Storage (GCS) bucket. After you group one or more Parquet or CSV files in an unpartitioned or partitioned folder in a GCS bucket, you can organize them in an AWS Glue database table.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

For an article that shows how to use Athena to run queries on Parquet or CSV files in a GCS bucket, see the AWS Big Data Blog post Use Amazon Athena to query data stored in Google Cloud Platform.

Prerequisites

- Set up an AWS Glue database and table that correspond to your bucket and folders in Google Cloud Storage. For the steps, see Setting up databases and tables in AWS Glue later in this document.
- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- Any relevant Lambda limits. For more information, see [Lambda quotas] in the *AWS Lambda Developer Guide*.
- Currently, the connector supports only the VARCHAR type for partition columns (`string` or `varchar` in an AWS Glue table schema). Other partition field types raise errors when you query them in Athena.

Terms

The following terms relate to the GCS connector.

- **Handler** – A Lambda handler that accesses your GCS bucket. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your GCS bucket.
- **Record handler** – A Lambda handler that retrieves data records from your GCS bucket.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your GCS bucket.

Supported file types

The GCS connector supports the Parquet and CSV file types.

> **Note**
>
> Make sure you do not place both CSV and Parquet files in the same GCS bucket or path. Doing so can result in a runtime error when Parquet files are attempted to be read as CSV or vice versa.

Parameters

Use the parameters in this section to configure the GCS connector.

Note

Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **secret_manager_gcp_creds_name** – The name of the secret in AWS Secrets Manager that contains your GCS credentials in JSON format (for example, `GoogleCloudPlatformCredentials`).

Setting up databases and tables in AWS Glue

Because the built-in schema inference capability of the GCS connector is limited, we recommend that you use AWS Glue for your metadata. The following procedures show how to create a database and table in AWS Glue that you can access from Athena.

Creating a database in AWS Glue

You can use the AWS Glue console to create a database for use with the GCS connector.

To create a database in AWS Glue

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. From the navigation pane, choose **Databases**.
3. Choose **Add database**.
4. For **Name**, enter a name for the database that you want to use with the GCS connector.
5. For **Location**, specify `google-cloud-storage-flag`. This location tells the GCS connector that the AWS Glue database contains tables for GCS data to be queried in Athena. The connector recognizes databases in Athena that have this flag and ignores databases that do not.
6. Choose **Create database**.

Creating a table in AWS Glue

Now you can create a table for the database. When you create an AWS Glue table to use with the GCS connector, you must specify additional metadata.

To create a table in the AWS Glue console

1. In the AWS Glue console, from the navigation pane, choose **Tables**.
2. On the **Tables** page, choose **Add table**.
3. On the **Set table properties** page, enter the following information.
   - **Name** – A unique name for the table.
   - **Database** – Choose the AWS Glue database that you created for the GCS connector.
   - **Include path** – In the **Data store** section, for **Include path**, enter the URI location for GCS prefixed by `gs://` (for example, `gs://`*`gcs_table`*`/`*`data`*`/`). If you have one or more partition folders, don't include them in the path.

> **Note**
>
> When you enter the non `s3://` table path, the AWS Glue console shows an error. You can ignore this error. The table will be created successfully.

   - **Data format** – For **Classification**, select **CSV** or **Parquet**.

4. Choose **Next.**
5. On the **Choose or define schema** page, defining a table schema is highly recommended, but not mandatory. If you do not define a schema, the GCS connector attempts to infer a schema for you.

   Do one of the following:

   - If you want the GCS connector to attempt to infer a schema for you, choose **Next**, and then choose **Create**.
   - To define a schema yourself, follow the steps in the next section.

Defining a table schema in AWS Glue

Defining a table schema in AWS Glue requires more steps but gives you greater control over the table creation process.

To define a schema for your table in AWS Glue

1. On the **Choose or define schema** page, choose **Add**.
2. Use the **Add schema entry** dialog box to provide a column name and data type.
3. To designate the column as a partition column, select the **Set as partition key** option.
4. Choose **Save** to save the column.
5. Choose **Add** to add another column.
6. When you are finished adding columns, choose **Next**.
7. On the **Review and create** page, review the table, and then choose **Create**.
8. If your schema contains partition information, follow the steps in the next section to add a partition pattern to the table's properties in AWS Glue.

Adding a partition pattern to table properties in AWS Glue

If your GCS buckets have partitions, you must add the partition pattern to the properties of the table in AWS Glue.

To add partition information to table properties AWS Glue

1. On the details page for the table that you created in AWS Glue, choose **Actions**, **Edit table**.
2. On the **Edit table** page, scroll down to the **Table properties** section.
3. Choose **Add** to add a partition key.
4. For **Key**, enter **`partition.pattern`**. This key defines the folder path pattern.
5. For **Value**, enter a folder path pattern like **`StateName=${statename}/ZipCode=${zipcode}/`**, where **`statename`** and **`zipcode`** enclosed by **`${}`** are partition column names. The GCS connector supports both Hive and non-Hive partition schemes.
6. When you are finished, choose **Save**.
7. To view the table properties that you just created, choose the **Advanced properties** tab.

At this point, you can navigate to the Athena console. The database and table that you created in AWS Glue are available for querying in Athena.

Data type support

The following tables show the supported data types for CSV and for Parquet.

CSV

Nature of data	Inferred Data Type
Data looks like a number	BIGINT
Data looks like a string	VARCHAR
Data looks like a floating point (float, double, or decimal)	DOUBLE
Data looks like a Date	Timestamp
Data that contains true/false values	BOOL

Parquet

PARQUET	Athena (Arrow)
BINARY	VARCHAR
BOOLEAN	BOOL
DOUBLE	DOUBLE
ENUM	VARCHAR
FIXED_LEN_BYTE_ARRAY	DECIMAL
FLOAT	FLOAT (32-bit)
INT32	1. INT32 2. DATEDAY (when the Parquet column logical type is DATE)
INT64	1. INT64 2. TIMESTAMP (when the Parquet column logical type is TIMESTAMP)
INT96	Timestamp
MAP	MAP
STRUCT	STRUCT
LIST	LIST

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-gcs.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access –** The connector requires write access to a location in Amazon S3 in order to spill results from large queries.

- **Athena GetQueryExecution –** The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **AWS Glue Data Catalog –** The GCS connector requires read only access to the AWS Glue Data Catalog to obtain schema information.
- **CloudWatch Logs –** The connector requires access to CloudWatch Logs for storing logs.

Performance

When the table schema contains partition fields and the `partition.pattern` table property is configured correctly, you can include the partition field in the `WHERE` clause of your queries. For such queries, the GCS connector uses the partition columns to refine the GCS folder path and avoid scanning unneeded files in GCS folders.

For Parquet datasets, selecting a subset of columns results in fewer data being scanned. This usually results in a shorter query execution runtime when column projection is applied.

For CSV datasets, column projection is not supported and does not reduce the amount of data being scanned.

`LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect `SELECT` queries with a `LIMIT` clause to scan at least 16 MB of data. The GCS connector scans more data for larger datasets than for smaller datasets, regardless of the `LIMIT` clause applied. For example, the query `SELECT * LIMIT 10000` scans more data for a larger underlying dataset than a smaller one.

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena HBase connector

The Amazon Athena HBase connector enables Amazon Athena to communicate with your Apache HBase instances so that you can query your HBase data with SQL.

Unlike traditional relational data stores, HBase collections do not have set schema. HBase does not have a metadata store. Each entry in a HBase collection can have different fields and data types.

The HBase connector supports two mechanisms for generating table schema information: basic schema inference and AWS Glue Data Catalog metadata.

Schema inference is the default. This option scans a small number of documents in your collection, forms a union of all fields, and coerces fields that have non overlapping data types. This option works well for collections that have mostly uniform entries.

For collections with a greater variety of data types, the connector supports retrieving metadata from the AWS Glue Data Catalog. If the connector sees an AWS Glue database and table that match your HBase namespace and collection names, it gets its schema information from the corresponding AWS Glue table. When you create your AWS Glue table, we recommend that you make it a superset of all fields that you might want to access from your HBase collection.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the HBase connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **disable_glue** – (Optional) If present and set to true, the connector does not attempt to retrieve supplemental metadata from AWS Glue.
- **glue_catalog** – (Optional) Use this option to specify a cross-account AWS Glue catalog. By default, the connector attempts to get metadata from its own AWS Glue account.
- **default_hbase** – If present, specifies an HBase connection string to use when no catalog-specific environment variable exists.
- **enable_case_insensitive_match** – (Optional) When `true`, performs case insensitive searches against table names in HBase. The default is `false`. Use if your query contains uppercase table names.

Specifying connection strings

You can provide one or more properties that define the HBase connection details for the HBase instances that you use with the connector. To do this, set a Lambda environment variable that corresponds to the catalog name that you want to use in Athena. For example, suppose you want to use the following queries to query two different HBase instances from Athena:

```
SELECT * FROM "hbase_instance_1".database.table
```

```
SELECT * FROM "hbase_instance_2".database.table
```

Before you can use these two SQL statements, you must add two environment variables to your Lambda function: `hbase_instance_1` and `hbase_instance_2`. The value for each should be a HBase connection string in the following format:

```
master_hostname:hbase_port:zookeeper_port
```

Using secrets

You can optionally use AWS Secrets Manager for part or all of the value for your connection string details. To use the Athena Federated Query feature with Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

If you use the syntax `${my_secret}` to put the name of a secret from Secrets Manager in your connection string, the connector replaces the secret name with your user name and password values from Secrets Manager.

For example, suppose you set the Lambda environment variable for `hbase_instance_1` to the following value:

```
${hbase_host_1}:${hbase_master_port_1}:${hbase_zookeeper_port_1}
```

The Athena Query Federation SDK automatically attempts to retrieve a secret named `hbase_instance_1_creds` from Secrets Manager and inject that value in place of `${hbase_instance_1_creds}`. Any part of the connection string that is enclosed by the `${ }` character combination is interpreted as a secret from Secrets Manager. If you specify a secret name that the connector cannot find in Secrets Manager, the connector does not replace the text.

Setting up databases and tables in AWS Glue

The connector's built-in schema inference supports only values that are serialized in HBase as strings (for example, `String.valueOf(int)`). Because the connector's built-in schema inference capability is limited, you might want to use AWS Glue for metadata instead. To enable an AWS Glue table for use with HBase, you must have an AWS Glue database and table with names that match the HBase namespace and table that you want to supply supplemental metadata for. The use of HBase column family naming conventions is optional but not required.

To use an AWS Glue table for supplemental metadata

1. When you edit the table and database in the AWS Glue console, add the following table properties:

   - **hbase-metadata-flag** – This property indicates to the HBase connector that the connector can use the table for supplemental metadata. You can provide any value for `hbase-metadata-flag` as long as the `hbase-metadata-flag` property is present in the list of table properties.
   - **hbase-native-storage-flag** – Use this flag to toggle the two value serialization modes supported by the connector. By default, when this field is not present, the connector assumes all values are stored in HBase as strings. As such it will attempt to parse data types such as `INT`, `BIGINT`, and `DOUBLE` from HBase as strings. If this field is set with any value on the table in AWS Glue, the connector switches to "native" storage mode and attempts to read `INT`, `BIGINT`, `BIT`, and `DOUBLE` as bytes by using the following functions:

     ```
     ByteBuffer.wrap(value).getInt()
     ByteBuffer.wrap(value).getLong()
     ByteBuffer.wrap(value).get()
     ByteBuffer.wrap(value).getDouble()
     ```

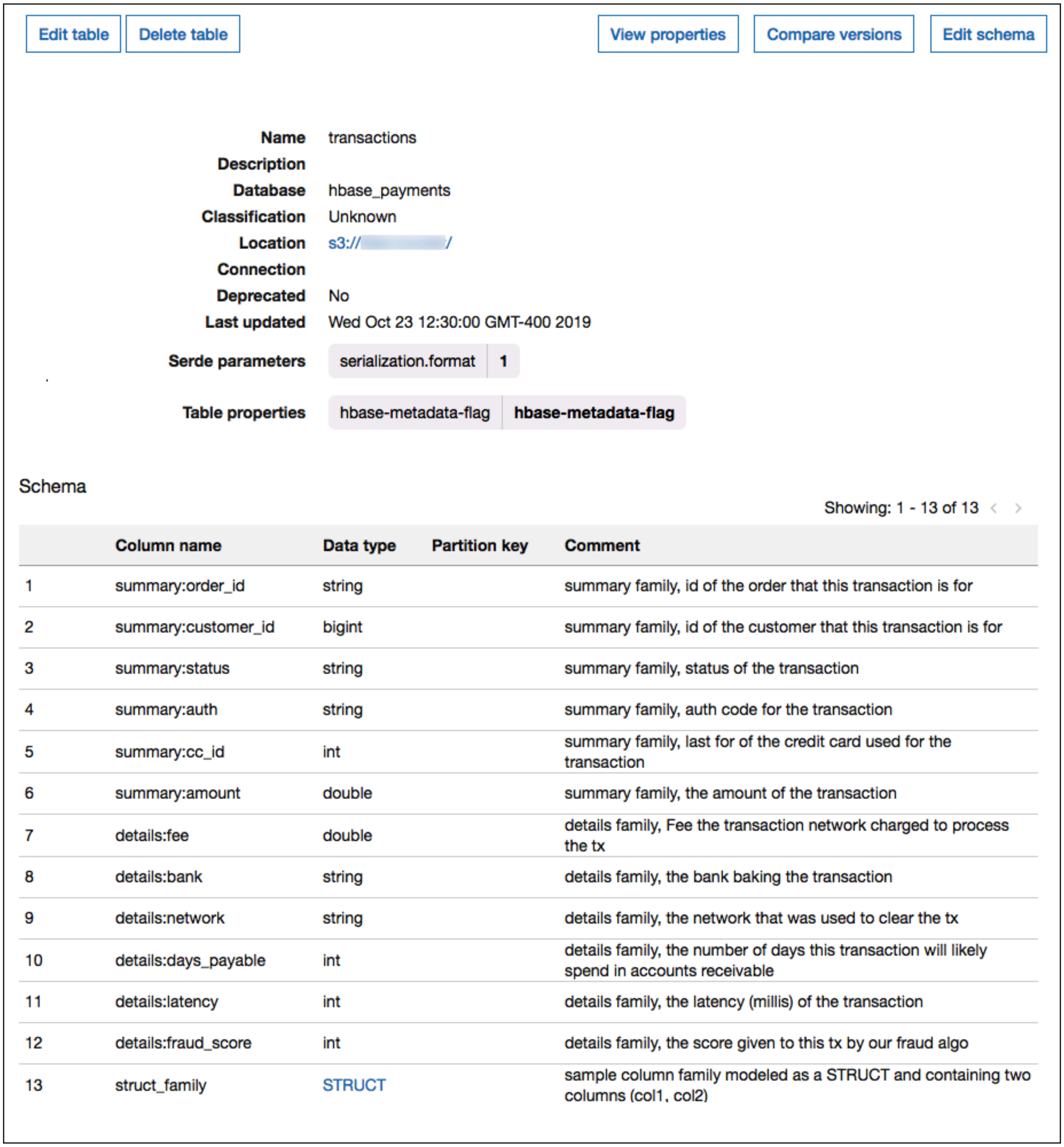
2. Make sure that you use the data types appropriate for AWS Glue as listed in this document.

Modeling column families

The Athena HBase connector supports two ways to model HBase column families: fully qualified (flattened) naming like `family:column`, or using `STRUCT` objects.

In the `STRUCT` model, the name of the `STRUCT` field should match the column family, and children of the `STRUCT` should match the names of the columns of the family. However, because predicate push down and columnar reads are not yet fully supported for complex types like STRUCT, using STRUCT is currently not advised.

The following image shows a table configured in AWS Glue that uses a combination of the two approaches.



Data type support

The connector retrieves all HBase values as the basic byte type. Then, based on how you defined your tables in AWS Glue Data Catalog, it maps the values into one of the Apache Arrow data types in the following table.

AWS Glue data type	Apache Arrow data type
int	INT
bigint	BIGINT
double	FLOAT8
float	FLOAT4
boolean	BIT
binary	VARBINARY
string	VARCHAR

> **Note**
>
> If you do not use AWS Glue to supplement your metadata, the connector's schema inferencing uses only the data types `BIGINT`, `FLOAT8`, and `VARCHAR`.

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-hbase.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access –** The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution –** The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **AWS Glue Data Catalog –** The HBase connector requires read only access to the AWS Glue Data Catalog to obtain schema information.

- **CloudWatch Logs –** The connector requires access to CloudWatch Logs for storing logs.
- **AWS Secrets Manager read access –** If you choose to store HBase endpoint details in Secrets Manager, you must grant the connector access to those secrets.
- **VPC access –** The connector requires the ability to attach and detach interfaces to your VPC so that it can connect to it and communicate with your HBase instances.

Performance

The Athena HBase connector attempts to parallelize queries against your HBase instance by reading each region server in parallel. The Athena HBase connector performs predicate pushdown to decrease the data scanned by the query.

The Lambda function also performs *projection* pushdown to decrease the data scanned by the query. However, selecting a subset of columns sometimes results in a longer query execution runtime. `LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect `SELECT` queries with a `LIMIT` clause to scan at least 16 MB of data.

HBase is prone to query failures and variable query execution times. You might have to retry your queries multiple times for them to succeed. The HBase connector is resilient to throttling due to concurrency.

Passthrough queries

The HBase connector supports passthrough queries and is NoSQL based. For information about querying Apache HBase using filtering, see Filter language in the Apache documentation.

To use passthrough queries with HBase, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

The following example HBase passthrough query filters for employees aged 24 or 30 within the `employee` collection of the `default` database.

```
SELECT * FROM TABLE(
        system.query(
```

```
            DATABASE => 'default',
            COLLECTION => 'employee',
            FILTER => 'SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:30'')' ||
                       ' OR SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:24'')'
       ))
```

License information

The Amazon Athena HBase connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Hortonworks connector

The Amazon Athena connector for Hortonworks enables Amazon Athena to run SQL queries on the Cloudera Hortonworks data platform. The connector transforms your Athena SQL queries to their equivalent HiveQL syntax.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.

Terms

The following terms relate to the Hortonworks Hive connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Hortonworks Hive connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
hive://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`HiveMuxCompositeHandler`
Metadata handler	`HiveMuxMetadataHandler`

Handler	Class
Record handler	HiveMuxRecordHandler

Multiplexing handler parameters

Parameter	Description
$*catalog*_connection_string	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is myhivecatalog , then the environment variable name is myhivecatalog_connection_string .
default	Required. The default connection string. This string is used when the catalog is lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }.

The following example properties are for a Hive MUX Lambda function that supports two database instances: hive1 (the default), and hive2.

Property	Value
default	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog1_connection_string	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog2_connection_string	hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/RDS/hive1host}`.

```
hive://jdbc:hive2://hive1host:10000/default?...&${Test/RDS/hive1host}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
hive://jdbc:hive2://hive1host:10000/default?...&UID=sample2&PWD=sample2&...
```

Currently, the Hortonworks Hive connector recognizes the `UID` and `PWD` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Hortonworks Hive instance.

Handler type	Class
Composite handler	`HiveCompositeHandler`
Metadata handler	`HiveMetadataHandler`
Record handler	`HiveRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Hortonworks Hive instance supported by a Lambda function.

Property	Value
`default`	`hive://jdbc:hive2://hive1host:10000/default?secret=${Test/RDS/hive1host}`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see [PutObject](https://docs.aws.amazon.com/AmazonS3/latest/API/API_PutObject.html) in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC, Hortonworks Hive, and Arrow.

JDBC	Hortonworks Hive	Arrow
Boolean	Boolean	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint
Integer	INT	Int
Long	BIGINT	Bigint
float	float4	Float4
Double	float8	Float8
Date	date	DateDay
Timestamp	timestamp	DateMilli
String	VARCHAR	Varchar

JDBC	Hortonworks Hive	Arrow
Bytes	bytes	Varbinary
BigDecimal	Decimal	Decimal
ARRAY	N/A (see note)	List

Note

Currently, Hortonworks Hive does not support the aggregate types `ARRAY`, `MAP`, `STRUCT`, or `UNIONTYPE`. Columns of aggregate types are treated as `VARCHAR` columns in SQL.

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

Hortonworks Hive supports static partitions. The Athena Hortonworks Hive connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, static partitioning is highly recommended. Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The Hortonworks Hive connector is resilient to throttling due to concurrency.

The Athena Hortonworks Hive connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the WHERE clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Hortonworks Hive connector can combine these expressions and push them directly to Hortonworks Hive for enhanced functionality and to reduce the amount of data scanned.

The following Athena Hortonworks Hive connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The Hortonworks Hive connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Hortonworks Hive, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Hortonworks Hive. The query selects all columns in the customer table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Hortonworks Hive connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Apache Kafka connector

The Amazon Athena connector for Apache Kafka enables Amazon Athena to run SQL queries on your Apache Kafka topics. Use this connector to view Apache Kafka topics as tables and messages as rows in Athena.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Date and timestamp data types in filter conditions must be cast to appropriate data types.
- Date and timestamp data types are not supported for the CSV file type and are treated as varchar values.

- Mapping into nested JSON fields is not supported. The connector maps top-level fields only.
- The connector does not support complex types. Complex types are interpreted as strings.
- To extract or work with complex JSON values, use the JSON-related functions available in Athena. For more information, see Extract JSON data from strings.
- The connector does not support access to Kafka message metadata.

Terms

- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Kafka endpoint** – A text string that establishes a connection to a Kafka instance.

Cluster compatibility

The Kafka connector can be used with the following cluster types.

- **Standalone Kafka** – A direct connection to Kafka (authenticated or unauthenticated).
- **Confluent** – A direct connection to Confluent Kafka. For information about using Athena with Confluent Kafka data, see Visualize Confluent data in QuickSight using Amazon Athena in the *AWS Business Intelligence Blog*.

Connecting to Confluent

Connecting to Confluent requires the following steps:

1. Generate an API key from Confluent.
2. Store the username and password for the Confluent API key into AWS Secrets Manager.
3. Provide the secret name for the `secrets_manager_secret` environment variable in the Kafka connector.
4. Follow the steps in the Setting up the Kafka connector section of this document.

Supported authentication methods

The connector supports the following authentication methods.

- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH
- **Self-managed Kafka and Confluent Platform** – SSL, SASL/SCRAM, SASL/PLAINTEXT, NO_AUTH
- **Self-managed Kafka and Confluent Cloud** – SASL/PLAIN

For more information, see Configuring authentication for the Athena Kafka connector.

Supported input data formats

The connector supports the following input data formats.

- JSON
- CSV
- AVRO
- PROTOBUF (PROTOCOL BUFFERS)

Parameters

Use the parameters in this section to configure the Athena Kafka connector.

- **auth_type** – Specifies the authentication type of the cluster. The connector supports the following types of authentication:
  - **NO_AUTH** – Connect directly to Kafka (for example, to a Kafka cluster deployed over an EC2 instance that does not use authentication).
  - **SASL_SSL_PLAIN** – This method uses the `SASL_SSL` security protocol and the `PLAIN` SASL mechanism. For more information, see SASL configuration in the Apache Kafka documentation.
  - **SASL_PLAINTEXT_PLAIN** – This method uses the `SASL_PLAINTEXT` security protocol and the `PLAIN` SASL mechanism. For more information, see SASL configuration in the Apache Kafka documentation.
  - **SASL_SSL_SCRAM_SHA512** – You can use this authentication type to control access to your Apache Kafka clusters. This method stores the user name and password in AWS Secrets

Manager. The secret must be associated with the Kafka cluster. For more information, see Authentication using SASL/SCRAM in the Apache Kafka documentation.

- **SASL_PLAINTEXT_SCRAM_SHA512** – This method uses the `SASL_PLAINTEXT` security protocol and the `SCRAM_SHA512 SASL` mechanism. This method uses your user name and password stored in AWS Secrets Manager. For more information, see the SASL configuration section of the Apache Kafka documentation.
- **SSL** – SSL authentication uses key store and trust store files to connect with the Apache Kafka cluster. You must generate the trust store and key store files, upload them to an Amazon S3 bucket, and provide the reference to Amazon S3 when you deploy the connector. The key store, trust store, and SSL key are stored in AWS Secrets Manager. Your client must provide the AWS secret key when the connector is deployed. For more information, see Encryption and Authentication using SSL in the Apache Kafka documentation.

For more information, see Configuring authentication for the Athena Kafka connector.

- **certificates_s3_reference** – The Amazon S3 location that contains the certificates (the key store and trust store files).
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **kafka_endpoint** – The endpoint details to provide to Kafka.
- **schema_registry_url** – The URL address for the schema registry (for example, `http://schema-registry.example.org:8081`). Applies to the `AVRO` and `PROTOBUF` data formats. Athena only supports Confluent schema registry.
- **secrets_manager_secret** – The name of the AWS secret in which the credentials are saved.
- **Spill parameters** – Lambda functions temporarily store ("spill") data that do not fit into memory to Amazon S3. All database instances accessed by the same Lambda function spill to the same location. Use the parameters in the following table to specify the spill location.

Parameter	Description
`spill_bucket`	Required. The name of the Amazon S3 bucket where the Lambda function can spill data.

Parameter	Description
`spill_prefix`	Required. The prefix within the spill bucket where the Lambda function can spill data.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

- **Subnet IDs** – One or more subnet IDs that correspond to the subnet that the Lambda function can use to access your data source.
  - **Public Kafka cluster or standard Confluent Cloud cluster** – Associate the connector with a private subnet that has a NAT Gateway.
  - **Confluent Cloud cluster with private connectivity** – Associate the connector with a private subnet that has a route to the Confluent Cloud cluster.
    - For AWS Transit Gateway, the subnets must be in a VPC that is attached to the same transit gateway that Confluent Cloud uses.
    - For VPC Peering, the subnets must be in a VPC that is peered to Confluent Cloud VPC.
    - For AWS PrivateLink, the subnets must be in a VPC that a has route to the VPC endpoints that connect to Confluent Cloud.

Note

If you deploy the connector into a VPC in order to access private resources and also want to connect to a publicly accessible service like Confluent, you must associate the connector with a private subnet that has a NAT Gateway. For more information, see NAT gateways in the Amazon VPC User Guide.

Data type support

The following table shows the corresponding data types supported for Kafka and Apache Arrow.

Kafka	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR
TIMESTAMP	MILLISECOND
DATE	DAY
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8
DOUBLE	FLOAT8

Partitions and splits

Kafka topics are split into partitions. Each partition is ordered. Each message in a partition has an incremental ID called an *offset*. Each Kafka partition is further divided to multiple splits for parallel processing. Data is available for the retention period configured in Kafka clusters.

Best practices

As a best practice, use predicate pushdown when you query Athena, as in the following examples.

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE integercol = 2147483647
```

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

Setting up the Kafka connector

Before you can use the connector, you must set up your Apache Kafka cluster, use the AWS Glue Schema Registry to define your schema, and configure authentication for the connector.

When working with the AWS Glue Schema Registry, note the following points:

- Make sure that the text in **Description** field of the AWS Glue Schema Registry includes the string `{AthenaFederationKafka}`. This marker string is required for AWS Glue Registries that you use with the Amazon Athena Kafka connector.
- For best performance, use only lowercase for your database names and table names. Using mixed casing causes the connector to perform a case insensitive search that is more computationally intensive.

To set up your Apache Kafka environment and AWS Glue Schema Registry

1. Set up your Apache Kafka environment.
2. Upload the Kafka topic description file (that is, its schema) in JSON format to the AWS Glue Schema Registry. For more information, see Integrating with AWS Glue Schema Registry in the AWS Glue Developer Guide.
3. To use the `AVRO` or `PROTOBUF` data format when you define the schema in the AWS Glue Schema Registry:

   - For **Schema name**, enter the Kafka topic name in the same casing as the original.
   - For **Data format**, choose **Apache Avro** or **Protocol Buffers**.

   For example schemas, see the following section.

Schema examples for the AWS Glue Schema Registry

Use the format of the examples in this section when you upload your schema to the AWS Glue Schema Registry.

JSON type schema example

In the following example, the schema to be created in the AWS Glue Schema Registry specifies `json` as the value for `dataFormat` and uses `datatypejson` for `topicName`.

> **Note**
>
> The value for `topicName` should use the same casing as the topic name in Kafka.

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
        "name": "tinyintcol",
```

```
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
}
```

CSV type schema example

In the following example, the schema to be created in the AWS Glue Schema Registry specifies `csv` as the value for `dataFormat` and uses `datatypecsvbulk` for `topicName`. The value for `topicName` should use the same casing as the topic name in Kafka.

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
        "type": "BOOLEAN",
```

```
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

AVRO type schema example

The following example is used to create an AVRO-based schema in the AWS Glue Schema Registry. When you define the schema in the AWS Glue Schema Registry, for **Schema name**, you enter the Kafka topic name in the same casing as the original, and for **Data format**, you choose **Apache Avro**. Because you specify this information directly in the registry, the `dataformat`and `topicName` fields are not required.

```
{
    "type": "record",
    "name": "avrotest",
    "namespace": "example.com",
```

```
    "fields": [{
            "name": "id",
            "type": "int"
        },
        {
            "name": "name",
            "type": "string"
        }
    ]
}
```

PROTOBUF type schema example

The following example is used to create an PROTOBUF-based schema in the AWS Glue Schema Registry. When you define the schema in the AWS Glue Schema Registry, for **Schema name**, you enter the Kafka topic name in the same casing as the original, and for **Data format**, you choose **Protocol Buffers**. Because you specify this information directly in the registry, the `dataformat`and `topicName` fields are not required. The first line defines the schema as PROTOBUF.

```
syntax = "proto3";
message protobuftest {
string name = 1;
int64 calories = 2;
string colour = 3;
}
```

For more information about adding a registry and schemas in the AWS Glue Schema Registry, see [Getting started with Schema Registry](#) in the AWS Glue documentation.

Configuring authentication for the Athena Kafka connector

You can use a variety of methods to authenticate to your Apache Kafka cluster, including SSL, SASL/SCRAM, SASL/PLAIN, and SASL/PLAINTEXT.

The following table shows the authentication types for the connector and the security protocol and SASL mechanism for each. For more information, see the [Security](#) section of the Apache Kafka documentation.

auth_type	security.protocol	sasl.mechanism	Cluster type compatibility
SASL_SSL_PLAIN	SASL_SSL	PLAIN	• Self-managed Kafka • Confluent Platform • Confluent Cloud
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN	• Self-managed Kafka • Confluent Platform
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512	• Self-managed Kafka • Confluent Platform
SASL_PLAINTEXT_SCRAM_SHA512	SASL_PLAINTEXT	SCRAM-SHA-512	• Self-managed Kafka • Confluent Platform
SSL	SSL	N/A	• Self-managed Kafka • Confluent Platform

SSL

If the cluster is SSL authenticated, you must generate the trust store and key store files and upload them to the Amazon S3 bucket. You must provide this Amazon S3 reference when you deploy the connector. The key store, trust store, and SSL key are stored in the AWS Secrets Manager. You provide the AWS secret key when you deploy the connector.

For information on creating a secret in Secrets Manager, see Create an AWS Secrets Manager secret.

To use this authentication type, set the environment variables as shown in the following table.

Parameter	Value
auth_type	SSL
certificates_s3_reference	The Amazon S3 location that contains the certificates.

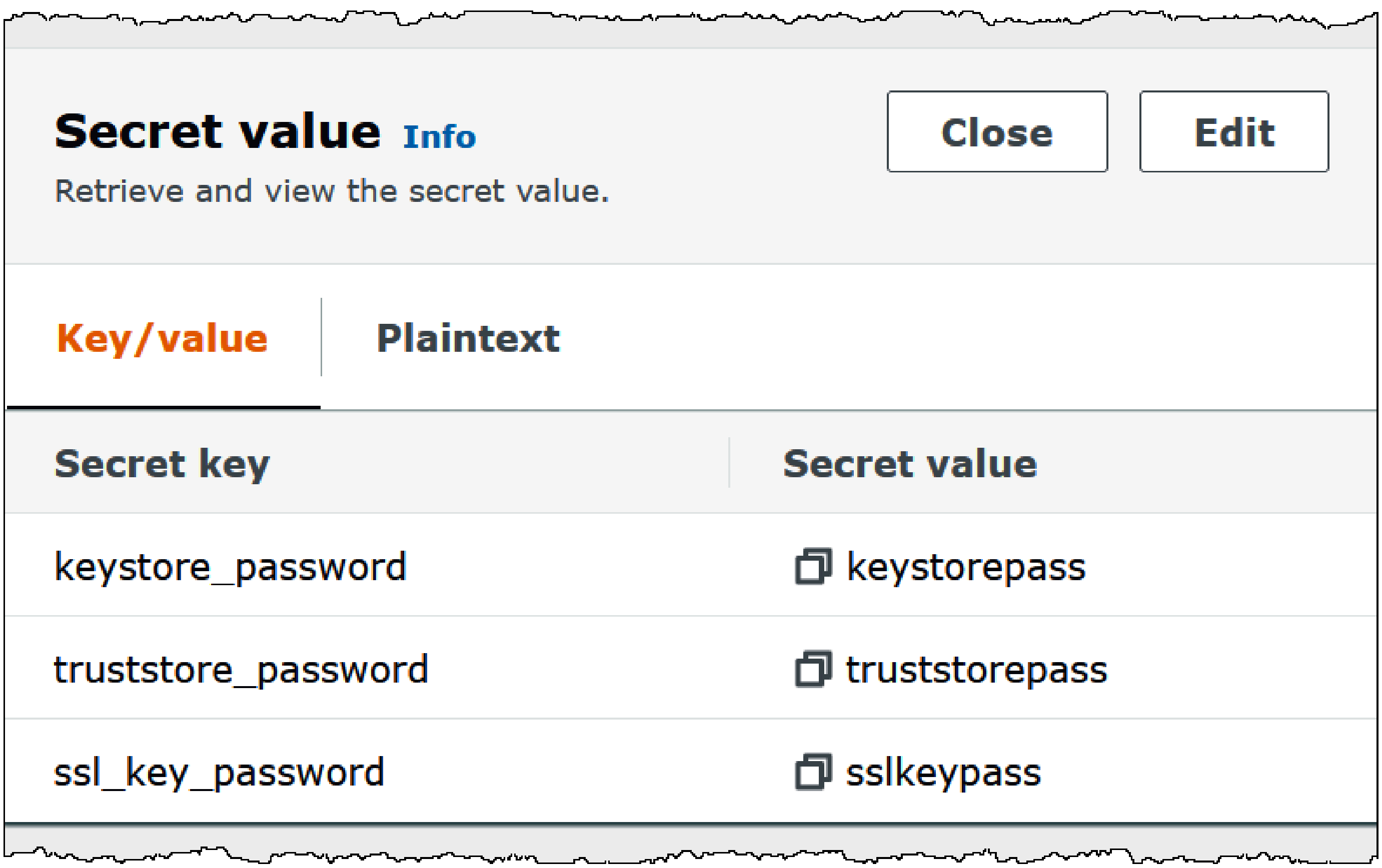
Parameter	Value
`secrets_manager_secret`	The name of your AWS secret key.

After you create a secret in Secrets Manager, you can view it in the Secrets Manager console.

To view your secret in Secrets Manager

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the navigation pane, choose **Secrets**.
3. On the **Secrets** page, choose the link to your secret.
4. On the details page for your secret, choose **Retrieve secret value**.

   The following image shows an example secret with three key/value pairs: `keystore_password`, `truststore_password`, and `ssl_key_password`.



For more information about using SSL with Kafka, see Encryption and Authentication using SSL in the Apache Kafka documentation.

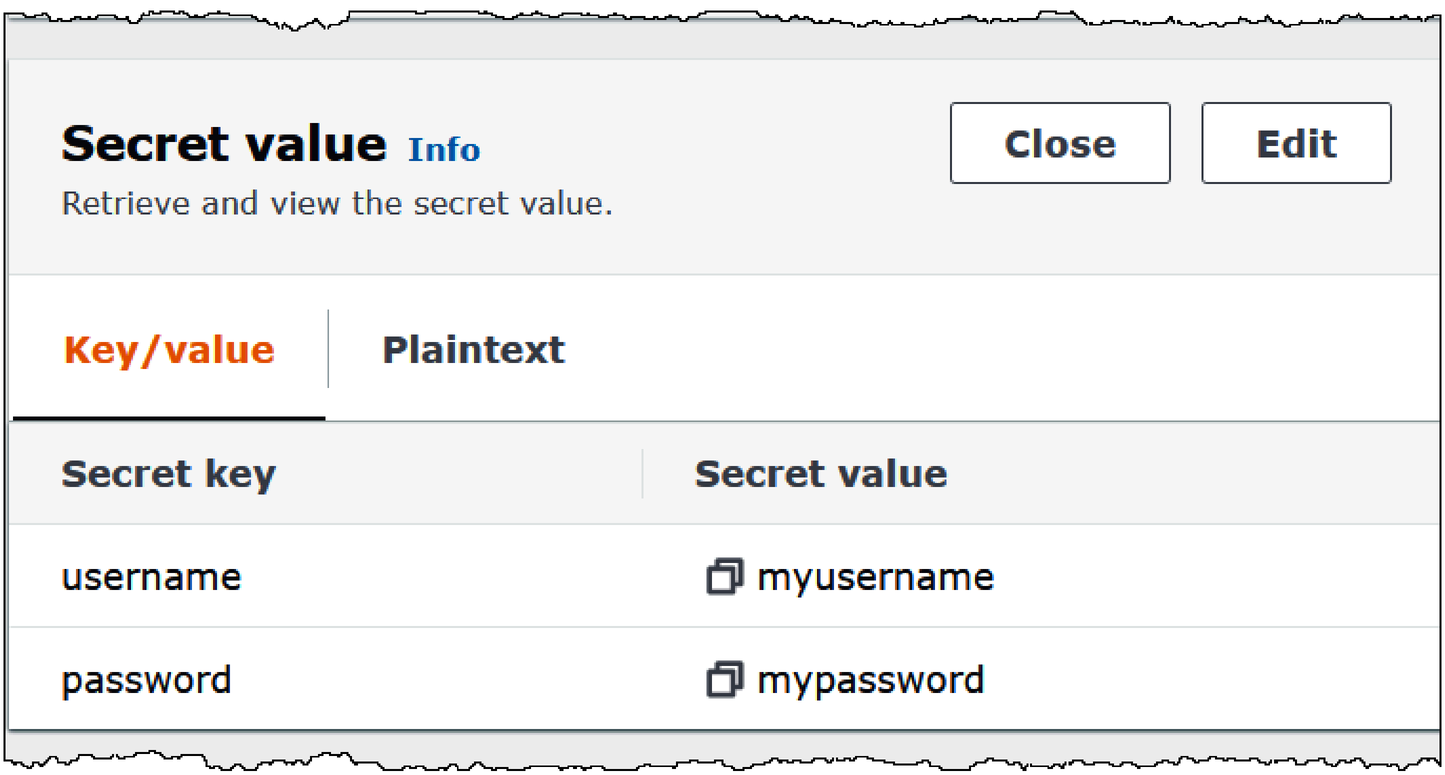
SASL/SCRAM

If your cluster uses SCRAM authentication, provide the Secrets Manager key that is associated with the cluster when you deploy the connector. The user's AWS credentials (secret key and access key) are used to authenticate with the cluster.

Set the environment variables as shown in the following table.

Parameter	Value
auth_type	SASL_SSL_SCRAM_SHA512
secrets_manager_secret	The name of your AWS secret key.

The following image shows an example secret in the Secrets Manager console with two key/value pairs: one for `username`, and one for `password`.



For more information about using SASL/SCRAM with Kafka, see Authentication using SASL/SCRAM in the Apache Kafka documentation.

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena MSK connector

The Amazon Athena connector for Amazon MSK enables Amazon Athena to run SQL queries on your Apache Kafka topics. Use this connector to view Apache Kafka topics as tables and messages as rows in Athena. For additional information, see Analyze real-time streaming data in Amazon MSK with Amazon Athena in the AWS Big Data Blog.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Date and timestamp data types in filter conditions must be cast to appropriate data types.
- Date and timestamp data types are not supported for the CSV file type and are treated as varchar values.
- Mapping into nested JSON fields is not supported. The connector maps top-level fields only.
- The connector does not support complex types. Complex types are interpreted as strings.
- To extract or work with complex JSON values, use the JSON-related functions available in Athena. For more information, see Extract JSON data from strings.
- The connector does not support access to Kafka message metadata.

Terms

- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Kafka endpoint** – A text string that establishes a connection to a Kafka instance.

Cluster compatibility

The MSK connector can be used with the following cluster types.

- **MSK Provisioned cluster** – You manually specify, monitor, and scale cluster capacity.
- **MSK Serverless cluster** – Provides on-demand capacity that scales automatically as application I/O scales.
- **Standalone Kafka** – A direct connection to Kafka (authenticated or unauthenticated).

Supported authentication methods

The connector supports the following authentication methods.

- SASL/IAM
- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH

  For more information, see Configuring authentication for the Athena MSK connector.

Supported input data formats

The connector supports the following input data formats.

- JSON
- CSV

Parameters

Use the parameters in this section to configure the Athena MSK connector.

- **auth_type** – Specifies the authentication type of the cluster. The connector supports the following types of authentication:
  - **NO_AUTH** – Connect directly to Kafka with no authentication (for example, to a Kafka cluster deployed over an EC2 instance that does not use authentication).
  - **SASL_SSL_PLAIN** – This method uses the `SASL_SSL` security protocol and the `PLAIN` SASL mechanism.
  - **SASL_PLAINTEXT_PLAIN** – This method uses the `SASL_PLAINTEXT` security protocol and the `PLAIN` SASL mechanism.

    Note

    The `SASL_SSL_PLAIN` and `SASL_PLAINTEXT_PLAIN` authentication types are supported by Apache Kafka but not by Amazon MSK.
  - **SASL_SSL_AWS_MSK_IAM** – IAM access control for Amazon MSK enables you to handle both authentication and authorization for your MSK cluster. Your user's AWS credentials (secret key and access key) are used to connect with the cluster. For more information, see IAM access control in the Amazon Managed Streaming for Apache Kafka Developer Guide.
  - **SASL_SSL_SCRAM_SHA512** – You can use this authentication type to control access to your Amazon MSK clusters. This method stores the user name and password on AWS Secrets Manager. The secret must be associated with the Amazon MSK cluster. For more information, see Setting up SASL/SCRAM authentication for an Amazon MSK cluster in the Amazon Managed Streaming for Apache Kafka Developer Guide.
  - **SSL** – SSL authentication uses key store and trust store files to connect with the Amazon MSK cluster. You must generate the trust store and key store files, upload them to an Amazon S3 bucket, and provide the reference to Amazon S3 when you deploy the connector. The key store, trust store, and SSL key are stored in AWS Secrets Manager. Your client must provide the AWS secret key when the connector is deployed. For more information, see Mutual TLS authentication in the Amazon Managed Streaming for Apache Kafka Developer Guide.

  For more information, see Configuring authentication for the Athena MSK connector.
- **certificates_s3_reference** – The Amazon S3 location that contains the certificates (the key store and trust store files).

- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **kafka_endpoint** – The endpoint details to provide to Kafka. For example, for an Amazon MSK cluster, you provide a bootstrap URL for the cluster.
- **secrets_manager_secret** – The name of the AWS secret in which the credentials are saved. This parameter is not required for IAM authentication.
- **Spill parameters** – Lambda functions temporarily store ("spill") data that do not fit into memory to Amazon S3. All database instances accessed by the same Lambda function spill to the same location. Use the parameters in the following table to specify the spill location.

Parameter	Description
`spill_bucket`	Required. The name of the Amazon S3 bucket where the Lambda function can spill data.
`spill_prefix`	Required. The prefix within the spill bucket where the Lambda function can spill data.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types supported for Kafka and Apache Arrow.

Kafka	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR

Kafka	Arrow
TIMESTAMP	MILLISECOND
DATE	DAY
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8
DOUBLE	FLOAT8

Partitions and splits

Kafka topics are split into partitions. Each partition is ordered. Each message in a partition has an incremental ID called an *offset*. Each Kafka partition is further divided to multiple splits for parallel processing. Data is available for the retention period configured in Kafka clusters.

Best practices

As a best practice, use predicate pushdown when you query Athena, as in the following examples.

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE integercol = 2147483647
```

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

Setting up the MSK connector

Before you can use the connector, you must set up your Amazon MSK cluster, use the [AWS Glue Schema Registry] to define your schema, and configure authentication for the connector.

> **Note**
>
> If you deploy the connector into a VPC in order to access private resources and also want to connect to a publicly accessible service like Confluent, you must associate the connector with a private subnet that has a NAT Gateway. For more information, see NAT gateways in the Amazon VPC User Guide.

When working with the AWS Glue Schema Registry, note the following points:

- Make sure that the text in **Description** field of the AWS Glue Schema Registry includes the string `{AthenaFederationMSK}`. This marker string is required for AWS Glue Registries that you use with the Amazon Athena MSK connector.
- For best performance, use only lowercase for your database names and table names. Using mixed casing causes the connector to perform a case insensitive search that is more computationally intensive.

To set up your Amazon MSK environment and AWS Glue Schema Registry

1. Set up your Amazon MSK environment. For information and steps, see Setting up Amazon MSK and Getting started using Amazon MSK in the Amazon Managed Streaming for Apache Kafka Developer Guide.
2. Upload the Kafka topic description file (that is, its schema) in JSON format to the AWS Glue Schema Registry. For more information, see Integrating with AWS Glue Schema Registry in the AWS Glue Developer Guide. For example schemas, see the following section.

Schema examples for the AWS Glue Schema Registry

Use the format of the examples in this section when you upload your schema to the AWS Glue Schema Registry.

JSON type schema example

In the following example, the schema to be created in the AWS Glue Schema Registry specifies `json` as the value for `dataFormat` and uses `datatypejson` for `topicName`.

> **Note**
>
> The value for `topicName` should use the same casing as the topic name in Kafka.

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
        "name": "tinyintcol",
```

```
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
}
```

CSV type schema example

In the following example, the schema to be created in the AWS Glue Schema Registry specifies `csv` as the value for `dataFormat` and uses `datatypecsvbulk` for `topicName`. The value for `topicName` should use the same casing as the topic name in Kafka.

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
        "type": "BOOLEAN",
```

```
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

Configuring authentication for the Athena MSK connector

You can use a variety of methods to authenticate to your Amazon MSK cluster, including IAM, SSL, SCRAM, and standalone Kafka.

The following table shows the authentication types for the connector and the security protocol and SASL mechanism for each. For more information, see Authentication and authorization for Apache Kafka APIs in the Amazon Managed Streaming for Apache Kafka Developer Guide.

auth_type	security.protocol	sasl.mechanism
SASL_SSL_PLAIN	SASL_SSL	PLAIN
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN
SASL_SSL_AWS_MSK_IAM	SASL_SSL	AWS_MSK_IAM
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512
SSL	SSL	N/A

Note

The SASL_SSL_PLAIN and SASL_PLAINTEXT_PLAIN authentication types are supported by Apache Kafka but not by Amazon MSK.

SASL/IAM

If the cluster uses IAM authentication, you must configure the IAM policy for the user when you set up the cluster. For more information, see IAM access control in the Amazon Managed Streaming for Apache Kafka Developer Guide.

To use this authentication type, set the `auth_type` Lambda environment variable for the connector to `SASL_SSL_AWS_MSK_IAM`.

SSL

If the cluster is SSL authenticated, you must generate the trust store and key store files and upload them to the Amazon S3 bucket. You must provide this Amazon S3 reference when you deploy the connector. The key store, trust store, and SSL key are stored in the AWS Secrets Manager. You provide the AWS secret key when you deploy the connector.

For information on creating a secret in Secrets Manager, see Create an AWS Secrets Manager secret.

To use this authentication type, set the environment variables as shown in the following table.

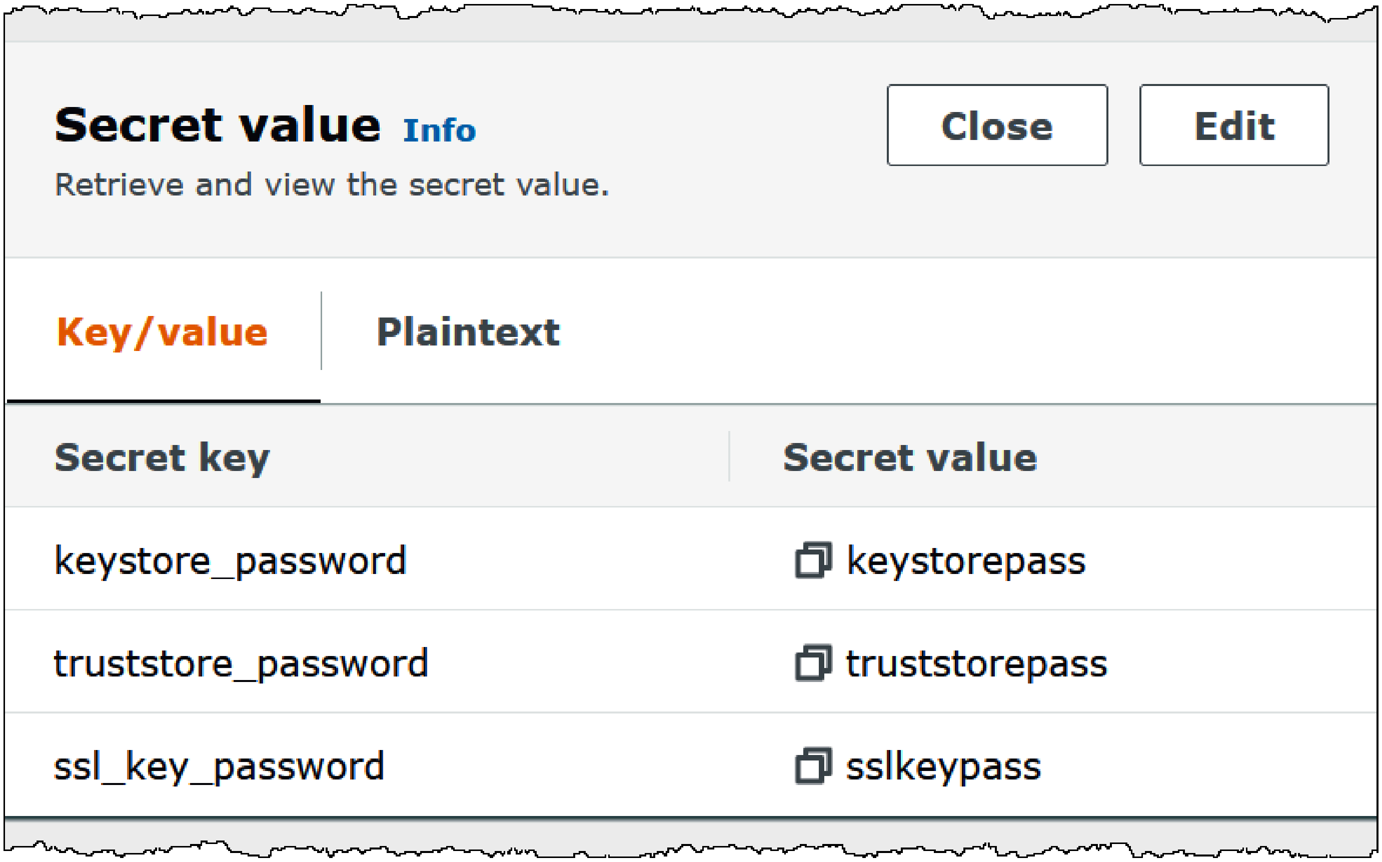
Parameter	Value
auth_type	SSL
certificates_s3_reference	The Amazon S3 location that contains the certificates.
secrets_manager_secret	The name of your AWS secret key.

After you create a secret in Secrets Manager, you can view it in the Secrets Manager console.

To view your secret in Secrets Manager

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the navigation pane, choose **Secrets**.
3. On the **Secrets** page, choose the link to your secret.
4. On the details page for your secret, choose **Retrieve secret value**.

   The following image shows an example secret with three key/value pairs: `keystore_password`, `truststore_password`, and `ssl_key_password`.



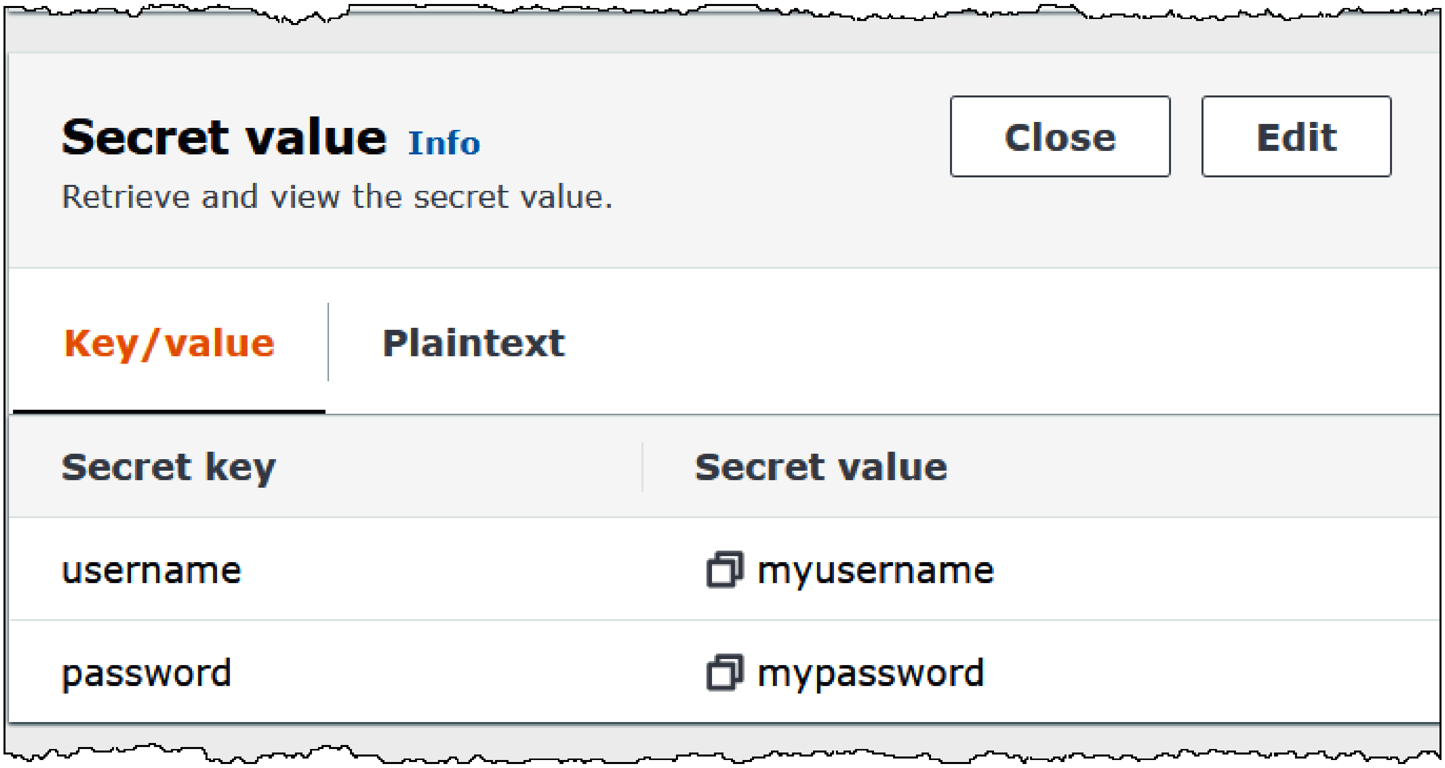
SASL/SCRAM

If your cluster uses SCRAM authentication, provide the Secrets Manager key that is associated with the cluster when you deploy the connector. The user's AWS credentials (secret key and access key) are used to authenticate with the cluster.

Set the environment variables as shown in the following table.

Parameter	Value
`auth_type`	`SASL_SSL_SCRAM_SHA512`
`secrets_manager_secret`	The name of your AWS secret key.

The following image shows an example secret in the Secrets Manager console with two key/value pairs: one for `username`, and one for `password`.



License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena MySQL connector

The Amazon Athena Lambda MySQL connector enables Amazon Athena to access MySQL databases.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Because Athena converts queries to lower case, MySQL table names must be in lower case. For example, Athena queries against a table named `myTable` will fail.
- If you migrate your MySQL connections to Glue Catalog and Lake Formation, only the lowercase table and column names will be recognized.

Terms

The following terms relate to the MySQL connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.

- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the MySQL connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

Glue connections (recommended)

We recommended that you configure a MySQL connector by using a Glue connections object.

To do this, set the `glue_connection` environment variable of the MySQL connector Lambda to the name of the Glue connection to use.

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type MYSQL
```

Legacy connections

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
mysql://${jdbc_connection_string}
```

Note

If you receive the error java.sql.SQLException: Zero date value prohibited when doing a SELECT query on a MySQL table, add the following parameter to your connection string:

```
zeroDateTimeBehavior=convertToNull
```

For more information, see Error 'Zero date value prohibited' while trying to select from MySQL table on GitHub.com.

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`MySqlMuxCompositeHandler`
Metadata handler	`MySqlMuxMetadataHandler`
Record handler	`MySqlMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mymysqlcatalog`, then the environment variable name is `mymysqlcatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${AWS_LAMBDA_FUNCTION_NAME}`.

The following example properties are for a MySql MUX Lambda function that supports two database instances: `mysql1` (the default), and `mysql2`.

Property	Value
`default`	`mysql://jdbc:mysql://mysql2.host:3333/default?` user=sample2&password=sample2
`mysql_catalog1_connection_string`	`mysql://jdbc:mysql://mysql1.host:3306/default?${Test/RDS/MySql1}`
`mysql_catalog2_connection_string`	`mysql://jdbc:mysql://mysql2.host:3333/default?` user=sample2&password=sample2

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/RDS/MySql1}`.

```
mysql://jdbc:mysql://mysql1.host:3306/default?...&${Test/RDS/MySql1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
mysql://jdbc:mysql://mysql1host:3306/default?...&user=sample2&password=sample2&...
```

Currently, the MySQL connector recognizes the `user` and `password` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single MySQL instance.

Handler type	Class
Composite handler	`MySqlCompositeHandler`
Metadata handler	`MySqlMetadataHandler`
Record handler	`MySqlRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single MySQL instance supported by a Lambda function.

Property	Value
`default`	`mysql://mysql1.host:3306/default?secret=Test/RDS/` MySql1

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see [PutObject]() in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Arrow.

JDBC	Arrow
Boolean	Bit
Integer	Tiny
Short	Smallint
Integer	Int
Long	Bigint
float	Float4
Double	Float8
Date	DateDay
Timestamp	DateMilli
String	Varchar
Bytes	Varbinary
BigDecimal	Decimal
ARRAY	List

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

MySQL supports native partitions. The Athena MySQL connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, native partitioning is highly recommended.

The Athena MySQL connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena MySQL connector can combine these expressions and push them directly to MySQL for enhanced functionality and to reduce the amount of data scanned.

The following Athena MySQL connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

For an article on using predicate pushdown to improve performance in federated queries, including MySQL, see [Improve federated queries with predicate pushdown in Amazon Athena]() in the *AWS Big Data Blog*.

Passthrough queries

The MySQL connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with MySQL, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in MySQL. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the MySQL connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Neptune connector

Amazon Neptune is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets. Neptune's purpose-built, high-performance graph database engine stores billions of relationships optimally and queries graphs with a latency of only milliseconds. For more information, see the Neptune User Guide.

The Amazon Athena Neptune Connector enables Athena to communicate with your Neptune graph database instance, making your Neptune graph data accessible by SQL queries.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

Prerequisites

Using the Neptune connector requires the following three steps.

- Setting up a Neptune cluster
- Setting up an AWS Glue Data Catalog
- Deploying the connector to your AWS account. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector. For additional details specific to deploying the Neptune connector, see Deploy the Amazon Athena Neptune Connector on GitHub.com.

Limitations

Currently, the Neptune Connector has the following limitation.

- Projecting columns, including the primary key (ID), is not supported.

Setting up a Neptune cluster

If you don't have an existing Amazon Neptune cluster and property graph dataset in it that you would like to use, you must set one up.

Make sure you have an internet gateway and NAT gateway in the VPC that hosts your Neptune cluster. The private subnets that the Neptune connector Lambda function uses should have a route to the internet through this NAT Gateway. The Neptune connector Lambda function uses the NAT Gateway to communicate with AWS Glue.

For instructions on setting up a new Neptune cluster and loading it with a sample dataset, see Sample Neptune Cluster Setup on GitHub.com.

Setting up an AWS Glue Data Catalog

Unlike traditional relational data stores, Neptune graph DB nodes and edges do not use a set schema. Each entry can have different fields and data types. However, because the Neptune connector retrieves metadata from the AWS Glue Data Catalog, you must create an AWS Glue database that has tables with the required schema. After you create the AWS Glue database and tables, the connector can populate the list of tables available to query from Athena.

Enabling case insensitive column matching

To resolve column names from your Neptune table with the correct casing even when the column names are all lower cased in AWS Glue, you can configure the Neptune connector for case insensitive matching.

To enable this feature, set the Neptune connector Lambda function environment variable `enable_caseinsensitivematch` to `true`.

Specifying the AWS Glue glabel table parameter for cased table names

Because AWS Glue supports only lowercase table names, it is important to specify the `glabel` AWS Glue table parameter when you create an AWS Glue table for Neptune and your Neptune table name includes casing.

In your AWS Glue table definition, include the `glabel` parameter and set its value to your table name with its original casing. This ensures that the correct casing is preserved when AWS Glue interacts with your Neptune table. The following example sets the value of `glabel` to the table name `Airport`.

```
glabel = Airport
```

Table properties (3)

Key	Value
separatorChar	,
componenttype	vertex
glabel	Airport

For more information on setting up a AWS Glue Data Catalog to work with Neptune, see [Set up AWS Glue Catalog](#) on GitHub.com.

Performance

The Athena Neptune connector performs predicate pushdown to decrease the data scanned by the query. However, predicates using the primary key result in query failure. `LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect `SELECT` queries with a `LIMIT` clause to scan at least 16 MB of data. The Neptune connector is resilient to throttling due to concurrency.

Passthrough queries

The Neptune connector supports passthrough queries. You can use this feature to run Gremlin queries on property graphs and to run SPARQL queries on RDF data.

To create passthrough queries with Neptune, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'database_name',
            COLLECTION => 'collection_name',
            QUERY => 'query_string'
        ))
```

The following example Neptune passthrough query filters for airports with the code `ATL`. The doubled single quotes are for escaping.

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'graph-database',
            COLLECTION => 'airport',
            QUERY => 'g.V().has(''airport'', ''code'', ''ATL'').valueMap()'
        ))
```

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena OpenSearch connector

OpenSearch Service

The Amazon Athena OpenSearch connector enables Amazon Athena to communicate with your OpenSearch instances so that you can use SQL to query your OpenSearch data.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

> **Note**
>
> Due to a known issue, the OpenSearch connector cannot be used with a VPC.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Terms

The following terms relate to the OpenSearch connector.

- **Domain** – A name that this connector associates with the endpoint of your OpenSearch instance. The domain is also used as the database name. For OpenSearch instances defined within the Amazon OpenSearch Service, the domain is auto-discoverable. For other instances, you must provide a mapping between the domain name and endpoint.
- **Index** – A database table defined in your OpenSearch instance.
- **Mapping** – If an index is a database table, then the mapping is its schema (that is, the definitions of its fields and attributes).

  This connector supports both metadata retrieval from the OpenSearch instance and from the AWS Glue Data Catalog. If the connector finds a AWS Glue database and table that match your OpenSearch domain and index names, the connector attempts to use them for schema definition. We recommend that you create your AWS Glue table so that it is a superset of all fields in your OpenSearch index.
- **Document** – A record within a database table.

- **Data stream** – Time based data that is composed of multiple backing indices. For more information, see Data streams in the OpenSearch documentation and Getting started with data streams in the *Amazon OpenSearch Service Developer Guide*.

> **Note**
>
> Because data stream indices are internally created and managed by open search, the connector chooses the schema mapping from the first available index. For this reason, we strongly recommend setting up an AWS Glue table as a supplemental metadata source. For more information, see Setting up databases and tables in AWS Glue.

Parameters

Use the parameters in this section to configure the OpenSearch connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your

Lambda function use stronger encryption keys generated by KMS like a7e63k4b-8loc-40db-a2a1-4d0en2cd8331, you can specify a KMS key ID.

- **disable_spill_encryption** – (Optional) When set to True, disables spill encryption. Defaults to False so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **disable_glue** – (Optional) If present and set to true, the connector does not attempt to retrieve supplemental metadata from AWS Glue.
- **query_timeout_cluster** – The timeout period, in seconds, for cluster health queries used in the generation of parallel scans.
- **query_timeout_search** – The timeout period, in seconds, for search queries used in the retrieval of documents from an index.
- **auto_discover_endpoint** – Boolean. The default is true. When you use the Amazon OpenSearch Service and set this parameter to true, the connector can auto-discover your domains and endpoints by calling the appropriate describe or list API operations on the OpenSearch Service. For any other type of OpenSearch instance (for example, self-hosted), you must specify the associated domain endpoints in the domain_mapping variable. If auto_discover_endpoint=true, the connector uses AWS credentials to authenticate to the OpenSearch Service. Otherwise, the connector retrieves user name and password credentials from AWS Secrets Manager through the domain_mapping variable.
- **domain_mapping** – Used only when auto_discover_endpoint is set to false and defines the mapping between domain names and their associated endpoints. The domain_mapping variable can accommodate multiple OpenSearch endpoints in the following format:

```
domain1=endpoint1,domain2=endpoint2,domain3=endpoint3,...
```

  For the purpose of authenticating to an OpenSearch endpoint, the connector supports substitution strings injected using the format ${SecretName}: with user name and password retrieved from AWS Secrets Manager. The colon (:) at the end of the expression serves as a separator from the rest of the endpoint.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS

Secrets Manager, see [Move hardcoded secrets to AWS Secrets Manager] in the *AWS Secrets Manager User Guide*.

The following example uses the `opensearch-creds` secret.

```
movies=https://${opensearch-creds}:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

At runtime, `${opensearch-creds}` is rendered as the user name and password, as in the following example.

```
movies=https://myusername@mypassword:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

In the `domain_mapping` parameter, each domain-endpoint pair can use a different secret. The secret itself must be specified in the format *`user_name`*@*`password`*. Although the password may contain embedded `@` signs, the first `@` serves as a separator from *`user_name`*.

It is also important to note that the comma (,) and equal sign (=) are used by this connector as separators for the domain-endpoint pairs. For this reason, you should not use them anywhere inside the stored secret.

Setting up databases and tables in AWS Glue

The connector obtains metadata information by using AWS Glue or OpenSearch. You can set up an AWS Glue table as a supplemental metadata definition source. To enable this feature, define a AWS Glue database and table that match the domain and index of the source that you are supplementing. The connector can also take advantage of metadata definitions stored in the OpenSearch instance by retrieving the mapping for the specified index.

Defining metadata for arrays in OpenSearch

OpenSearch does not have a dedicated array data type. Any field can contain zero or more values so long as they are of the same data type. If you want to use OpenSearch as your metadata definition source, you must define a `_meta` property for all indices used with Athena for the fields that to be considered a list or array. If you fail to complete this step, queries return only the first element in the list field. When you specify the `_meta` property, field names should be fully qualified for nested JSON structures (for example, `address.street`, where `street` is a nested field inside an `address` structure).

The following example defines `actor` and `genre` lists in the `movies` table.

```
PUT movies/_mapping
{
  "_meta": {
    "actor": "list",
    "genre": "list"
  }
}
```

Data types

The OpenSearch connector can extract metadata definitions from either AWS Glue or the OpenSearch instance. The connector uses the mapping in the following table to convert the definitions to Apache Arrow data types, including the points noted in the section that follows.

OpenSearch	Apache Arrow	AWS Glue
text, keyword, binary	VARCHAR	string
long	BIGINT	bigint
scaled_float	BIGINT	SCALED_FLOAT(...)
integer	INT	int
short	SMALLINT	smallint
byte	TINYINT	tinyint
double	FLOAT8	double
float, half_float	FLOAT4	float
boolean	BIT	boolean
date, date_nanos	DATEMILLI	timestamp
JSON structure	STRUCT	STRUCT

OpenSearch	Apache Arrow	AWS Glue
_meta (for information, see the section Defining metadata for arrays in OpenSearch.)	LIST	ARRAY

Notes on data types

- Currently, the connector supports only the OpenSearch and AWS Glue data-types listed in the preceding table.
- A `scaled_float` is a floating-point number scaled by a fixed double scaling factor and represented as a `BIGINT` in Apache Arrow. For example, 0.756 with a scaling factor of 100 is rounded to 76.
- To define a `scaled_float` in AWS Glue, you must select the `array` column type and declare the field using the format SCALED_FLOAT(*scaling_factor*).

  The following examples are valid:

  ```
  SCALED_FLOAT(10.51)
  SCALED_FLOAT(100)
  SCALED_FLOAT(100.0)
  ```

  The following examples are not valid:

  ```
  SCALED_FLOAT(10.)
  SCALED_FLOAT(.5)
  ```

- When converting from `date_nanos` to `DATEMILLI`, nanoseconds are rounded to the nearest millisecond. Valid values for `date` and `date_nanos` include, but are not limited to, the following formats:

  ```
  "2020-05-18T10:15:30.123456789"
  "2020-05-15T06:50:01.123Z"
  "2020-05-15T06:49:30.123-05:00"
  1589525370001 (epoch milliseconds)
  ```

- An OpenSearch `binary` is a string representation of a binary value encoded using `Base64` and is converted to a `VARCHAR`.

Running SQL queries

The following are examples of DDL queries that you can use with this connector. In the examples, *function_name* corresponds to the name of your Lambda function, *domain* is the name of the domain that you want to query, and *index* is the name of your index.

```
SHOW DATABASES in `lambda:function_name`
```

```
SHOW TABLES in `lambda:function_name`.domain
```

```
DESCRIBE `lambda:function_name`.domain.index
```

Performance

The Athena OpenSearch connector supports shard-based parallel scans. The connector uses cluster health information retrieved from the OpenSearch instance to generate multiple requests for a document search query. The requests are split for each shard and run concurrently.

The connector also pushes down predicates as part of its document search queries. The following example query and predicate shows how the connector uses predicate push down.

Query

```
SELECT * FROM "lambda:elasticsearch".movies.movies
WHERE year >= 1955 AND year <= 1962 OR year = 1996
```

Predicate

```
(_exists_:year) AND year:([1955 TO 1962] OR 1996)
```

Passthrough queries

The OpenSearch connector supports passthrough queries and uses the Query DSL language. For more information about querying with Query DSL, see Query DSL in the Elasticsearch documentation or Query DSL in the OpenSearch documentation.

To use passthrough queries with the OpenSearch connector, use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            schema => 'schema_name',
            index => 'index_name',
            query => "{query_string}"
        ))
```

The following OpenSearch example passthrough query filters for employees with active employment status in the `employee` index of the `default` schema.

```
SELECT * FROM TABLE(
        system.query(
            schema => 'default',
            index => 'employee',
            query => "{ ''bool'':{''filter'':{''term'':{''status'': ''active''}}}}"
        ))
```

Additional resources

- For an article on using the Amazon Athena OpenSearch connector to query data in Amazon OpenSearch Service and Amazon S3 in a single query, see Query data in Amazon OpenSearch Service using SQL from Amazon Athena in the *AWS Big Data Blog*.
- For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Oracle connector

The Amazon Athena connector for Oracle enables Amazon Athena to run SQL queries on data stored in Oracle running on-premises or on Amazon EC2 or Amazon RDS. You can also use the connector to query data on Oracle exadata.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Only version 12.1.0.2 Oracle Databases are supported.
- If the Oracle connector does not use a Glue connection, the database, table, and column names will be converted to upper case by the connector.

  If the Oracle connector uses a Glue connection, the database, table, and column names will not default to upper case by the connector. Add double quotes (") around object names to preserve the case. To change this casing behavior, change the Lambda by environment variable `casing_mode` to `upper` or `lower` as needed.
- When you use the Oracle `NUMBER` without Precision and Scale defined, Athena treats this as `BIGINT`. To get the required decimal places in Athena, specify `default_scale=<number of decimal places>` in your Lambda environment variables.

Terms

The following terms relate to the Oracle connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.

- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Oracle connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
oracle://${jdbc_connection_string}
```

> **Note**
>
> If your password contains special characters (for example, `some.password`), enclose your password in double quotes when you pass it to the connection string (for example, `"some.password"`). Failure to do so can result in an Invalid Oracle URL specified error.

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`OracleMuxCompositeHandler`
Metadata handler	`OracleMuxMetadataHandler`
Record handler	`OracleMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `myoraclecatalog`, then the environment variable name is `myoraclecatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Oracle MUX Lambda function that supports two database instances: `oracle1` (the default), and `oracle2`.

Property	Value
`default`	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename`

Property	Value
`oracle_catalog1_connection_string`	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename`
`oracle_catalog2_connection_string`	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle2}@//oracle2.hostname:port/servicename`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

> **Note**
>
> If your password contains special characters (for example, `some.password`), enclose your password in double quotes when you store it in Secrets Manager (for example, `"some.password"`). Failure to do so can result in an Invalid Oracle URL specified error.

Example connection string with secret name

The following string has the secret name `${Test/RDS/Oracle}`.

```
oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
oracle://jdbc:oracle:thin:username/password@//hostname:port/servicename
```

Currently, the Oracle connector recognizes the `UID` and `PWD` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Oracle instance.

Handler type	Class
Composite handler	`OracleCompositeHandler`
Metadata handler	`OracleMetadataHandler`
Record handler	`OracleRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

Parameter	Description
IsFIPSEnabled	Optional. Set to `true` when FIPS mode is enabled. The default is `false`.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The connector supports SSL based connections to Amazon RDS instances. Support is limited to the Transport Layer Security (TLS) protocol and to authentication of the server by the client. Mutual authentication it is not supported in Amazon RDS. The second row in the table below shows the syntax for using SSL.

The following example property is for a single Oracle instance supported by a Lambda function.

Property	Value
default	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename`
	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS) (HOST=<HOST_NAME>)(PORT=))(CONNECT_DATA=(SID=))(SECURITY=(SSL_SERVER_CERT_DN=)))`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-`

Parameter	Description
	`encryption" : "AES256"}`). For other possible headers, see [PutObject](#) in the *Amazon Simple Storage Service API Reference*.

Casing

You can use the following casing parameter to set different casing modes. You can change the default casing mode in the Lambda environment variables for your connector regardless of a glue connection.

- **casing_mode** – (Optional) Specifies how to handle casing for schema and table names. The `casing_mode` parameter uses the following values to specify the behavior of casing:
  - **lower** – Lower case all given schema and table names. This is the default for connectors that have an associated glue connection.
  - **upper** – Upper case all given schema and table names. This is the default for connectors that do not have an associated glue connection.
  - **case_insensitive_search** – Perform case insensitive searches against schema and tables names in Oracle. Use this value if your query contains schema or table names that do not match the default casing for your connector.

Data type support

The following table shows the corresponding data types for JDBC, Oracle, and Arrow.

JDBC	Oracle	Arrow
Boolean	boolean	Bit
Integer	N/A	Tiny
Short	smallint	Smallint
Integer	integer	Int
Long	bigint	Bigint

JDBC	Oracle	Arrow
float	float4	Float4
Double	float8	Float8
Date	date	DateDay
Timestamp	timestamp	DateMilli
String	text	Varchar
Bytes	bytes	Varbinary
BigDecimal	numeric(p,s)	Decimal
ARRAY	N/A (see note)	List

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

Oracle supports native partitions. The Athena Oracle connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, native partitioning is highly recommended. Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The Oracle connector is resilient to throttling due to concurrency. However, query runtimes tend to be long.

The Athena Oracle connector performs predicate pushdown to decrease the data scanned by the query. Simple predicates and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Oracle connector can combine these expressions and push them directly to Oracle for enhanced functionality and to reduce the amount of data scanned.

The following Athena Oracle connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

Passthrough queries

The Oracle connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Oracle, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Oracle. The query selects all columns in the `customer` table.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Oracle connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena PostgreSQL connector

The Amazon Athena PostgreSQL connector enables Athena to access your PostgreSQL databases.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Like PostgreSQL, Athena treats trailing spaces in PostgreSQL `CHAR` types as semantically insignificant for length and comparison purposes. Note that this applies only to `CHAR` but not to `VARCHAR` types. Athena ignores trailing spaces for the `CHAR` type, but treats them as significant for the `VARCHAR` type.
- When you use the citext case-insensitive character string data type, PostgreSQL uses a case insensitive data comparison that is different from Athena. This difference creates a data discrepancy during SQL `JOIN` operations. To workaround this issue, use the PostgreSQL

connector passthrough query feature. For more information, see the passthrough queries section later in this document.

Terms

The following terms relate to the PostgreSQL connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the PostgreSQL connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

Glue connections (recommended)

We recommended that you configure a PostgreSQL connector by using a Glue connections object.

To do this, set the `glue_connection` environment variable of the PostgreSQL connector Lambda to the name of the Glue connection to use.

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type POSTGRESQL
```

Legacy connections

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
postgres://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`PostGreSqlMuxCompositeHandler`
Metadata handler	`PostGreSqlMuxMetadataHandler`
Record handler	`PostGreSqlMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in

Parameter	Description
	Athena. For example, if the catalog registered with Athena is `mypostgrescatalog` , then the environment variable name is `mypostgrescatalog_connection_string` .
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a PostGreSql MUX Lambda function that supports two database instances: `postgres1` (the default), and `postgres2`.

Property	Value
`default`	`postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}`
`postgres_catalog1_connection_string`	`postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}`
`postgres_catalog2_connection_string`	`postgres://jdbc:postgresql://postgres2.host:5432/default?user=sample&password=sample`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see [Move hardcoded secrets to AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have [internet access](#) or a [VPC endpoint](#) to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

  ```
  {"username": "${username}", "password": "${password}"}
  ```

Example connection string with secret name

The following string has the secret name `${Test/RDS/PostGres1}`.

```
postgres://jdbc:postgresql://postgres1.host:5432/default?...&${Test/RDS/PostGres1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
postgres://jdbc:postgresql://postgres1.host:5432/
default?...&user=sample2&password=sample2&...
```

Currently, the PostgreSQL connector recognizes the `user` and `password` JDBC properties.

Enabling SSL

To support SSL in your PostgreSQL connection, append the following to your connection string:

```
&sslmode=verify-ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

Example

The following example connection string does not use SSL.

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-endpoint:5432/asdf?
user=someuser&password=somepassword
```

To enable SSL, modify the string as follows.

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-
endpoint:5432/asdf?user=someuser&password=somepassword&sslmode=verify-
ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single PostgreSQL instance.

Handler type	Class
Composite handler	`PostGreSqlCompositeHandler`
Metadata handler	`PostGreSqlMetadataHandler`
Record handler	`PostGreSqlRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single PostgreSQL instance supported by a Lambda function.

Property	Value
`default`	`postgres://jdbc:postgresql://postgres1.host:5432/default?secret=${Test/RDS/PostgreSQL1}`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC, PostGreSQL, and Arrow.

JDBC	PostGreSQL	Arrow
Boolean	Boolean	Bit
Integer	N/A	Tiny
Short	smallint	Smallint

JDBC	PostGreSQL	Arrow
Integer	integer	Int
Long	bigint	Bigint
float	float4	Float4
Double	float8	Float8
Date	date	DateDay
Timestamp	timestamp	DateMilli
String	text	Varchar
Bytes	bytes	Varbinary
BigDecimal	numeric(p,s)	Decimal
ARRAY	N/A (see note)	List

Note

The ARRAY type is supported for the PostgreSQL connector with the following constraints: Multidimensional arrays (*<data_type>*`[][]` or nested arrays) are not supported. Columns with unsupported ARRAY data-types are converted to an array of string elements (`array<varchar>`).

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

Performance

PostgreSQL supports native partitions. The Athena PostgreSQL connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, native partitioning is highly recommended.

The Athena PostgreSQL connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time. However, selecting a subset of columns sometimes results in a longer query execution runtime.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena PostgreSQL connector can combine these expressions and push them directly to PostgreSQL for enhanced functionality and to reduce the amount of data scanned.

The following Athena PostgreSQL connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
```

```
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The PostgreSQL connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with PostgreSQL, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in PostgreSQL. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the PostgreSQL connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Redis OSS connector

The Amazon Athena Redis OSS connector enables Amazon Athena to communicate with your Redis OSS instances so that you can query your Redis OSS data with SQL. You can use the AWS Glue Data Catalog to map your Redis OSS key-value pairs into virtual tables.

Unlike traditional relational data stores, Redis OSS does not have the concept of a table or a column. Instead, Redis OSS offers key-value access patterns where the key is essentially a `string` and the value is a `string`, `z-set`, or hmap.

You can use the AWS Glue Data Catalog to create schema and configure virtual tables. Special table properties tell the Athena Redis OSS connector how to map your Redis OSS keys and values into a table. For more information, see Setting up databases and tables in AWS Glue later in this document.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

The Amazon Athena Redis OSS connector supports Amazon MemoryDB and Amazon ElastiCache (Redis OSS).

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

Parameters

Use the parameters in this section to configure the Redis connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your

Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.

- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **glue_catalog** – (Optional) Use this option to specify a cross-account AWS Glue catalog. By default, the connector attempts to get metadata from its own AWS Glue account.

Setting up databases and tables in AWS Glue

To enable an AWS Glue table for use with Redis OSS, you can set the following table properties on the table: `redis-endpoint`, `redis-value-type`, and either `redis-keys-zset` or `redis-key-prefix`.

In addition, any AWS Glue database that contains Redis OSS tables must have a `redis-db-flag` in the URI property of the database. To set the `redis-db-flag` URI property, use the AWS Glue console to edit the database.

The following list describes the table properties.

- **redis-endpoint** – (Required) The *hostname*:*port*:*password* of the Redis OSS server that contains data for this table (for example, `athena-federation-demo.cache.amazonaws.com:6379`) Alternatively, you can store the endpoint, or part of the endpoint, in AWS Secrets Manager by using ${*Secret_Name*} as the table property value.

> **Note**
>
> To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

- **redis-keys-zset** – (Required if `redis-key-prefix` is not used) A comma-separated list of keys whose value is a zset (for example, `active-orders,pending-orders`). Each of the values in the zset is treated as a key that is part of the table. Either the `redis-keys-zset` property or the `redis-key-prefix` property must be set.

- **redis-key-prefix** – (Required if `redis-keys-zset` is not used) A comma separated list of key prefixes to scan for values in the table (for example, `accounts-*,acct-`). Either the `redis-key-prefix` property or the `redis-keys-zset` property must be set.
- **redis-value-type** – (Required) Defines how the values for the keys defined by either `redis-key-prefix` or `redis-keys-zset` are mapped to your table. A literal maps to a single column. A zset also maps to a single column, but each key can store many rows. A hash enables each key to be a row with multiple columns (for example, a hash, literal, or zset.)
- **redis-ssl-flag** – (Optional) When `True`, creates a Redis connection that uses SSL/TLS. The default is `False`.
- **redis-cluster-flag** – (Optional) When `True`, enables support for clustered Redis instances. The default is `False`.
- **redis-db-number** – (Optional) Applies only to standalone, non-clustered instances.) Set this number (for example 1, 2, or 3) to read from a non-default Redis database. The default is Redis logical database 0. This number does not refer to a database in Athena or AWS Glue, but to a Redis logical database. For more information, see SELECT index in the Redis documentation.

Data types

The Redis OSS connector supports the following data types. Redis OSS streams are not supported.

- String
- Hash
- Sorted Set (ZSet)

All Redis OSS values are retrieved as the `string` data type. Then they are converted to one of the following Apache Arrow data types based on how your tables are defined in the AWS Glue Data Catalog.

AWS Glue data type	Apache Arrow data type
int	INT
string	VARCHAR
bigint	BIGINT

AWS Glue data type	Apache Arrow data type
double	FLOAT8
float	FLOAT4
smallint	SMALLINT
tinyint	TINYINT
boolean	BIT
binary	VARBINARY

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-redis.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access –** The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution –** The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **AWS Glue Data Catalog –** The Redis connector requires read only access to the AWS Glue Data Catalog to obtain schema information.
- **CloudWatch Logs –** The connector requires access to CloudWatch Logs for storing logs.
- **AWS Secrets Manager read access –** If you choose to store Redis endpoint details in Secrets Manager, you must grant the connector access to those secrets.
- **VPC access –** The connector requires the ability to attach and detach interfaces to your VPC so that it can connect to it and communicate with your Redis instances.

Performance

The Athena Redis OSS connector attempts to parallelize queries against your Redis OSS instance according to the type of table that you have defined (for example, zset keys or prefix keys).

The Athena Redis connector performs predicate pushdown to decrease the data scanned by the query. However, queries containing a predicate against the primary key fail with timeout. `LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect SELECT queries with a LIMIT clause to scan at least 16 MB of data. The Redis connector is resilient to throttling due to concurrency.

Passthrough queries

The Redis connector supports passthrough queries. You can use this feature to run queries that use Lua script on Redis databases.

To create passthrough queries with Redis, use the following syntax:

```
SELECT * FROM TABLE(
        system.script(
            script => 'return redis.[call|pcall](query_script)',
            keys => '[key_pattern]',
            argv => '[script_arguments]'
))
```

The following example runs a Lua script to get the value at key l:a.

```
SELECT * FROM TABLE(
        system.script(
            script => 'return redis.call("GET", KEYS[1])',
            keys => '[l:a]',
            argv => '[]'
))
```

License information

The Amazon Athena Redis connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Redshift connector

The Amazon Athena Redshift connector enables Amazon Athena to access your Amazon Redshift and Amazon Redshift Serverless databases, including Redshift Serverless views. You can connect to either service using the JDBC connection string configuration settings described on this page.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Because Redshift does not support external partitions, all data specified by a query is retrieved every time.
- Like Redshift, Athena treats trailing spaces in Redshift `CHAR` types as semantically insignificant for length and comparison purposes. Note that this applies only to `CHAR` but not to `VARCHAR` types. Athena ignores trailing spaces for the `CHAR` type, but treats them as significant for the `VARCHAR` type.

Terms

The following terms relate to the Redshift connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.

- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Redshift connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

Glue connections (recommended)

We recommended that you configure a Redshift connector by using a Glue connections object.

To do this, set the `glue_connection` environment variable of the Redshift connector Lambda to the name of the Glue connection to use.

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type REDSHIFT
```

Legacy connections

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
redshift://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`RedshiftMuxCompositeHandler`
Metadata handler	`RedshiftMuxMetadataHandler`
Record handler	`RedshiftMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `myredshiftcatalog`, then the environment variable name is `myredshiftcatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Redshift MUX Lambda function that supports two database instances: `redshift1` (the default), and `redshift2`.

Property	Value
`default`	`redshift://jdbc:redshift://redshift1.host:5439/dev?user=sample2&password=sample2`

Property	Value
`redshift_catalog1_connection_string`	`redshift://jdbc:redshift://redshift1.host:3306/default?${Test/RDS/Redshift1}`
`redshift_catalog2_connection_string`	`redshift://jdbc:redshift://redshift2.host:3333/default?user=sample2&password=sample2`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name ${Test/RDS/ Redshift1}.

```
redshift://jdbc:redshift://redshift1.host:3306/default?...&${Test/RDS/Redshift1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
redshift://jdbc:redshift://redshift1.host:3306/
default?...&user=sample2&password=sample2&...
```

Currently, the Redshift connector recognizes the `user` and `password` JDBC properties.

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see [PutObject](#) in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Apache Arrow.

JDBC	Arrow
Boolean	Bit

JDBC	Arrow
Integer	Tiny
Short	Smallint
Integer	Int
Long	Bigint
float	Float4
Double	Float8
Date	DateDay
Timestamp	DateMilli
String	Varchar
Bytes	Varbinary
BigDecimal	Decimal
ARRAY	List

Partitions and splits

Redshift does not support external partitions. For information about performance related issues, see Performance.

Performance

The Athena Redshift connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, `ORDER BY` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time. However, selecting a subset of columns sometimes results in a longer query execution runtime. Amazon Redshift is particularly susceptible to query execution slowdown when you run multiple queries concurrently.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Top N queries

A top `N` query specifies an ordering of the result set and a limit on the number of rows returned. You can use this type of query to determine the top `N` max values or top `N` min values for your datasets. With top `N` pushdown, the connector returns only `N` ordered rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Redshift connector can combine these expressions and push them directly to Redshift for enhanced functionality and to reduce the amount of data scanned.

The following Athena Redshift connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
ORDER BY col_a DESC
LIMIT 10;
```

For an article on using predicate pushdown to improve performance in federated queries, including Amazon Redshift, see Improve federated queries with predicate pushdown in Amazon Athena in the *AWS Big Data Blog*.

Passthrough queries

The Redshift connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Redshift, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Redshift. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Redshift connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena SAP HANA connector

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- In SAP HANA, object names are converted to uppercase when they are stored in the SAP HANA database. However, because names in quotation marks are case sensitive, it is possible for two tables to have the same name in lower and upper case (for example, `EMPLOYEE` and `employee`).

  In Athena Federated Query, schema table names are provided to the Lambda function in lower case. To work around this issue, you can provide `@schemaCase` query hints to retrieve the data from the tables that have case sensitive names. Following are two sample queries with query hints.

  ```
  SELECT *
  FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
  ```

  ```
  SELECT *
  FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
  ```

Terms

The following terms relate to the SAP HANA connector.

- **Database instance** – Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.

- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler** – A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the SAP HANA connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
saphana://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`SaphanaMuxCompositeHandler`
Metadata handler	`SaphanaMuxMetadataHandler`
Record handler	`SaphanaMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mysaphanacatalog`, then the environment variable name is `mysaphanacatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Saphana MUX Lambda function that supports two database instances: `saphana1` (the default), and `saphana2`.

Property	Value
`default`	`saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}`
`saphana_catalog1_connection_string`	`saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}`

Property	Value
saphana_catalog2_connection_string	saphana://jdbc:sap://saphana2.host:port/?user=sample2&password=sample2

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/RDS/Saphana1}`.

```
saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
saphana://jdbc:sap://saphana1.host:port/?user=sample2&password=sample2&...
```

Currently, the SAP HANA connector recognizes the `user` and `password` JDBC properties.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single SAP HANA instance.

Handler type	Class
Composite handler	`SaphanaCompositeHandler`
Metadata handler	`SaphanaMetadataHandler`
Record handler	`SaphanaRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single SAP HANA instance supported by a Lambda function.

Property	Value
`default`	`saphana://jdbc:sap://saphana1.host:port/?secret=Test/RDS/Saphana1`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Apache Arrow.

JDBC	Arrow
Boolean	Bit
Integer	Tiny
Short	Smallint
Integer	Int

JDBC	Arrow
Long	Bigint
float	Float4
Double	Float8
Date	DateDay
Timestamp	DateMilli
String	Varchar
Bytes	Varbinary
BigDecimal	Decimal
ARRAY	List

Data type conversions

In addition to the JDBC to Arrow conversions, the connector performs certain other conversions to make the SAP HANA source and Athena data types compatible. These conversions help ensure that queries get executed successfully. The following table shows these conversions.

Source data type (SAP HANA)	Converted data type (Athena)
DECIMAL	BIGINT
INTEGER	INT
DATE	DATEDAY
TIMESTAMP	DATEMILLI

All other unsupported data types are converted to VARCHAR.

Partitions and splits

A partition is represented by a single partition column of type `Integer`. The column contains partition names of the partitions defined on an SAP HANA table. For a table that does not have partition names, * is returned, which is equivalent to a single partition. A partition is equivalent to a split.

Name	Type	Description
PART_ID	Integer	Named partition in SAP HANA.

Performance

SAP HANA supports native partitions. The Athena SAP HANA connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, native partitioning is highly recommended. Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The connector shows significant throttling, and sometimes query failures, due to concurrency.

The Athena SAP HANA connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena SAP HANA connector can combine these expressions and push them directly to SAP HANA for enhanced functionality and to reduce the amount of data scanned.

The following Athena SAP HANA connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL

- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The SAP HANA connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with SAP HANA, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in SAP HANA. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the SAP HANA connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Snowflake connector

The Amazon Athena connector for Snowflake enables Amazon Athena to run SQL queries on data stored in your Snowflake SQL database or RDS instances using JDBC.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- Currently, Snowflake views with single split are supported.
- In Snowflake, because object names are case sensitive, two tables can have the same name in lower and upper case (for example, `EMPLOYEE` and `employee`). In Athena Federated Query, schema table names are provided to the Lambda function in lower case. To work around this issue, you can provide `@schemaCase` query hints to retrieve the data from the tables that have case sensitive names. Following are two sample queries with query hints.

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
```

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
```

- If you migrate your Snowflake connections to Glue Catalog and Lake Formation, Athena will not default all requests to upper case or support annotation. The default behavior for Glue Connection will not adjust casing.

  Snowflake supports the following casing modes:

  - NONE (default for connector with Glue Connection)
  - CASE_INSENSITIVE_SEARCH
  - ANNOTATION (default for connector without Glue Connection)

Terms

The following terms relate to the Snowflake connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Snowflake connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

Glue connections (recommended)

We recommended that you configure a Snowflake connector by using a Glue connections object.

To do this, set the `glue_connection` environment variable of the Snowflake connector Lambda to the name of the Glue connection to use.

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type SNOWFLAKE
```

Legacy connections

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you [manually deploy](#) an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
snowflake://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`SnowflakeMuxCompositeHandler`
Metadata handler	`SnowflakeMuxMetadataHandler`
Record handler	`SnowflakeMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mysnowflakecatalog`, then the environment variable name is `mysnowflakecatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Snowflake MUX Lambda function that supports two database instances: `snowflake1` (the default), and `snowflake2`.

Property	Value
`default`	`snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1&${Test/RDS/Snowflake1}`
`snowflake_catalog1_connection_string`	`snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}`

Property	Value
snowflake_catalog2_connection_string	snowflake://jdbc:snowflake://snowflake2.host:port/?warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/RDS/Snowflake1}`.

```
snowflake://jdbc:snowflake://snowflake1.host:port/?
warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
snowflake://jdbc:snowflake://snowflake1.host:port/
warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2&...
```

Currently, Snowflake recognizes the `user` and `password` JDBC properties. It also accepts the user name and password in the format *`username`*`/`*`password`* without the keys `user` or `password`.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Snowflake instance.

Handler type	Class
Composite handler	`SnowflakeCompositeHandler`
Metadata handler	`SnowflakeMetadataHandler`
Record handler	`SnowflakeRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Snowflake instance supported by a Lambda function.

Property	Value
default	snowflake://jdbc:snowflake://snowflake1.host:port/?secret=Test/RDS/Snowflake1

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
spill_bucket	Required. Spill bucket name.
spill_prefix	Required. Spill bucket key prefix.
spill_put_request_headers	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Casing

You can use the following casing parameter to set different casing modes. You can change the default casing mode in the Lambda environment variables for your connector regardless of a glue connection.

- **casing_mode** – (Optional) Specifies how to handle casing for schema and table names. The `casing_mode` parameter uses the following values to specify the behavior of casing:
  - **none** – Do not change case of the given schema and table names. This is the default for connectors that have an associated glue connection.
  - **annotation** – Adjusts the table name if an annotation is present. This is the default for connectors that do not have an associated glue connection.

- **case_insensitive_search** – Perform case insensitive searches against schema and tables names in Snowflake. Use this value if your query contains schema or table names that do not match the default casing for your connector.

Data type support

The following table shows the corresponding data types for JDBC and Apache Arrow.

JDBC	Arrow
Boolean	Bit
Integer	Tiny
Short	Smallint
Integer	Int
Long	Bigint
float	Float4
Double	Float8
Date	DateDay
Timestamp	DateMilli
String	Varchar
Bytes	Varbinary
BigDecimal	Decimal
ARRAY	List

Data type conversions

In addition to the JDBC to Arrow conversions, the connector performs certain other conversions to make the Snowflake source and Athena data types compatible. These conversions help ensure that queries get executed successfully. The following table shows these conversions.

Source data type (Snowflake)	Converted data type (Athena)
TIMESTAMP	TIMESTAMPMILLI
DATE	TIMESTAMPMILLI
INTEGER	INT
DECIMAL	BIGINT
TIMESTAMP_NTZ	TIMESTAMPMILLI

All other unsupported data types are converted to `VARCHAR`.

Partitions and splits

Partitions are used to determine how to generate splits for the connector. Athena constructs a synthetic column of type `varchar` that represents the partitioning scheme for the table to help the connector generate splits. The connector does not modify the actual table definition.

To create this synthetic column and the partitions, Athena requires a primary key to be defined. However, because Snowflake does not enforce primary key constraints, you must enforce uniqueness yourself. Failure to do so causes Athena to default to a single split.

Performance

For optimal performance, use filters in queries whenever possible. In addition, we highly recommend native partitioning to retrieve huge datasets that have uniform partition distribution. Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The Snowflake connector is resilient to throttling due to concurrency.

The Athena Snowflake connector performs predicate pushdown to decrease the data scanned by the query. `LIMIT` clauses, simple predicates, and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

LIMIT clauses

A `LIMIT N` statement reduces the data scanned by the query. With `LIMIT N` pushdown, the connector returns only `N` rows to Athena.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Snowflake connector can combine these expressions and push them directly to Snowflake for enhanced functionality and to reduce the amount of data scanned.

The following Athena Snowflake connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

Passthrough queries

The Snowflake connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Snowflake, you can use the following syntax:

```
SELECT * FROM TABLE(
```

```
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Snowflake. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Snowflake connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Microsoft SQL Server connector

The Amazon Athena connector for Microsoft SQL Server enables Amazon Athena to run SQL queries on your data stored in Microsoft SQL Server using JDBC.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- In filter conditions, you must cast the `Date` and `Timestamp` data types to the appropriate data type.
- To search for negative values of type `Real` and `Float`, use the <= or >= operator.
- The `binary`, `varbinary`, `image`, and `rowversion` data types are not supported.

Terms

The following terms relate to the SQL Server connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the SQL Server connector.

Note

Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
sqlserver://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`SqlServerMuxCompositeHandler`
Metadata handler	`SqlServerMuxMetadataHandler`
Record handler	`SqlServerMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `mysqlservercatalog`, then the environment variable name is `mysqlservercatalog_connection_string`.

Parameter	Description
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a SqlServer MUX Lambda function that supports two database instances: `sqlserver1` (the default), and `sqlserver2`.

Property	Value
`default`	`sqlserver://jdbc:sqlserver://sqlserver1. hostname:port;databaseName= <database_name> ;${secret1_name }`
`sqlserver_catalog1_connection_string`	`sqlserver://jdbc:sqlserver://sqlserver1. hostname:port;databaseName= <database_name> ;${secret1_name }`
`sqlserver_catalog2_connection_string`	`sqlserver://jdbc:sqlserver://sqlserver2. hostname:port;databaseName= <database_name> ;${secret2_name }`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

  ```
  {"username": "${username}", "password": "${password}"}
  ```

Example connection string with secret name

The following string has the secret name `${secret_name}`.

```
sqlserver://jdbc:sqlserver://hostname:port;databaseName=<database_name>;${secret_name}
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
sqlserver://
jdbc:sqlserver://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single SQL Server instance.

Handler type	Class
Composite handler	`SqlServerCompositeHandler`
Metadata handler	`SqlServerMetadataHandler`

Handler type	Class
Record handler	SqlServerRecordHandler

Single connection handler parameters

Parameter	Description
default	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single SQL Server instance supported by a Lambda function.

Property	Value
default	`sqlserver://jdbc:sqlserver:// hostname:port;databaseName= <database_name> ;${secret_name }`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
spill_bucket	Required. Spill bucket name.
spill_prefix	Required. Spill bucket key prefix.
spill_put_request_headers	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers,

Parameter	Description
	see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for SQL Server and Apache Arrow.

SQL Server	Arrow
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)

SQL Server	Arrow
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR
text	VARCHAR
ntext	VARCHAR

Partitions and splits

A partition is represented by a single partition column of type `varchar`. In case of the SQL Server connector, a partition function determines how partitions are applied on the table. The partition function and column name information are retrieved from the SQL Server metadata table. A custom query then gets the partition. Splits are created based upon the number of distinct partitions received.

Performance

Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The SQL Server connector is resilient to throttling due to concurrency.

The Athena SQL Server connector performs predicate pushdown to decrease the data scanned by the query. Simple predicates and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena SQL Server connector can combine these expressions and push them directly to SQL Server for enhanced functionality and to reduce the amount of data scanned.

The following Athena SQL Server connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

Passthrough queries

The SQL Server connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with SQL Server, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in SQL Server. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
```

```
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the SQL Server connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Teradata connector

The Amazon Athena connector for Teradata enables Athena to run SQL queries on data stored in your Teradata databases.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Limitations

- Write DDL operations are not supported.
- In a multiplexer setup, the spill bucket and prefix are shared across all database instances.
- Any relevant Lambda limits. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.

Terms

The following terms relate to the Teradata connector.

- **Database instance –** Any instance of a database deployed on premises, on Amazon EC2, or on Amazon RDS.
- **Handler –** A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler –** A Lambda handler that retrieves metadata from your database instance.
- **Record handler –** A Lambda handler that retrieves data records from your database instance.
- **Composite handler –** A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter –** A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String –** A string of text used to establish a connection to a database instance.
- **Catalog –** A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.
- **Multiplexing handler –** A Lambda handler that can accept and use multiple database connections.

Parameters

Use the parameters in this section to configure the Teradata connector.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
teradata://${jdbc_connection_string}
```

Using a multiplexing handler

You can use a multiplexer to connect to multiple database instances with a single Lambda function. Requests are routed by catalog name. Use the following classes in Lambda.

Handler	Class
Composite handler	`TeradataMuxCompositeHandler`
Metadata handler	`TeradataMuxMetadataHandler`

Handler	Class
Record handler	`TeradataMuxRecordHandler`

Multiplexing handler parameters

Parameter	Description
`$catalog_connection_string`	Required. A database instance connection string. Prefix the environment variable with the name of the catalog used in Athena. For example, if the catalog registered with Athena is `myteradatacatalog`, then the environment variable name is `myteradatacatalog_connection_string`.
`default`	Required. The default connection string. This string is used when the catalog is `lambda:${ AWS_LAMBDA_FUNCTION_NAME }`.

The following example properties are for a Teradata MUX Lambda function that supports two database instances: `teradata1` (the default), and `teradata2`.

Property	Value
`default`	`teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2`
`teradata_catalog1_connection_string`	`teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/Teradata1}`
`teradata_catalog2_connection_string`	`teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2`

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret name

The following string has the secret name `${Test/RDS/Teradata1}`.

```
teradata://jdbc:teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/
Teradata1}&...
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
teradata://jdbc:teradata://teradata1.host/
TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,...&user=sample2&password=sample2&...
```

Currently, Teradata recognizes the `user` and `password` JDBC properties. It also accepts the user name and password in the format *`username`*`/`*`password`* without the keys `user` or `password`.

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Teradata instance.

Handler type	Class
Composite handler	`TeradataCompositeHandler`
Metadata handler	`TeradataMetadataHandler`
Record handler	`TeradataRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

The following example property is for a single Teradata instance supported by a Lambda function.

Property	Value
`default`	`teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,secret=Test/RDS/Teradata1`

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the corresponding data types for JDBC and Apache Arrow.

JDBC	Arrow
Boolean	Bit
Integer	Tiny
Short	Smallint
Integer	Int
Long	Bigint
float	Float4
Double	Float8
Date	DateDay

JDBC	Arrow
Timestamp	DateMilli
String	Varchar
Bytes	Varbinary
BigDecimal	Decimal
ARRAY	List

Partitions and splits

A partition is represented by a single partition column of type `Integer`. The column contains partition names of the partitions defined on a Teradata table. For a table that does not have partition names, * is returned, which is equivalent to a single partition. A partition is equivalent to a split.

Name	Type	Description
partition	Integer	Named partition in Teradata.

Performance

Teradata supports native partitions. The Athena Teradata connector can retrieve data from these partitions in parallel. If you want to query very large datasets with uniform partition distribution, native partitioning is highly recommended. Selecting a subset of columns significantly slows down query runtime. The connector shows some throttling due to concurrency.

The Athena Teradata connector performs predicate pushdown to decrease the data scanned by the query. Simple predicates and complex expressions are pushed down to the connector to reduce the amount of data scanned and decrease query execution run time.

Predicates

A predicate is an expression in the `WHERE` clause of a SQL query that evaluates to a Boolean value and filters rows based on multiple conditions. The Athena Teradata connector can combine these expressions and push them directly to Teradata for enhanced functionality and to reduce the amount of data scanned.

The following Athena Teradata connector operators support predicate pushdown:

- **Boolean:** AND, OR, NOT
- **Equality:** EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- **Arithmetic:** ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- **Other:** LIKE_PATTERN, IN

Combined pushdown example

For enhanced querying capabilities, combine the pushdown types, as in the following example:

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

Passthrough queries

The Teradata connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Teradata, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Teradata. The query selects all columns in the `customer` table.

```
SELECT * FROM TABLE(
        system.query(
```

```
            query => 'SELECT * FROM customer'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Teradata connector on GitHub.com.

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Timestream connector

The Amazon Athena Timestream connector enables Amazon Athena to communicate with Amazon Timestream, making your time series data accessible through Amazon Athena. You can optionally use AWS Glue Data Catalog as a source of supplemental metadata.

Amazon Timestream is a fast, scalable, fully managed, purpose-built time series database that makes it easy to store and analyze trillions of time series data points per day. Timestream saves you time and cost in managing the lifecycle of time series data by keeping recent data in memory and moving historical data to a cost optimized storage tier based upon user defined policies.

This connector can be registered with Glue Data Catalog as a federated catalog. It supports data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

If you have Lake Formation enabled in your account, the IAM role for your Athena federated Lambda connector that you deployed in the AWS Serverless Application Repository must have read access in Lake Formation to the AWS Glue Data Catalog.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the Timestream connector.

Glue connections (recommended)

We recommended that you configure a Timestream connector by using a Glue connections object. To do this, set the `glue_connection` environment variable of the Timestream connector Lambda to the name of the Glue connection to use.

Glue connections properties

Use the following command to get the schema for a Glue connection object. This schema contains all the parameters that you can use to control your connection.

```
aws glue describe-connection-type --connection-type TIMESTREAM
```

Lambda environment properties

glue_connection – Specifies the name of the Glue connection associated with the federated connector.

Legacy connections

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.

The parameter names and definitions listed below are for Athena data source connectors created without an associated Glue connection. Use the following parameters only when you manually deploy an earlier version of an Athena data source connector or when the `glue_connection` environment property is not specified.

Lambda environment properties

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.

- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.
- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.
- **glue_catalog** – (Optional) Use this option to specify a cross-account AWS Glue catalog. By default, the connector attempts to get metadata from its own AWS Glue account.

Setting up databases and tables in AWS Glue

You can optionally use the AWS Glue Data Catalog as a source of supplemental metadata. To enable an AWS Glue table for use with Timestream, you must have an AWS Glue database and table with names that match the Timestream database and table that you want to supply supplemental metadata for.

> **Note**
>
> For best performance, use only lowercase for your database names and table names. Using mixed casing causes the connector to perform a case insensitive search that is more computationally intensive.

To configure AWS Glue table for use with Timestream, you must set its table properties in AWS Glue.

To use an AWS Glue table for supplemental metadata

1. Edit the table in the AWS Glue console to add the following table properties:

   - **timestream-metadata-flag** – This property indicates to the Timestream connector that the connector can use the table for supplemental metadata. You can provide any value for timestream-metadata-flag as long as the timestream-metadata-flag property is present in the list of table properties.

- **_view_template** – When you use AWS Glue for supplemental metadata, you can use this table property and specify any Timestream SQL as the view. The Athena Timestream connector uses the SQL from the view together with your SQL from Athena to run your query. This is useful if you want to use a feature of Timestream SQL that is not otherwise available in Athena.

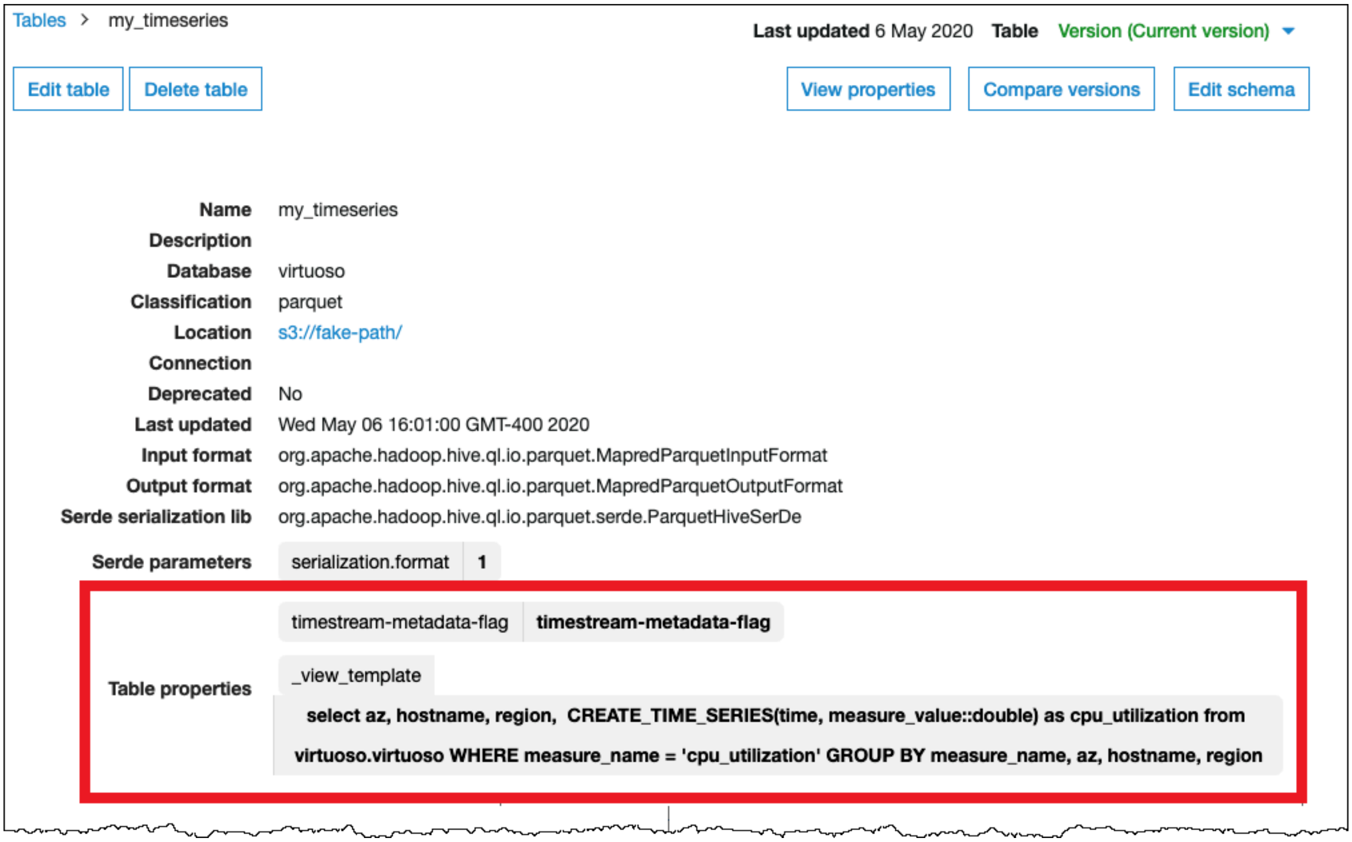
2. Make sure that you use the data types appropriate for AWS Glue as listed in this document.

Data types

Currently, the Timestream connector supports only a subset of the data types available in Timestream, specifically: the scalar values varchar, double, and timestamp.

To query the timeseries data type, you must configure a view in AWS Glue table properties that uses the Timestream CREATE_TIME_SERIES function. You also need to provide a schema for the view that uses the syntax ARRAY<STRUCT<time:timestamp,measure_value::double:double>> as the type for any of your time series columns. Be sure to replace double with the appropriate scalar type for your table.

The following image shows an example of AWS Glue table properties configured to set up a view over a time series.



Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-timestream.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access** – The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution** – The connector uses this permission to fast-fail when the upstream Athena query has terminated.
- **AWS Glue Data Catalog** – The Timestream connector requires read only access to the AWS Glue Data Catalog to obtain schema information.
- **CloudWatch Logs** – The connector requires access to CloudWatch Logs for storing logs.
- **Timestream Access** – For running Timestream queries.

Performance

We recommend that you use the `LIMIT` clause to limit the data returned (not the data scanned) to less than 256 MB to ensure that interactive queries are performant.

The Athena Timestream connector performs predicate pushdown to decrease the data scanned by the query. LIMIT clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect SELECT queries with a LIMIT clause to scan at least 16 MB of data. Selecting a subset of columns significantly speeds up query runtime and reduces data scanned. The Timestream connector is resilient to throttling due to concurrency.

Passthrough queries

The Timestream connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Timestream, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

The following example query pushes down a query to a data source in Timestream. The query selects all columns in the customer table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

The Amazon Athena Timestream connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena TPC benchmark DS (TPC-DS) connector

The Amazon Athena TPC-DS connector enables Amazon Athena to communicate with a source of randomly generated TPC Benchmark DS data for use in benchmarking and functional testing of Athena Federation. The Athena TPC-DS connector generates a TPC-DS compliant database at one of four scale factors. We do not recommend the use of this connector as an alternative to Amazon S3-based data lake performance tests.

This connector can not be registered with Glue Data Catalog as a federated catalog. This connector does not support data access controls defined in Lake Formation at the catalog, database, table, column, row, and tag levels. This connector uses Glue Connections to centralize configuration properties in Glue.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.

Parameters

Use the parameters in this section to configure the TPC-DS connector.

> **Note**
>
> Athena data source connectors created on December 3, 2024 and later use AWS Glue connections.
> The parameter names and definitions listed below are for Athena data source connectors created prior to December 3, 2024. These can differ from their corresponding AWS Glue connection properties. Starting December 3, 2024, use the parameters below only when you manually deploy an earlier version of an Athena data source connector.

- **spill_bucket** – Specifies the Amazon S3 bucket for data that exceeds Lambda function limits.
- **spill_prefix** – (Optional) Defaults to a subfolder in the specified `spill_bucket` called `athena-federation-spill`. We recommend that you configure an Amazon S3 storage lifecycle on this location to delete spills older than a predetermined number of days or hours.
- **spill_put_request_headers** – (Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.
- **kms_key_id** – (Optional) By default, any data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key. To have your Lambda function use stronger encryption keys generated by KMS like `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`, you can specify a KMS key ID.

- **disable_spill_encryption** – (Optional) When set to `True`, disables spill encryption. Defaults to `False` so that data that is spilled to S3 is encrypted using AES-GCM – either using a randomly generated key or KMS to generate keys. Disabling spill encryption can improve performance, especially if your spill location uses server-side encryption.

Test databases and tables

The Athena TPC-DS connector generates a TPC-DS compliant database at one of the four scale factors `tpcds1`, `tpcds10`, `tpcds100`, `tpcds250`, or `tpcds1000`.

Summary of tables

For a complete list of the test data tables and columns, run the `SHOW TABLES` or `DESCRIBE TABLE` queries. The following summary of tables is provided for convenience.

1. call_center
2. catalog_page
3. catalog_returns
4. catalog_sales
5. customer
6. customer_address
7. customer_demographics
8. date_dim
9. dbgen_version
10. household_demographics
11. income_band
12. inventory
13. item
14. promotion
15. reason
16. ship_mode
17. store
18. store_returns
19. store_sales

20time_dim

21warehouse

22web_page

23web_returns

24web_sales

25web_site

For TPC-DS queries that are compatible with this generated schema and data, see the athena-tpcds/src/main/resources/queries/ directory on GitHub.

Example query

The following `SELECT` query example queries the `tpcds` catalog for customer demographics in specific counties.

```
SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  count(*) cnt1,
  cd_purchase_estimate,
  count(*) cnt2,
  cd_credit_rating,
  count(*) cnt3,
  cd_dep_count,
  count(*) cnt4,
  cd_dep_employed_count,
  count(*) cnt5,
  cd_dep_college_count,
  count(*) cnt6
FROM
  "lambda:tpcds".tpcds1.customer c, "lambda:tpcds".tpcds1.customer_address ca,
 "lambda:tpcds".tpcds1.customer_demographics
WHERE
  c.c_current_addr_sk = ca.ca_address_sk AND
    ca_county IN ('Rush County', 'Toole County', 'Jefferson County',
                  'Dona Ana County', 'La Porte County') AND
    cd_demo_sk = c.c_current_cdemo_sk AND
    exists(SELECT *
           FROM "lambda:tpcds".tpcds1.store_sales, "lambda:tpcds".tpcds1.date_dim
```

```
            WHERE c.c_customer_sk = ss_customer_sk AND
              ss_sold_date_sk = d_date_sk AND
              d_year = 2002 AND
              d_moy BETWEEN 1 AND 1 + 3) AND
     (exists(SELECT *
             FROM "lambda:tpcds".tpcds1.web_sales, "lambda:tpcds".tpcds1.date_dim
             WHERE c.c_customer_sk = ws_bill_customer_sk AND
               ws_sold_date_sk = d_date_sk AND
               d_year = 2002 AND
               d_moy BETWEEN 1 AND 1 + 3) OR
       exists(SELECT *
              FROM "lambda:tpcds".tpcds1.catalog_sales, "lambda:tpcds".tpcds1.date_dim
              WHERE c.c_customer_sk = cs_ship_customer_sk AND
                cs_sold_date_sk = d_date_sk AND
                d_year = 2002 AND
                d_moy BETWEEN 1 AND 1 + 3))
GROUP BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
ORDER BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
LIMIT 100
```

Required Permissions

For full details on the IAM policies that this connector requires, review the `Policies` section of the athena-tpcds.yaml file. The following list summarizes the required permissions.

- **Amazon S3 write access –** The connector requires write access to a location in Amazon S3 in order to spill results from large queries.
- **Athena GetQueryExecution –** The connector uses this permission to fast-fail when the upstream Athena query has terminated.

Performance

The Athena TPC-DS connector attempts to parallelize queries based on the scale factor that you choose. Predicate pushdown is performed within the Lambda function.

License information

The Amazon Athena TPC-DS connector project is licensed under the Apache-2.0 License.

Additional resources

For additional information about this connector, visit the corresponding site on GitHub.com.

Amazon Athena Vertica connector

Vertica is a columnar database platform that can be deployed in the cloud or on premises that supports exabyte scale data warehouses. You can use the Amazon Athena Vertica connector in federated queries to query Vertica data sources from Athena. For example, you can run analytical queries over a data warehouse on Vertica and a data lake in Amazon S3.

This connector does not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

Prerequisites

- Deploy the connector to your AWS account using the Athena console or the AWS Serverless Application Repository. For more information, see Create a data source connection or Use the AWS Serverless Application Repository to deploy a data source connector.
- Set up a VPC and a security group before you use this connector. For more information, see Create a VPC for a data source connector or AWS Glue connection.

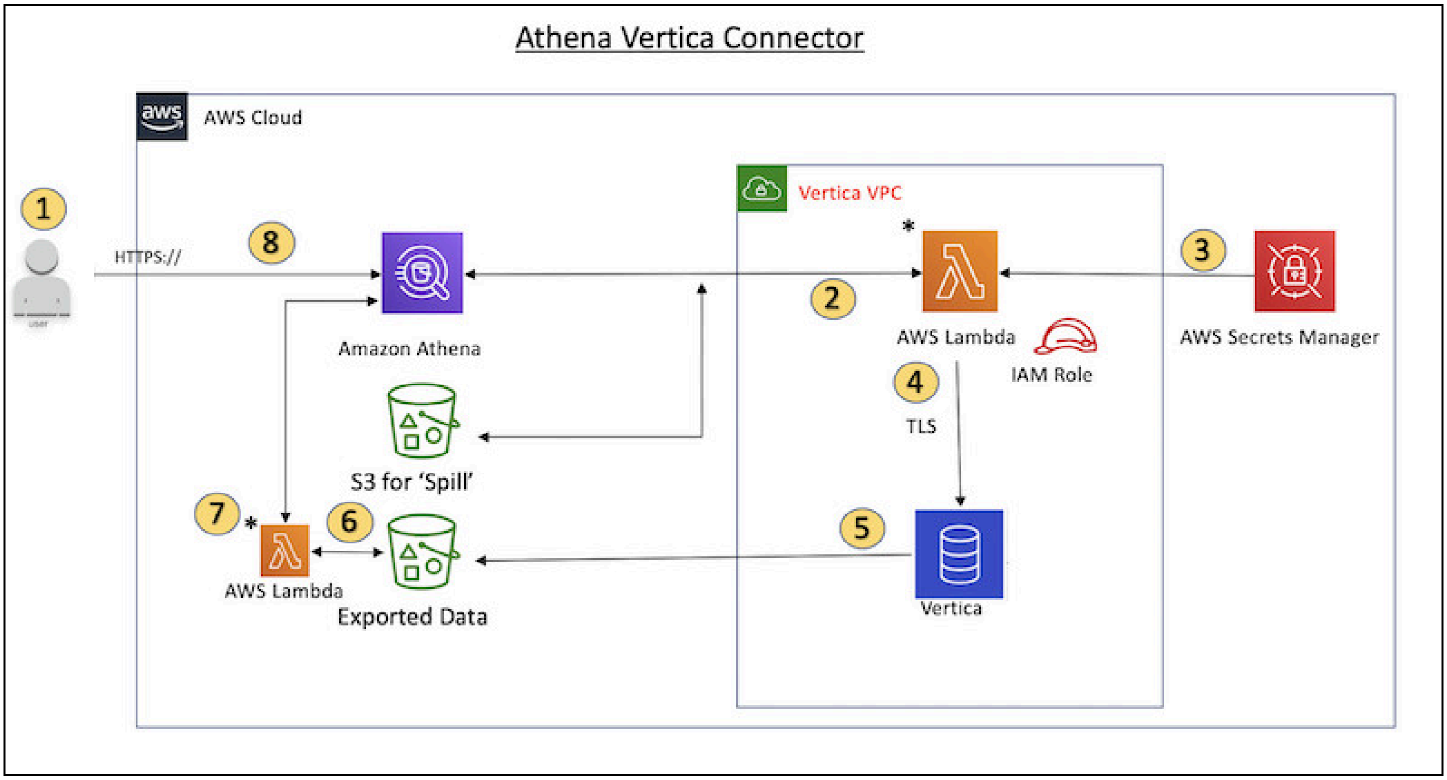
Limitations

- Because the Athena Vertica connector reads exported Parquet files from Amazon S3, performance of the connector can be slow. When you query large tables, we recommend that you use a CREATE TABLE AS (SELECT ...) query and SQL predicates.
- Currently, due to a known issue in Athena Federated Query, the connector causes Vertica to export all columns of the queried table to Amazon S3, but only the queried columns are visible in the results on the Athena console.
- Write DDL operations are not supported.

- Any relevant Lambda limits. For more information, see [Lambda quotas](#) in the *AWS Lambda Developer Guide*.

Workflow

The following diagram shows the workflow of a query that uses the Vertica connector.



1. A SQL query is issued against one or more tables in Vertica.
2. The connector parses the SQL query to send the relevant portion to Vertica through the JDBC connection.
3. The connection strings use the user name and password stored in AWS Secrets Manager to gain access to Vertica.
4. The connector wraps the SQL query with a Vertica `EXPORT` command, as in the following example.

```
EXPORT TO PARQUET (directory = 's3://amzn-s3-demo-bucket/folder_name,
   Compression='Snappy', fileSizeMB=64) OVER() as
SELECT
PATH_ID,
...
SOURCE_ITEMIZED,
```

```
SOURCE_OVERRIDE
FROM DELETED_OBJECT_SCHEMA.FORM_USAGE_DATA
WHERE PATH_ID <= 5;
```

5. Vertica processes the SQL query and sends the result set to an Amazon S3 bucket. For better throughput, Vertica uses the `EXPORT` option to parallelize the write operation of multiple Parquet files.
6. Athena scans the Amazon S3 bucket to determine the number of files to read for the result set.
7. Athena makes multiple calls to the Lambda function and uses an Apache `ArrowReader` to read the Parquet files from the resulting data set. Multiple calls enable Athena to parallelize the reading of the Amazon S3 files and achieve a throughput of up to 100GB per second.
8. Athena processes the data returned from Vertica with data scanned from the data lake and returns the result.

Terms

The following terms relate to the Vertica connector.

- **Database instance** – Any instance of a Vertica database deployed on Amazon EC2.
- **Handler** – A Lambda handler that accesses your database instance. A handler can be for metadata or for data records.
- **Metadata handler** – A Lambda handler that retrieves metadata from your database instance.
- **Record handler** – A Lambda handler that retrieves data records from your database instance.
- **Composite handler** – A Lambda handler that retrieves both metadata and data records from your database instance.
- **Property or parameter** – A database property used by handlers to extract database information. You configure these properties as Lambda environment variables.
- **Connection String** – A string of text used to establish a connection to a database instance.
- **Catalog** – A non-AWS Glue catalog registered with Athena that is a required prefix for the `connection_string` property.

Parameters

The Amazon Athena Vertica connector exposes configuration options through Lambda environment variables. You can use the following Lambda environment variables to configure the connector.

- **AthenaCatalogName** – Lambda function name
- **ExportBucket** – The Amazon S3 bucket where the Vertica query results are exported.
- **SpillBucket** – The name of the Amazon S3 bucket where this function can spill data.
- **SpillPrefix** – The prefix for the `SpillBucket` location where this function can spill data.
- **SecurityGroupIds** – One or more IDs that correspond to the security group that should be applied to the Lambda function (for example, `sg1`, `sg2`, or `sg3`).
- **SubnetIds** – One or more subnet IDs that correspond to the subnet that the Lambda function can use to access your data source (for example, `subnet1`, or `subnet2`).
- **SecretNameOrPrefix** – The name or prefix of a set of names in Secrets Manager that this function has access to (for example, `vertica-*`)
- **VerticaConnectionString** – The Vertica connection details to use by default if no catalog specific connection is defined. The string can optionally use AWS Secrets Manager syntax (for example, `${secret_name}`).
- **VPC ID** – The VPC ID to be attached to the Lambda function.

Connection string

Use a JDBC connection string in the following format to connect to a database instance.

```
vertica://jdbc:vertica://host_name:port/database?user=vertica-
username&password=vertica-password
```

Using a single connection handler

You can use the following single connection metadata and record handlers to connect to a single Vertica instance.

Handler type	Class
Composite handler	`VerticaCompositeHandler`
Metadata handler	`VerticaMetadataHandler`
Record handler	`VerticaRecordHandler`

Single connection handler parameters

Parameter	Description
`default`	Required. The default connection string.

The single connection handlers support one database instance and must provide a `default` connection string parameter. All other connection strings are ignored.

Providing credentials

To provide a user name and password for your database in your JDBC connection string, you can use connection string properties or AWS Secrets Manager.

- **Connection String** – A user name and password can be specified as properties in the JDBC connection string.

> **Important**
>
> As a security best practice, don't use hardcoded credentials in your environment variables or connection strings. For information about moving your hardcoded secrets to AWS Secrets Manager, see Move hardcoded secrets to AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

- **AWS Secrets Manager** – To use the Athena Federated Query feature with AWS Secrets Manager, the VPC connected to your Lambda function should have internet access or a VPC endpoint to connect to Secrets Manager.

  You can put the name of a secret in AWS Secrets Manager in your JDBC connection string. The connector replaces the secret name with the `username` and `password` values from Secrets Manager.

  For Amazon RDS database instances, this support is tightly integrated. If you use Amazon RDS, we highly recommend using AWS Secrets Manager and credential rotation. If your database does not use Amazon RDS, store the credentials as JSON in the following format:

```
{"username": "${username}", "password": "${password}"}
```

Example connection string with secret names

The following string has the secret names `${vertica-username}` and `${vertica-password}`.

```
vertica://jdbc:vertica://host_name:port/database?user=${vertica-username}&password=
${vertica-password}
```

The connector uses the secret name to retrieve secrets and provide the user name and password, as in the following example.

```
vertica://jdbc:vertica://host_name:port/database?user=sample-user&password=sample-
password
```

Currently, the Vertica connector recognizes the `vertica-username` and `vertica-password` JDBC properties.

Spill parameters

The Lambda SDK can spill data to Amazon S3. All database instances accessed by the same Lambda function spill to the same location.

Parameter	Description
`spill_bucket`	Required. Spill bucket name.
`spill_prefix`	Required. Spill bucket key prefix.
`spill_put_request_headers`	(Optional) A JSON encoded map of request headers and values for the Amazon S3 `putObject` request that is used for spilling (for example, `{"x-amz-server-side-encryption" : "AES256"}`). For other possible headers, see PutObject in the *Amazon Simple Storage Service API Reference*.

Data type support

The following table shows the supported data types for the Vertica connector.

Boolean
BigInt
Short
Integer
Long
Float
Double
Date
Varchar
Bytes
BigDecimal
TimeStamp as Varchar

Performance

The Lambda function performs projection pushdown to decrease the data scanned by the query. `LIMIT` clauses reduce the amount of data scanned, but if you don't provide a predicate, you should expect `SELECT` queries with a `LIMIT` clause to scan at least 16 MB of data. The Vertica connector is resilient to throttling due to concurrency.

Passthrough queries

The Vertica connector supports passthrough queries. Passthrough queries use a table function to push your full query down to the data source for execution.

To use passthrough queries with Vertica, you can use the following syntax:

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
```

```
        ))
```

The following example query pushes down a query to a data source in Vertica. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

License information

By using this connector, you acknowledge the inclusion of third party components, a list of which can be found in the pom.xml file for this connector, and agree to the terms in the respective third party licenses provided in the LICENSE.txt file on GitHub.com.

Additional resources

For the latest JDBC driver version information, see the pom.xml file for the Vertica connector on GitHub.com.

For additional information about this connector, see the corresponding site on GitHub.com and Querying a Vertica data source in Amazon Athena using the Athena Federated Query SDK in the *AWS Big Data Blog*.

Create a data source connection

To use an Athena data source connector, you create the AWS Glue connection that stores the connection information about the connector and your data source. When you create the connection, you give the data source a name that you will use to reference your data source in your SQL queries.

You can create and configure a data source connection in Athena by using the console or the CreateDataCatalog API operations.

Topics

- Permissions to create and use a data source in Athena
- Use the Athena console to connect to a data source
- Use the AWS Serverless Application Repository to deploy a data source connector

- Create a VPC for a data source connector or AWS Glue connection
- Register your connection as a Glue Data Catalog
- Enable cross-account federated queries
- Update a data source connector

Permissions to create and use a data source in Athena

To create and use a data source, you need the following sets of permissions.

- AmazonAthenaFullAccess that provides full access to Amazon Athena and scoped access to the dependencies needed to enable querying, writing results, and data management. For more information, see AmazonAthenaFullAccess in the AWS Managed Policy Reference Guide.
- Permissions to call the CreateDataCatalog API. These permissions are only needed when you create a data source that integrates with Glue connections. For more information on the example policy, see ???.
- If you want to use Lake Formation fine-grain access control, in addition to the permissions listed above, you also need the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lakeformation:RegisterResource",
        "iam:ListRoles",
        "glue:CreateCatalog",
        "glue:GetCatalogs",
        "glue:GetCatalog"
      ],
      "Resource": "*"
    }
  ]
}
```

Use the Athena console to connect to a data source

You can use the Athena console to create and configure a data source connection.

To create a connection to a data source

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose **Create data source**.
5. For **Choose a data source**, choose the data source that you want Athena to query, considering the following guidelines:

   - Choose a connection option that corresponds to your data source. Athena has prebuilt data source connectors that you can configure for sources including MySQL, Amazon DocumentDB, and PostgreSQL.
   - Choose **S3 - AWS Glue Data Catalog** if you want to query data in Amazon S3 and you are not using an Apache Hive metastore or one of the other federated query data source options on this page. Athena uses the AWS Glue Data Catalog to store metadata and schema information for data sources in Amazon S3. This is the default (non-federated) option. For more information, see Use AWS Glue Data Catalog to connect to your data. For steps using this workflow, see Register and use data catalogs in Athena.
   - Choose **S3 - Apache Hive metastore** to query data sets in Amazon S3 that use an Apache Hive metastore. For more information about this option, see Connect Athena to an Apache Hive metastore.

- Choose **Custom or shared connector** if you want to create your own data source connector for use with Athena. For information about writing a data source connector, see Develop a data source connector using the Athena Query Federation SDK.

6. Choose **Next**.
7. On the **Enter data source details** page, for **Data source name**, use the name autogenerated name, or enter a unique name that you want to use in your SQL statements when you query the data source from Athena. The name can be up to 127 characters and must be unique within your account. It cannot be changed after you create it. Valid characters are a-z, A-Z, 0-9, _ (underscore), @ (at sign) and - (hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for data source names.
8. If the data source you choose integrates with AWS Glue connections.

   a. For **AWS Glue connection details**, enter the information required. A connection contains the properties that are required to connect to a particular data source. The properties required vary depending on the connection type. For more information on properties related to your connector, see Available data source connectors. For information about additional connection properties, see AWS Glue connection properties in the *AWS Glue User Guide*.

   > **Note**
   >
   > - When you update the Glue connection properties, the Lambda connector needs to be restarted to get the updated properties. To do this, edit the environment properties and save it without actually changing anything.
   > - When you update a Glue connection, the following properties will not automatically get updated in the corresponding Lambda function. You must manually update your Lambda function for these properties.
   >   - Lambda VPC configuration – `security_group_ids`, `subnet_ids`
   >   - Lambda execution role – `spill_bucket`, `secret_name`, `spill_kms_key_id`

   b. For **Lambda execution IAM role**, choose one of the following:

   - **Create and use a new execution role –** (Default) Athena creates an execution role that it will then use to access resources in AWS Lambda on your behalf. Athena requires this role to create your federated data source.

- **Use an existing execution role** – Use this option to choose an existing execution role. For this option, choose execution role that you want to use from **Execution role** drop-down.

9. If the data source you choose does not integrate with AWS Glue connections.

   a. For **Lambda function**, choose **Create Lambda function**. The function page for the connector that you chose opens in the AWS Lambda console. The page includes detailed information about the connector.

   b. Under **Application settings**, read the description for each application setting carefully, and then enter values that correspond to your requirements.

      The application settings that you see vary depending on the connector for your data source. The minimum required settings include:

      - **AthenaCatalogName** – A name, in lower case, for the Lambda function that indicates the data source that it targets, such as `cloudwatchlogs`.
      - **SpillBucket** – An Amazon S3 bucket in your account to store data that exceeds Lambda function response size limits.

      > **Note**
      >
      > Spilled data is not reused in subsequent executions and can be safely deleted. Athena does not delete this data for you. To manage these objects, consider adding an object lifecycle policy that deletes old data from your Amazon S3 spill bucket. For more information, see Managing your storage lifecycle in the Amazon S3 User Guide.

   c. Select **I acknowledge that this app creates custom IAM roles and resource policies**. For more information, choose the **Info** link.

   d. Choose **Deploy**. When the deployment is complete, the Lambda function appears in the **Resources** section in the Lambda console.

      After you deploy the data source connector to your account, you can connect Athena to it.

   e. Return to the **Enter data source details** page of the Athena console.

   f. In the **Connection details** section, choose the refresh icon next to the **Select or enter a Lambda function** search box.

   g. Choose the name of the function that you just created in the Lambda console. The ARN of the Lambda function displays.

10. (Optional) For **Tags**, add key-value pairs to associate with this data source. For more information about tags, see Tag Athena resources.
11. Choose **Next**.
12. On the **Review and create** page, review the data source details. To make changes, choose **Edit**.
13. Read the information in **Athena will create resources in your account**. If you agree, select **I acknowledge that Athena will create resources on my behalf**.
14. Choose **Create data source**. **Athena** will create the following resources for you.
    - Lambda execution IAM role
    - AWS Glue connection (only if the data source is compatible with AWS Glue Connections)
    - Lambda function

The **Data source details** section of the page for your data source shows information about your new connector. You can now use the connector in your Athena queries.

For information about using data connectors in queries, see Run federated queries.

Use the AWS Serverless Application Repository to deploy a data source connector

To deploy a data source connector, you can use the AWS Serverless Application Repository instead of using a AWS Glue connection.

> **Note**
>
> We recommend that you use the SAR only if you have a custom connector or require the use of an older connector. Otherwise, the use of the Athena console is recommended.

You can use the AWS Serverless Application Repository to find the connector that you want to use, provide the parameters that the connector requires, and then deploy the connector to your account. Then, after you deploy the connector, you use the Athena console to make the data source available to Athena.

Deploying the connector to Your Account

To use the AWS Serverless Application Repository to deploy a data source connector to your account

1. Sign in to the AWS Management Console and open the **Serverless App Repository**.
2. In the navigation pane, choose **Available applications**.
3. Select the option **Show apps that create custom IAM roles or resource policies**.
4. In the search box, type the name of the connector. For a list of prebuilt Athena data connectors, see Available data source connectors.
5. Choose the name of the connector. Choosing a connector opens the Lambda function's **Application details** page in the AWS Lambda console.
6. On the right side of the details page, for **Application settings**, fill in the required information. The minimum required settings include the following. For information about the remaining configurable options for data connectors built by Athena, see the corresponding Available connectors topic on GitHub.
    - **AthenaCatalogName –** A name for the Lambda function in lower case that indicates the data source that it targets, such as `cloudwatchlogs`.
    - **SpillBucket –** Specify an Amazon S3 bucket in your account to receive data from any large response payloads that exceed Lambda function response size limits.
7. Select **I acknowledge that this app creates custom IAM roles and resource policies**. For more information, choose the **Info** link.
8. At the bottom right of the **Application settings** section, choose **Deploy.** When the deployment is complete, the Lambda function appears in the **Resources** section in the Lambda console.

Making the connector available in Athena

Now you are ready to use the Athena console to make the data source connector available to Athena.

To make the data source connector available to Athena

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose **Create data source**.
5. For **Choose a data source**, choose the data source for which you created a connector in the AWS Serverless Application Repository. This tutorial uses **Amazon CloudWatch Logs** as the federated data source.
6. Choose **Next**.
7. On the **Enter data source details** page, for **Data source name**, enter the name that you want to use in your SQL statements when you query the data source from Athena (for example, `CloudWatchLogs`). The name can be up to 127 characters and must be unique within your account. It cannot be changed after you create it. Valid characters are a-z, A-Z, 0-9, _ (underscore), @ (at sign) and - (hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for data source names.
8. In the **Connection details** section, use the **Select or enter a Lambda function** box to choose the name of the function that you just created. The ARN of the Lambda function displays.
9. (Optional) For **Tags**, add key-value pairs to associate with this data source. For more information about tags, see [Tag Athena resources](#).
10. Choose **Next**.
11. On the **Review and create** page, review the data source details, and then choose **Create data source**.
12. The **Data source details** section of the page for your data source shows information about your new connector. You can now use the connector in your Athena queries.

For information about using data connectors in queries, see Run federated queries.

Create a VPC for a data source connector or AWS Glue connection

Some Athena data source connectors and AWS Glue connections require a VPC and a security group. This topic shows you how to create a VPC with a subnet and a security group for the VPC. As part of this process, you retrieve the IDs for the VPC, subnet, and security group that you create. These IDs are required when you configure your AWS Glue connection or data source connector for use with Athena.

To create a VPC for an Athena data source connector

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **Create VPC**.
3. On the **Create VPC** page, under **VPC Settings**, for **Resources to create**, choose **VPC and more**.
4. Under **Name tag auto-generation**, for **Auto-generate**, enter a value that will be used to generate name tags for all resources in your VPC.
5. Choose **Create VPC**.
6. When the process completes, choose **View VPC**.
7. In the **Details** section, for **VPC ID**, copy your VPC ID for later reference.

Now you are ready to retrieve the subnet ID for the VPC that you just created.

To retrieve your VPC subnet ID

1. In the VPC console navigation pane, choose **Subnets**.
2. Select the name of a subnet whose **VPC** column has the VPC ID that you noted.
3. In the **Details** section, for **Subnet ID**, copy your subnet ID for later reference.

Next, you create a security group for your VPC.

To create a security group for your VPC

1. In the VPC console navigation pane, choose **Security**, **Security Groups**.

2. Choose **Create security group**.
3. On the **Create security group** page, enter the following information:

   - For **Security group name**, enter a name for your security group.
   - For **Description**, enter a description for the security group. A description is required.
   - For **VPC**, choose the VPC ID of the VPC that you created for your data source connector.
   - For **Inbound rules** and **Outbound rules**, add any inbound and outbound rules that you require.
4. Choose **Create security group**.
5. On the **Details** page for the security group, copy the **Security group ID** for later reference.

Register your connection as a Glue Data Catalog

After you create your data source, you can use the Athena console to register your connection as a Glue Data Catalog. Once registered, you can manage your federated data catalog and enable fine-grained access control using Lake Formation. For more information, see [Creating a federated catalog](#).

You can register the following connectors to integrate with AWS Glue for fine-grained access control.

- Redshift
- BigQuery
- DynamoDB (Preview)
- Snowflake (Preview)
- MySQL
- PostgreSQL
- AWS CMDB
- Timestream

Prerequisites

Before you begin, you must complete the following prerequisites.

- Ensure that you have the roles and permissions needed to register locations. For more information, see the [Requirements for roles](#) in the AWS Lake Formation Developer Guide.

- Ensure that you have the required Lake Formation roles. For more information, see Prerequisites for connecting the Data Catalog to external data sources in the AWS Lake Formation Developer Guide.
- The role that you register in Glue must have the permissions as listed in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject"
            ],
            "Resource": [
                "s3://amzn-s3-demo-bucket/<Your_Spill_Prefix>/*",
                "s3://amzn-s3-demo-bucket/<Your_Spill_Prefix>"
            ]
        },
        {
            "Sid": "lambdainvoke",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "<Lambda_function_arn>"
        },
        {
            "Sid": "gluepolicy",
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:<region>:<account_id>:connection/<connection_name>",
                "arn:aws:glue:<region>:<account_id>:catalog"
            ]
        }
    ]
}
```

- You are responsible to determine and manage the appropriate data access. With fine-grain access controls on federated queries, it is recommended that you use the AmazonAthenaFullAccess managed policy. If you want to use your own policy, you must ensure that the users executing federated queries do not have access to the following resources.
    - `lambda:InvokeFunction` on the Lambda connector that is specified in Glue connection

- Spill bucket location access in IAM
- Access to the Glue connection associated with your federated catalog
- Lake Formation Role in IAM

Register your connection using console

To register your connection as a Glue Data Catalog

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Data sources and catalogs**.
3. From the **Data sources** list, choose the data source that you created to open the **Data source details** page.
4. Choose **Get started with AWS Lake Formation**.

> **Note**
>
> After you choose this option, you must manage your Lambda function on your own. Athena will not delete your Lambda function.

5. For **Data catalog name** provide a unique name for your catalog.
6. Choose the **Lake Formation IAM role** that grants permission to Lake Formation to invoke the Lambda function. Make sure the role has the permissions as shown in the example.
7. In the text box, type **confirm** to delete the Athena data source, replace it with a Glue data catalog registration.

> **Note**
>
> This action will delete your Athena data source and create a new Glue Data Catalog in its place. After this process is completed, you may need to update queries that access the data source to refer to the newly created Glue data catalog instead.

8. Choose **Create catalog and go to Lake Formation**. This opens the Lake Formation console where you can manage the catalog and grant permissions to users on catalogs, databases and tables.

Enable cross-account federated queries

Federated query allows you to query data sources other than Amazon S3 using data source connectors deployed on AWS Lambda. The cross-account federated query feature allows the Lambda function and the data sources that are to be queried to be located in different accounts.

> **Note**
>
> Use this method only if you have not registered your federated data source with the AWS Glue Data Catalog. If you have registered your data source with the AWS Glue Data Catalog, use the AWS Glue Data Catalog cross account features and permissions model. For more information, see Granting cross-account access in the *AWS Glue User Guide*.

As a data administrator, you can enable cross-account federated queries by sharing your data connector with a data analyst's account or, as a data analyst, by using a shared Lambda ARN from a data administrator to add to your account. When configuration changes are made to a connector in the originating account, the updated configuration is automatically applied to the shared instances of the connector in other user's accounts.

Considerations and limitations

- The cross-account federated query feature is available for non-Hive metastore data connectors that use a Lambda-based data source.
- The feature is not available for the AWS Glue Data Catalog data source type. For information about cross-account access to AWS Glue Data Catalogs, see Configure cross-account access to AWS Glue data catalogs.
- If the response from your connector's Lambda function exceeds the Lambda response size limit of 6MB, Athena automatically encrypts, batches, and spills the response to an Amazon S3 bucket that you configure. The entity running the Athena query must have access to the spill location in order for Athena to read the spilled data. We recommend that you set an Amazon S3 lifecycle policy to delete objects from the spill location since the data is not needed after the query completes.
- Using federated queries across AWS Regions is not supported.

Required permissions

To set up the required permissions, actions must be taken in both Account A and Account B.

Actions for Account A

For data administrator Account A to share a Lambda function with data analyst Account B, Account B requires Lambda invoke function and spill bucket access. Accordingly, Account A should add a resource-based policy to the Lambda function and principal access to its spill bucket in Amazon S3.

1. The following policy grants Lambda invoke function permissions to Account B on a Lambda function in Account A.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountInvocationStatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": ["arn:aws:iam::account-B-id:user/username"]
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:aws-region:account-A-id:function:lambda-function-
name"
    }
  ]
}
```

2. The following policy allows spill bucket access to the principal in Account B.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::account-B-id:user/username"]
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
             ],
            "Resource": [
                "arn:aws:s3:::spill-bucket",
                "arn:aws:s3:::spill-bucket/*"
            ]
```

```
        }
      ]
 }
```

3. If the Lambda function is encrypting the spill bucket with a AWS KMS key instead of the default encryption offered by the federation SDK, the AWS KMS key policy in Account A must grant access to the user in Account B, as in the following example.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal":
    {
       "AWS": ["arn:aws:iam::account-B-id:user/username"]
    },
    "Action": [ "kms:Decrypt" ],
    "Resource": "*" // Resource policy that gets placed on the KMS key.
 }
```

Actions for Account B

For Account A to share its connector with Account B, Account B must create a role called `AthenaCrossAccountCreate-account-A-id` that Account A assumes by calling the AWS Security Token Service AssumeRole API action.

1. Use the IAM console or the AWS CLI to create the `AthenaCrossAccountCreate-account-A-id` role in as a custom trust policy role. A custom trust policy delegates access and allows others to perform actions in your AWS account. For steps, see Create a role using custom trust policies in the *IAM User Guide*.

   The trust relationship should have a principal object in which the key is `AWS` and the value is the ARN of Account A, as in the following example.

```
...
"Principal":
{
   "AWS": ["arn:aws:iam::account-A-id:user/username"]
},
...
```

2. Also in Account B, create a policy like the following that allows the `CreateDataCatalog` action.

```
{
 "Effect": "Allow",
 "Action": "athena:CreateDataCatalog",
 "Resource": "arn:aws:athena:*:account-B-id:datacatalog/*"
}
```

3. Add the policy that allows the `CreateDataCatalog` action to the `AthenaCrossAccountCreate-account-A-id` role that you created using Account B.

Sharing a data source in Account A with Account B

After permissions are in place, you can use the **Data sources and catalogs** page in the Athena console to share a data connector in your account (Account A) with another account (Account B). Account A retains full control and ownership of the connector. When Account A makes configuration changes to the connector, the updated configuration applies to the shared connector in Account B.

> **Note**
>
> You can only share a Lambda type data source and cannot share data sources that use AWS Glue connections. For more information, see ???.

To share a Lambda data source in Account A with Account B

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. Choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose the link of the connector that you want to share.
5. On the details page for a Lambda data source, from the **Actions** menu in the top right corner, choose **Share**.
6. In the **Share** ***`Lambda-name`*** **with another account?** dialog box, enter the required information.
   - For **Data source name**, enter the name of the copied data source as you want it to appear in the other account.
   - For **Account ID**, enter the ID of the account with which you want to share your data source (in this case, Account B).
7. Choose **Share**. The shared data connector that you specified is created in Account B. Configuration changes to the connector in Account A apply to the connector in Account B.

Adding a shared data source from Account A to Account B

As a data analyst, you may be given the ARN of a connector to add to your account from a data administrator. You can use the **Data sources and catalogs** page of the Athena console to add the Lambda ARN provided by your administrator to your account.

To add the Lambda ARN of a shared data connector to your account

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the navigation pane is not visible, choose the expansion menu on the left.

3. Choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose **Create data source**.
5. On the **Choose a data source** page, choose **Custom or shared connector**.
6. Choose **Next**.
7. On the **Enter data source details** page, in the **Connection details** section, for **Select or enter a Lambda function**, enter the Lambda ARN of Account A.
8. Choose **Next**.
9. On the **Review and create** page, choose **Create data source**.

Troubleshooting

If you receive an error message that Account A does not have the permissions to assume a role in Account B, make sure that the name of the role created in Account B is spelled correctly and that it has the proper policy attached.

Update a data source connector

Athena recommends that you regularly update the data source connectors that you use to the latest version to take advantage of new features and enhancements. Updating a data source connector includes the following steps:

- Find the latest Athena Query Federation version
- Find and note resource names
- Find the version of the connector that you are using
- Deploy the new version of your connector

Find the latest Athena Query Federation version

The latest version number of Athena data source connectors corresponds to the latest Athena Query Federation version. In certain cases, the GitHub releases can be slightly newer than what is available on the AWS Serverless Application Repository (SAR).

To find the latest Athena Query Federation version number

1. Visit the GitHub URL https://github.com/awslabs/aws-athena-query-federation/releases/latest.

2. Note the release number in the main page heading in the following format:

   Release v *year.week_of_year.iteration_of_week* of Athena Query Federation

   For example, the release number for **Release v2023.8.3 of Athena Query Federation** is 2023.8.3.

Find and note resource names

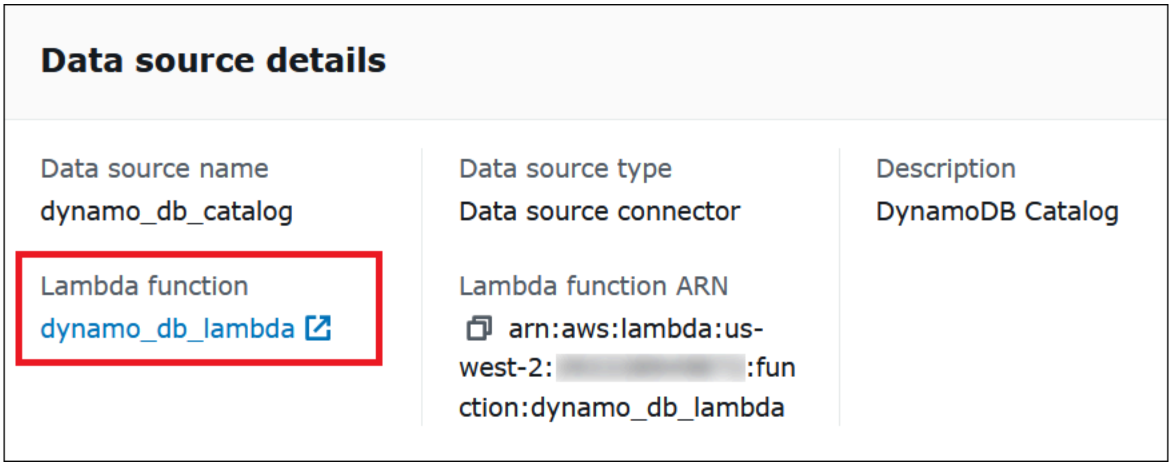
In preparation for the upgrade, you must find and note the following information:

1. The Lambda function name for the connector.
2. The Lambda function environment variables.
3. The Lambda application name, which manages the Lambda function for the connector.

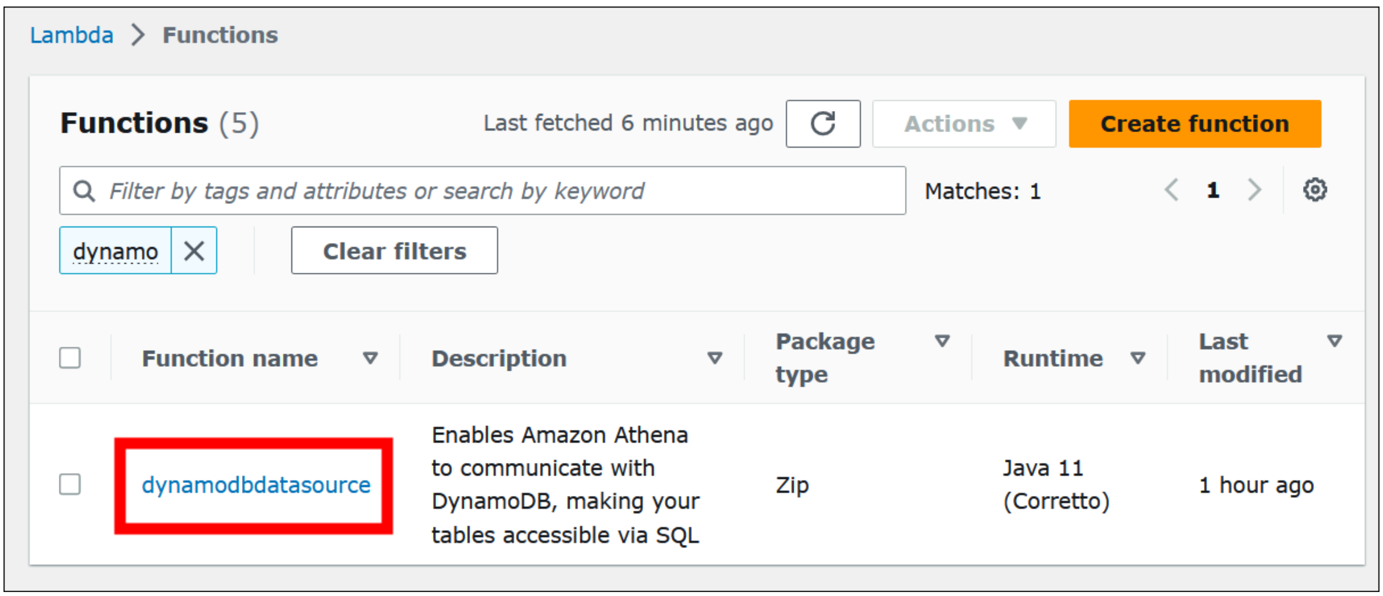
To find resource names from the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Data sources and catalogs**.
4. In the **Data source name** column, choose the link to the data source for your connector.
5. In the **Data source details** section, under **Lambda function**, choose the link to your Lambda function.



6. On the **Functions** page, in the **Function name** column, note the function name for your connector.



7. Choose the function name link.
8. Under the **Function overview** section, choose the **Configuration** tab.
9. In the pane on the left, choose **Environment variables**.
10. In the **Environment variables** section, make a note of the keys and their corresponding values.
11. Scroll to the top of the page.
12. In the message **This function belongs to an application. Click here to manage it**, choose the **Click here** link.
13. On the **serverlessrepo-*`your_application_name`*** page, make a note of your application name without **serverlessrepo**. For example, if the application name is **serverlessrepo-DynamoDbTestApp**, then your application name is **DynamoDbTestApp**.

14. Stay on the Lambda console page for your application, and then continue with the steps in **Finding the version of the connector that you are using**.

Find the version of the connector that you are using

Follow these steps to find the version of the connector that you are using.

To find the version of the connector that you are using

1. On the Lambda console page for your Lambda application, choose the **Deployments** tab.
2. On the **Deployments** tab, expand **SAM template**.
3. Search for **CodeUri**.
4. In the **Key** field under **CodeUri**, find the following string:

```
applications-connector_name-
versions-year.week_of_year.iteration_of_week/hash_number
```

   The following example shows a string for the CloudWatch connector:

```
applications-AthenaCloudwatchConnector-versions-2021.42.1/15151159...
```

5. Record the value for *year.week_of_year.iteration_of_week* (for example, **2021.42.1**). This is the version for your connector.

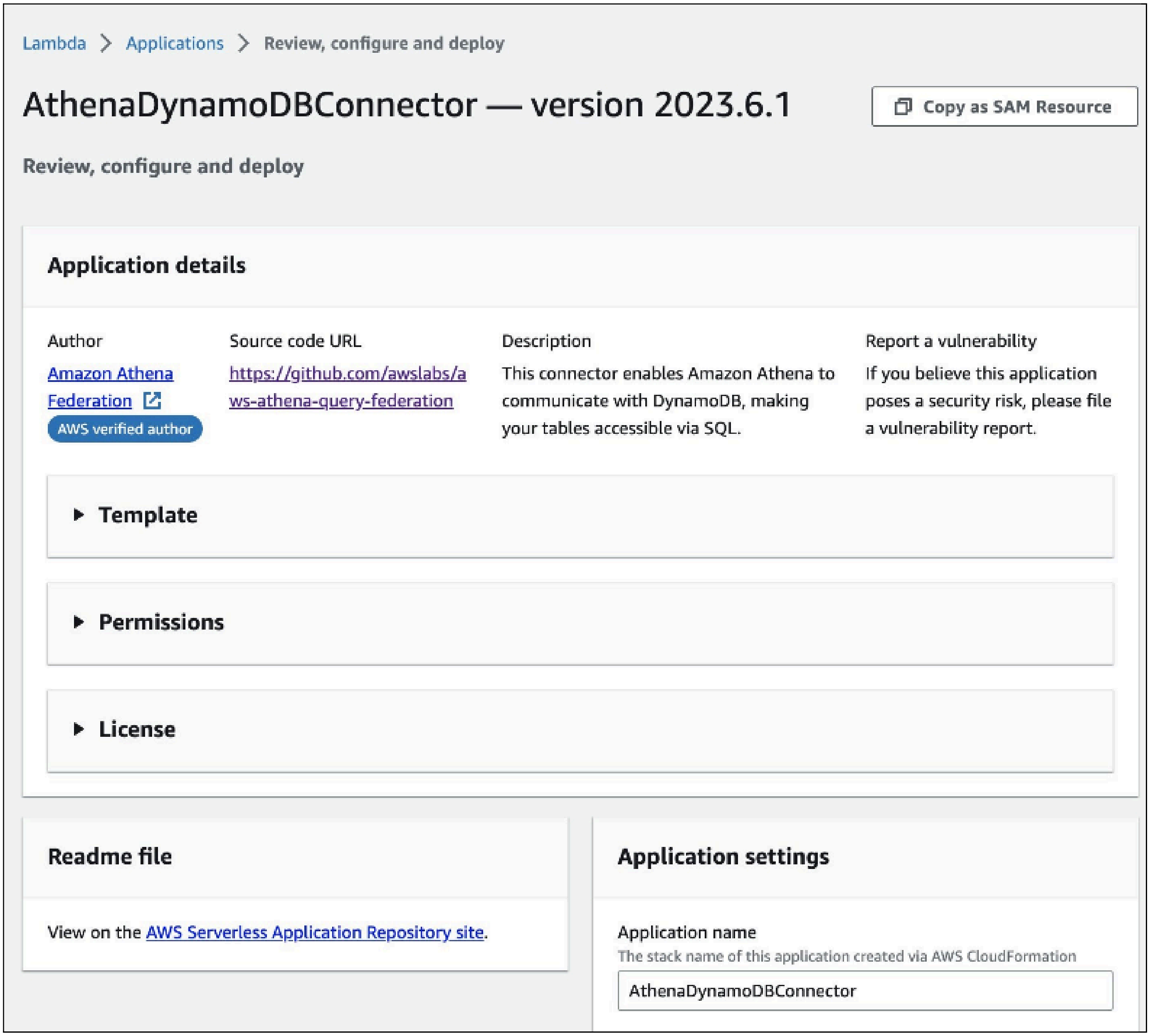
Deploy the new version of your connector

Follow these steps to deploy a new version of your connector.

To deploy a new version of your connector

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

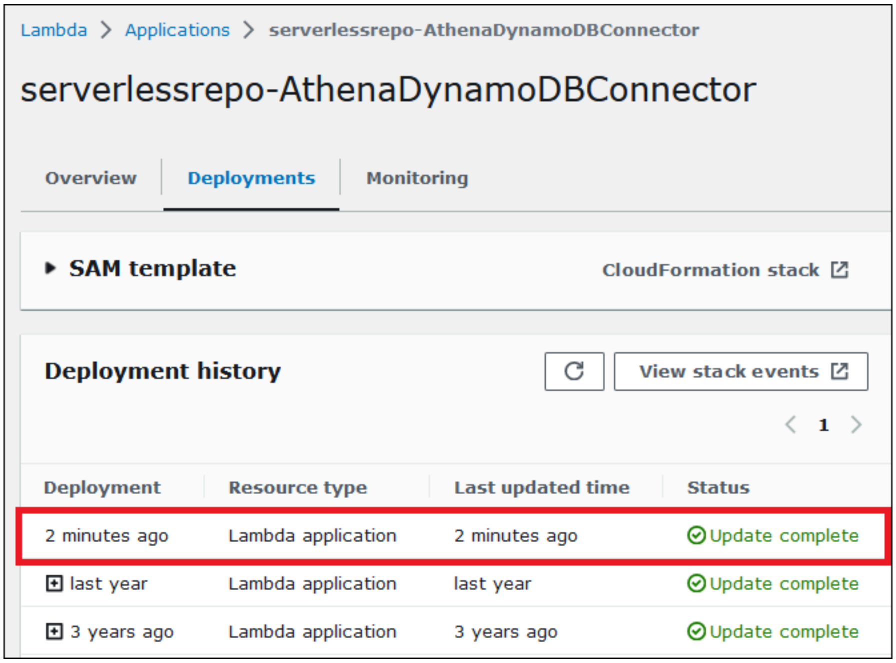
3. In the navigation pane, choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose **Create data source**.
5. Choose the data source that you want to upgrade, and then choose **Next**.
6. In the **Connection details** section, choose **Create Lambda function**. This opens the Lambda console where you will be able to deploy your updated application.



7. Because you are not actually creating a new data source, you can close the Athena console tab.
8. On the Lambda console page for the connector, perform the following steps:
   a. Ensure that you have removed the **serverlessrepo-** prefix from your application name, and then copy the application name to the **Application name** field.
   b. Copy your Lambda function name to the **AthenaCatalogName** field. Some connectors call this field **LambdaFunctionName**.
   c. Copy the environment variables that you recorded into their corresponding fields.

9. Select the option **I acknowledge that this app creates custom IAM roles and resource policies**, and then choose **Deploy**.
10. To verify that your application has been updated, choose the **Deployments** tab.

    The **Deployment history** section shows that your update is complete.



11. To confirm the new version number, you can expand **SAM template** as before, find **CodeUri**, and check the connector version number in the **Key** field.

You can now use your updated connector to create Athena federated queries.

Edit or delete a data source connection

You can use the Athena console to update the description, host, port, database, and other properties for an existing connection. You can also delete the data sources from Athena console.

Edit a data source connection

To edit a data source connection

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose the data source connection that you want to edit.
5. For **AWS Glue connection details**, choose **Edit**.
6. Choose **Next**.
7. On the **Edit <connection-name>** page, update the information as required. Available properties depend on the connection type.

> **Note**
>
> When you update connection properties for secrets, spill location, or AWS KMS key ID, make sure the Lambda execution role still has access to the updated resources. For more information, see Viewing and updating permissions in the execution role in the AWS Lambda Developer Guide.

- **Description** – Edit the description for your connection.
- **Host** – Edit the host name for your database.
- **Port** – Edit the port number for your database.
- **Database** – Edit the name of your database.
- **JDBC parameters** – Edit any additional JDBC parameters required for your connection.

- **Secret** – Choose or create a secret from AWS Secrets Manager. Use AWS secrets to avoid hardcoding sensitive information in your JDBC connection string. For more information, see What is AWS Secrets Manager? For information about creating a secret in Secrets Manager, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

  To use AWS Secrets Manager with Athena federated queries, you must configure an Amazon VPC private endpoint for Secrets Manager. For more information, see Create a Secrets Manager VPC private endpoint in the *AWS Secrets Manager User Guide*.

- **Spill location in Amazon S3** – Choose or create an Amazon S3 bucket location in your account to store data that exceeds Lambda function response size limits.

> **Note**
>
> Spilled data is not reused in subsequent executions and can be safely deleted after 12 hours. Athena does not delete this data for you. To manage these objects, consider adding an object lifecycle policy that deletes old data from your Amazon S3 spill bucket. For more information, see Managing your storage lifecycle in the *Amazon S3 User Guide*.

- **Encryption for query results in S3** – Choose one of the following:
  - (Default) **Use a randomly generated key** – Data that is spilled to Amazon S3 is encrypted using the AES-GCM authenticated encryption mode and a randomly generated key.
  - **Use an AWS KMS key** – Choose or create a stronger, AWS KMS generated encryption key. For more information, see Creating keys in the *AWS Key Management Service Developer Guide*.
  - **Turn off** – Do not encrypt spill data.
- **Networking settings** – Some connections require a virtual private cloud (VPC). Choose or create a VPC that has the data store that you want to access, a subnet, and one or more security groups. For more information, see Create a VPC for a data source connector or AWS Glue connection.

> **Note**
>
> - After you update connection properties for resources such as secrets, spill location, or AWS KMS key ID, make sure that the Lambda execution role continues to have access to the updated resources.

- After you update the network settings for your connection, make sure you update the Lambda function with the same settings to make your connection compatible with the data source.

For information about additional connection properties, see [AWS Glue connection properties](#) in the *AWS Glue User Guide* or [Available data source connectors](#) in the *Amazon Athena User Guide*.

8. Choose **Save**.

The **AWS Glue connection details** section of the page for your data source shows the updated information for your connector.

Delete a data source

When you delete a data source, it only deletes the Athena data source and does not delete resources like the Glue connections, IAM execution role, and Lambda function.

To delete a data source

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Data sources and catalogs**.
3. On the **Data sources and catalogs** page, choose the data source that you want to delete.
4. Choose **Delete**.
5. On the **Delete data source** page, type *confirm* to confirm deletion and the choose **Delete**. It may take some time before the data source deletion completes. You get a success alert once the data source is deleted.

Run federated queries

After you have configured one or more data connectors and deployed them to your account, you can use them in your Athena queries.

Query a single data source

The examples in this section assume that you have configured and deployed the Amazon Athena CloudWatch connector to your account. Use the same approach to query when you use other connectors.

To create an Athena query that uses the CloudWatch connector

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena query editor, create a SQL query that uses the following syntax in the `FROM` clause.

   ```
   MyCloudwatchCatalog.database_name.table_name
   ```

Examples

The following example uses the Athena CloudWatch connector to connect to the `all_log_streams` view in the `/var/ecommerce-engine/order-processor` CloudWatch Logs Log group. The `all_log_streams` view is a view of all the log streams in the log group. The example query limits the number of rows returned to 100.

```
SELECT *
FROM "MyCloudwatchCatalog"."/var/ecommerce-engine/order-processor".all_log_streams
LIMIT 100;
```

The following example parses information from the same view as the previous example. The example extracts the order ID and log level and filters out any message that has the level `INFO`.

```
SELECT
    log_stream as ec2_instance,
    Regexp_extract(message '.*orderId=(\d+) .*', 1) AS orderId,
    message AS order_processor_log,
    Regexp_extract(message, '(.*):.*', 1) AS log_level
FROM MyCloudwatchCatalog."/var/ecommerce-engine/order-processor".all_log_streams
WHERE Regexp_extract(message, '(.*):.*', 1) != 'INFO'
```

Query multiple data sources

As a more complex example, imagine an e-commerce company that uses the following data sources to store data related to customer purchases:

- [Amazon RDS for MySQL] to store product catalog data
- [Amazon DocumentDB] to store customer account data such as email and shipping addresses
- [Amazon DynamoDB] to store order shipment and tracking data

Imagine that a data analyst for this e-commerce application learns that shipping time in some regions has been impacted by local weather conditions. The analyst wants to know how many orders are delayed, where the affected customers are located, and which products are most affected. Instead of investigating the sources of information separately, the analyst uses Athena to join the data together in a single federated query.

Example

```
SELECT
     t2.product_name AS product,
     t2.product_category AS category,
     t3.customer_region AS region,
     count(t1.order_id) AS impacted_orders
FROM my_dynamodb.default.orders t1
JOIN my_mysql.products.catalog t2 ON t1.product_id = t2.product_id
JOIN my_documentdb.default.customers t3 ON t1.customer_id = t3.customer_id
WHERE
     t1.order_status = 'PENDING'
     AND t1.order_date between '2022-01-01' AND '2022-01-05'
GROUP BY 1, 2, 3
ORDER BY 4 DESC
```

Query federated views

When querying federated sources, you can use views to obfuscate the underlying data sources or hide complex joins from other analysts who query the data.

Considerations and limitations

- Federated views require Athena engine version 3.
- Federated views are stored in AWS Glue, not with the underlying data source.
- Federated views are not supported on data sources that are [registered as a Glue Data Catalog].
- Views created with federated catalogs must use fully qualified name syntax, as in the following example:

```
"ddbcatalog"."default"."customers"
```

- Users who run queries on federated sources must have permission to query the federated sources.
- The `athena:GetDataCatalog` permission is required for federated views. For more information, see Allow access to Athena Federated Query: Example policies .

Examples

The following example creates a view called `customers` on data stored in a federated data source.

Example

```
CREATE VIEW customers AS
SELECT *
FROM my_federated_source.default.table
```

The following example query shows a query that references the `customers` view instead of the underlying federated data source.

Example

```
SELECT id, SUM(order_amount)
FROM customers
GROUP by 1
ORDER by 2 DESC
LIMIT 50
```

The following example creates a view called `order_summary` that combines data from a federated data source and from an Amazon S3 data source. From the federated source, which has already been created in Athena, the view uses the `person` and `profile` tables. From Amazon S3, the view uses the `purchase` and `payment` tables. To refer to Amazon S3, the statement uses the keyword `awsdatacatalog`. Note that the federated data source uses the fully qualified name syntax *federated_source_name*.*federated_source_database*.*federated_source_table*.

Example

```
CREATE VIEW default.order_summary AS
SELECT *
```

```
FROM federated_source_name.federated_source_database."person" p
    JOIN federated_source_name.federated_source_database."profile" pr ON pr.id = p.id
    JOIN awsdatacatalog.default.purchase i ON p.id = i.id
    JOIN awsdatacatalog.default.payment pay ON pay.id = p.id
```

Additional resources

- For an example of a federated view that is decoupled from its originating source and is available for on-demand analysis in a multi-user model, see [Extend your data mesh with Amazon Athena and federated views]() in the *AWS Big Data Blog*.
- For more information about working with views in Athena, see [Work with views]().

Use federated passthrough queries

In Athena, you can run queries on federated data sources using the query language of the data source itself and push the full query down to the data source for execution. These queries are called *passthrough queries*. To run passthrough queries, you use a table function in your Athena query. You include the passthrough query to run on the data source in one of the arguments to the table function. Pass through queries return a table that you can analyze using Athena SQL.

Supported connectors

The following Athena data source connectors support passthrough queries.

- Azure Data Lake Storage
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- CloudWatch
- Db2
- Db2 iSeries
- DocumentDB
- DynamoDB
- HBase
- Google BigQuery
- Hortonworks

- MySQL
- Neptune
- OpenSearch
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server
- Teradata
- Timestream
- Vertica

Considerations and limitations

When using passthrough queries in Athena, consider the following points:

- Query passthrough is supported only for Athena SELECT statements or read operations.
- Query performance can vary depending on the configuration of the data source.
- Query passthrough does not support Lake Formation fine-grained access control.
- Passthrough queries are not supported on data sources that are registered as a Glue Data Catalog.

Syntax

The general Athena query passthrough syntax is as follows.

```
SELECT * FROM TABLE(catalog.system.function_name(arg1 => 'arg1Value'[, arg2 =>
 'arg2Value', ...]))
```

Note the following:

- **catalog** – The target Athena federated connector name or data catalog name.
- **system** – The namespace that contains the function. All Athena connector implementations use this namespace.

- **function_name** – The name of the function that pushes the passthrough query down to the data source. This is often called `query`. The combination `catalog.system.function_name` is the full resolution path for the function.
- **arg1, arg2, and so on** – Function arguments. The user must pass these to the function. In most cases, this is the query string that is passed down to the data source.

For most data sources, the first and only argument is `query` followed by the arrow operator => and the query string.

```
SELECT * FROM TABLE(catalog.system.query(query => 'query string'))
```

For simplicity, you can omit the optional named argument `query` and the arrow operator =>.

```
SELECT * FROM TABLE(catalog.system.query('query string'))
```

You can further simplify the query by removing the `catalog` name if the query is run within the context of the target catalog.

```
SELECT * FROM TABLE(system.query('query string'))
```

If the data source requires more than the query string, use named arguments in the order expected by the data source. For example, the expression `arg1 => 'arg1Value'` contains the first argument and its value. The name `arg1` is specific to the data source and can differ from connector to connector.

```
SELECT * FROM TABLE(
        system.query(
            arg1 => 'arg1Value',
            arg2 => 'arg2Value',
            arg3 => 'arg3Value'
        ));
```

The above can also be simplified by omitting the argument names. However, you must follow the order of the method's signature. See each connector's documentation for more information about the function's signature.

```
SELECT * FROM TABLE(catalog.system.query('arg1Value', 'arg2Value', 'arg3Value'))
```

You can run multiple passthrough queries across different Athena connectors by utilizing the full function resolution path, as in the following example.

```
SELECT c_customer_sk
    FROM TABLE (postgresql.system.query('select * from customer limit 10'))
UNION
SELECT c_customer_sk
    FROM TABLE(dynamodb.system.query('select * from customer')) LIMIT 10
```

You can use passthrough queries as part of a federated view. The same limitations apply. For more information, see Query federated views.

```
CREATE VIEW catalog.database.ViewName AS
    SELECT * FROM TABLE (
        catalog.system.query('query')
    )
```

For information about the exact syntax to use with a particular connector, see the individual connector documentation.

Quotation mark usage

Argument values, including the query string that you pass, must be enclosed in single quotes, as in the following example.

```
SELECT * FROM TABLE(system.query(query => 'SELECT * FROM testdb.persons LIMIT 10'))
```

When the query string is surrounded by double quotes, the query fails. The following query fails with the error message COLUMN_NOT_FOUND: line 1:43: Column 'select * from testdb.persons limit 10' cannot be resolved.

```
SELECT * FROM TABLE(system.query(query => "SELECT * FROM testdb.persons LIMIT 10"))
```

To escape a single quote, add a single quote to the original (for example, `terry's_group` to `terry''s_group`).

Examples

The following example query pushes down a query to a data source. The query selects all columns in the `customer` table, limiting the results to 10.

```
SELECT * FROM TABLE(
        catalog.system.query(
            query => 'SELECT * FROM customer LIMIT 10;'
        ))
```

The following statement runs the same query, but eliminates the optional named argument `query` and the arrow operator =>.

```
SELECT * FROM TABLE(
        catalog.system.query(
            'SELECT * FROM customer LIMIT 10;'
        ))
```

This can also be encapsulated within a federated view for ease of reuse. When used with a view, you must use the full function resolution path.

```
CREATE VIEW AwsDataCatalog.default.example_view AS
    SELECT * FROM TABLE (
        catalog.system.query('SELECT * FROM customer LIMIT 10;')
    )
```

Opt out of query passthrough

To disable passthrough queries, add a Lambda environment variable named `enable_query_passthrough` and set it to `false`.

Understand federated table name qualifiers

Athena uses the following terms to refer to hierarchies of data objects:

- **Data source** – a group of databases
- **Database** – a group of tables
- **Table** – data organized as a group of rows or columns

Sometimes these objects are also referred to with alternate but equivalent names such as the following:

- A data source is sometimes referred to as a *catalog*.
- A database is sometimes referred to as a *schema*.

Terms in federated data sources

When you query federated data sources, note that the underlying data source might not use the same terminology as Athena. Keep this distinction in mind when you write your federated queries. The following sections describe how data object terms in Athena correspond to those in federated data sources.

Amazon Redshift

An Amazon Redshift *database* is a group of Redshift *schemas* that contains a group of Redshift *tables*.

Athena	Redshift
Redshift data source	A Redshift connector Lambda function configured to point to a Redshift `database`.
`data_source.database.table`	`database.schema.table`

Example query

```
SELECT * FROM
Athena_Redshift_connector_data_source.Redshift_schema_name.Redshift_table_name
```

For more information about this connector, see Amazon Athena Redshift connector.

Cloudera Hive

An Cloudera Hive *server* or *cluster* is a group of Cloudera Hive *databases* that contains a group of Cloudera Hive *tables*.

Athena	Hive
Cloudera Hive data source	Cloudera Hive connector Lambda function configured to point to a Cloudera Hive `server`.
`data_source.database.table`	`server.database.table`

Example query

```
SELECT * FROM
Athena_Cloudera_Hive_connector_data_source.Cloudera_Hive_database_name.Cloudera_Hive_table_name
```

For more information about this connector, see Amazon Athena Cloudera Hive connector.

Cloudera Impala

An Impala *server* or *cluster* is a group of Impala *databases* that contains a group of Impala *tables*.

Athena	Impala
Impala data source	Impala connector Lambda function configured to point to an Impala `server`.
`data_source.database.table`	`server.database.table`

Example query

```
SELECT * FROM
Athena_Impala_connector_data_source.Impala_database_name.Impala_table_name
```

For more information about this connector, see Amazon Athena Cloudera Impala connector.

MySQL

A MySQL *server* is a group of MySQL *databases* that contains a group of MySQL *tables*.

Athena	MySQL
MySQL data source	MySQL connector Lambda function configured to point to a MySQL `server`.
`data_source.database.table`	`server.database.table`

Example query

```
SELECT * FROM
Athena_MySQL_connector_data source.MySQL_database_name.MySQL_table_name
```

For more information about this connector, see Amazon Athena MySQL connector.

Oracle

An Oracle *server* (or *database*) is a group of Oracle *schemas* that contains a group of Oracle *tables*.

Athena	Oracle
Oracle data source	Oracle connector Lambda function configured to point to an Oracle `server`.
`data_source.database.table`	`server.schema.table`

Example query

```
SELECT * FROM
Athena_Oracle_connector_data_source.Oracle_schema_name.Oracle_table_name
```

For more information about this connector, see Amazon Athena Oracle connector.

Postgres

A Postgres *server* (or *cluster*) is a group of Postgres *databases*. A Postgres *database* is a group of Postgres *schemas* that contains a group of Postgres *tables*.

Athena	Postgres
Postgres data source	Postgres connector Lambda function configured to point to a Postgres `server` and `database`.
`data_source.database.table`	`server.database.schema.table`

Example query

```
SELECT * FROM
Athena_Postgres_connector_data_source.Postgres_schema_name.Postgres_table_name
```

For more information about this connector, see Amazon Athena PostgreSQL connector.

Develop a data source connector using the Athena Query Federation SDK

To write your own data source connectors, you can use the Athena Query Federation SDK. The Athena Query Federation SDK defines a set of interfaces and wire protocols that you can use to enable Athena to delegate portions of its query execution plan to code that you write and deploy. The SDK includes a connector suite and an example connector.

Custom connectors do not use Glue Connections to centralize configuration properties in Glue. Connection configuration is done through Lambda.

You can also customize Amazon Athena's prebuilt connectors for your own use. You can modify a copy of the source code from GitHub and then use the Connector publish tool to create your own AWS Serverless Application Repository package. After you deploy your connector in this way, you can use it in your Athena queries.

For information about how to download the SDK and detailed instructions for writing your own connector, see Example Athena connector on GitHub.

Work with data source connectors for Apache Spark

Some Athena data source connectors are available as Spark DSV2 connectors. The Spark DSV2 connector names have a `-dsv2` suffix (for example, `athena-dynamodb-dsv2`).

Following are the currently available DSV2 connectors, their Spark `.format()` class name, and links to their corresponding Amazon Athena Federated Query documentation:

DSV2 connector	Spark .format() class name	Documentation
athena-cloudwatch-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.CloudwatchTableProvider`	CloudWatch

DSV2 connector	Spark .format() class name	Documentation
athena-cloudwatch-metrics-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.metrics.CloudwatchMetricsTableProvider`	CloudWatch metrics
athena-aws-cmdb-dsv2	`com.amazonaws.athena.connectors.dsv2.aws.cmdb.AwsCmdbTableProvider`	CMDB
athena-dynamodb-dsv2	`com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider`	DynamoDB

To download `.jar` files for the DSV2 connectors, visit the Amazon Athena Query Federation DSV2 GitHub page and see the **Releases**, **Release *<version>***, **Assets** section.

Specify the jar to Spark

To use the Athena DSV2 connectors with Spark, you submit the `.jar` file for the connector to the Spark environment that you are using. The following sections describe specific cases.

Athena for Spark

For information on adding custom `.jar` files and custom configuration to Amazon Athena for Apache Spark, see Use Spark properties to specify custom configuration.

General Spark

To pass in the connector `.jar` file to Spark, use the `spark-submit` command and specify the `.jar` file in the `--jars` option, as in the following example:

```
spark-submit \
  --deploy-mode cluster \
```

```
  --jars https://github.com/awslabs/aws-athena-query-federation-dsv2/releases/
download/some_version/athena-dynamodb-dsv2-some_version.jar
```

Amazon EMR Spark

In order to run a `spark-submit` command with the `--jars` parameter on Amazon EMR, you must add a step to your Amazon EMR Spark cluster. For details on how to use `spark-submit` on Amazon EMR, see [Add a Spark step](#) in the *Amazon EMR Release Guide*.

AWS Glue ETL Spark

For AWS Glue ETL, you can pass in the `.jar` file's GitHub.com URL to the `--extra-jars` argument of the `aws glue start-job-run` command. The AWS Glue documentation describes the `--extra-jars` parameter as taking an Amazon S3 path, but the parameter can also take an HTTPS URL. For more information, see [Job parameter reference](#) in the *AWS Glue Developer Guide*.

Query the connector on Spark

To submit the equivalent of your existing Athena federated query on Apache Spark, use the `spark.sql()` function. For example, suppose you have the following Athena query that you want to use on Apache Spark.

```
SELECT somecola, somecolb, somecolc
FROM ddb_datasource.some_schema_or_glue_database.some_ddb_or_glue_table
WHERE somecola > 1
```

To perform the same query on Spark using the Amazon Athena DynamoDB DSV2 connector, use the following code:

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())

dynamoDf.createOrReplaceTempView("ddb_spark_table")

spark.sql('''
SELECT somecola, somecolb, somecolc
FROM ddb_spark_table
```

```
WHERE somecola > 1
''')
```

Specify parameters

The DSV2 versions of the Athena data source connectors use the same parameters as the corresponding Athena data source connectors. For parameter information, refer to the documentation for the corresponding Athena data source connector.

In your PySpark code, use the following syntax to configure your parameters.

```
spark.read.option("athena.connectors.conf.parameter", "value")
```

For example, the following code sets the Amazon Athena DynamoDB connector `disable_projection_and_casing` parameter to `always`.

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .option("athena.connectors.conf.disable_projection_and_casing", "always")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())
```

Use Amazon DataZone in Athena

You can use Amazon DataZone to share, search, and discover data at scale across organizational boundaries. DataZone simplifies your experience across AWS analytics services like Athena, AWS Glue, and AWS Lake Formation. For example, if you have petabytes of data in different data sources, you can use Amazon DataZone to build business use case based groupings of people, data and tools. For more information, see What is Amazon DataZone?.

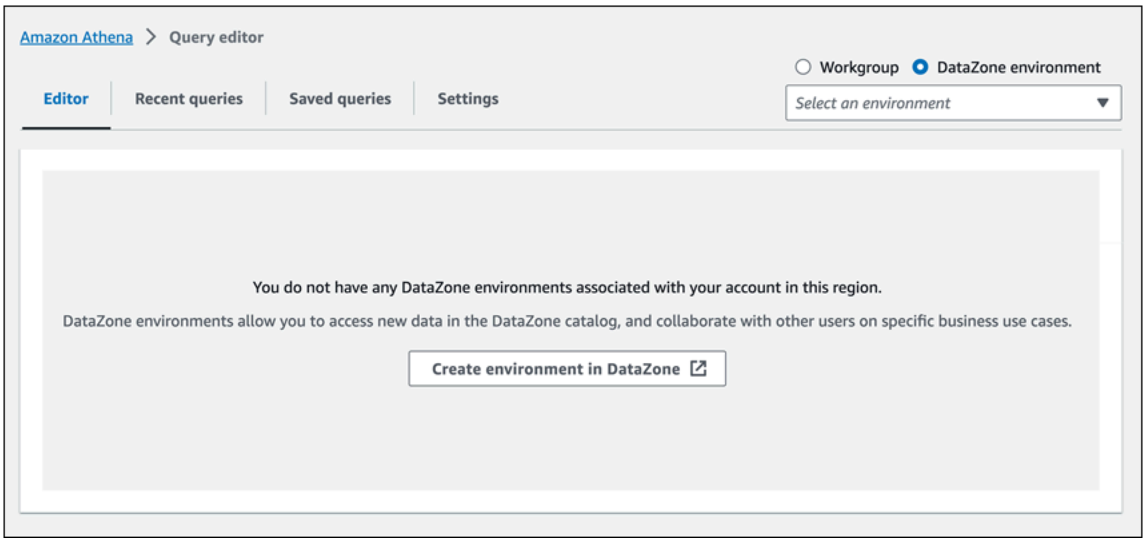
In Athena, you can use the query editor to access and query DataZone environments. A DataZone environment specifies a DataZone project and domain combination. When you use a DataZone environment from the Athena console, you assume the IAM role of the DataZone environment, and you see only the databases and tables that belong to that environment. Permissions are determined by the roles that you specify in DataZone.

In Athena, you can use the **DataZone environment** selector on the query editor page to choose a DataZone environment.

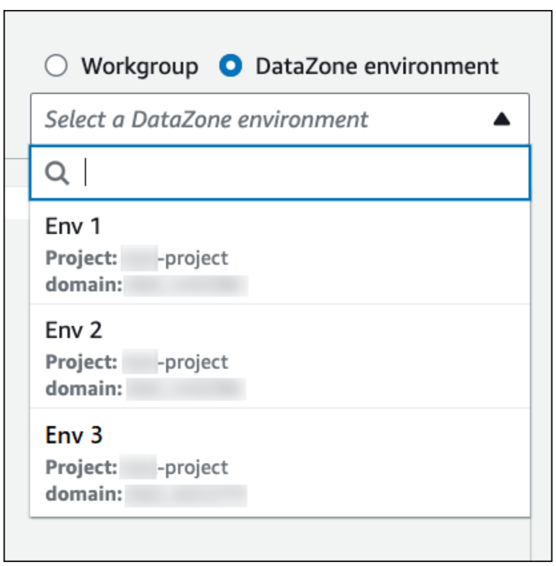
To open a DataZone environment in Athena

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the upper right of the Athena console, next to **Workgroup**, choose **DataZone environment**.

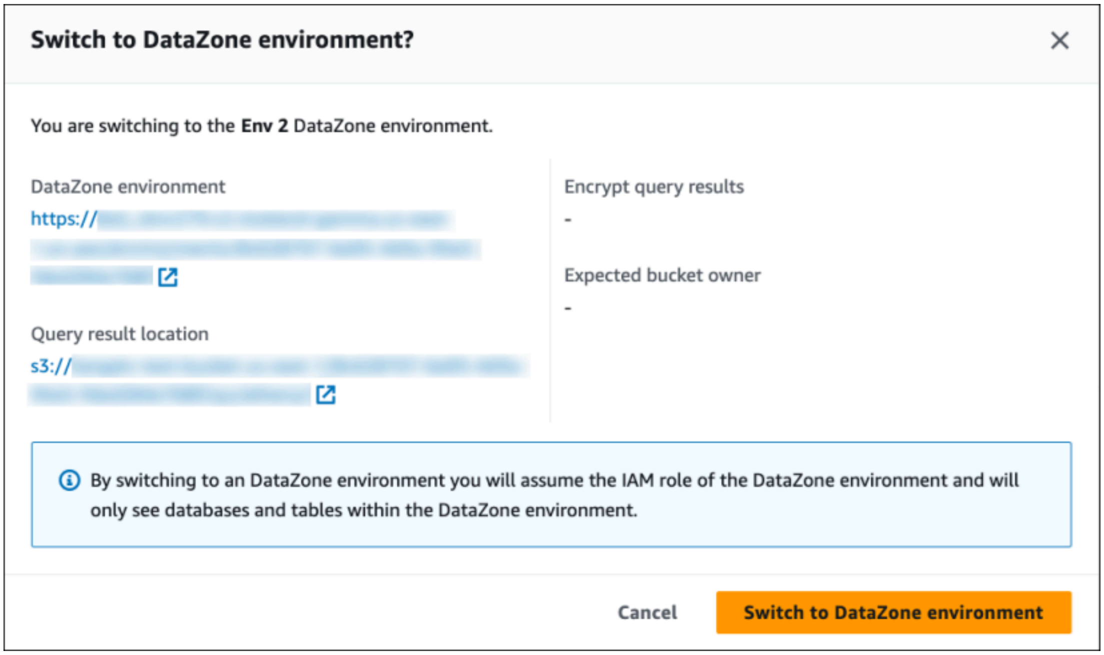
> **Note**
>
> The **DataZone environment** option is present only when you have one or more domains available in DataZone.



3. Use the **DataZone environment** selector to choose a DataZone environment.



4. In the **Switch to DataZone environment** dialog box, verify that the environment is the one that you want, and then choose **Switch to DataZone environment**.



For more information about getting started with DataZone and Athena, see the Getting started tutorial in the *Amazon DataZone User Guide*.

Use an external Hive metastore

You can use the Amazon Athena data connector for external Hive metastore to query data sets in Amazon S3 that use an Apache Hive metastore. No migration of metadata to the AWS Glue Data Catalog is necessary. In the Athena management console, you configure a Lambda function to communicate with the Hive metastore that is in your private VPC and then connect it to the metastore. The connection from Lambda to your Hive metastore is secured by a private Amazon VPC channel and does not use the public internet. You can provide your own Lambda function code, or you can use the default implementation of the Athena data connector for external Hive metastore.

Topics

- Use the AWS Serverless Application Repository to deploy a Hive data source connector
- Connect Athena to a Hive metastore using an existing IAM execution role
- Configure Athena to use a deployed Hive metastore connector
- Omit the catalog name in external Hive metastore queries
- Work with Hive views
- Use the AWS CLI with Hive metastores
- Modify the Athena external Hive metastore connector

Overview of features

With the Athena data connector for external Hive metastore, you can perform the following tasks:

- Use the Athena console to register custom catalogs and run queries using them.
- Define Lambda functions for different external Hive metastores and join them in Athena queries.
- Use the AWS Glue Data Catalog and your external Hive metastores in the same Athena query.
- Specify a catalog in the query execution context as the current default catalog. This removes the requirement to prefix catalog names to database names in your queries. Instead of using the syntax `catalog.database.table`, you can use `database.table`.
- Use a variety of tools to run queries that reference external Hive metastores. You can use the Athena console, the AWS CLI, the AWS SDK, Athena APIs, and updated Athena JDBC and ODBC drivers. The updated drivers have support for custom catalogs.

API support

Athena Data Connector for External Hive Metastore includes support for catalog registration API operations and metadata API operations.

- **Catalog registration** – Register custom catalogs for external Hive metastores and federated data sources.
- **Metadata** – Use metadata APIs to provide database and table information for AWS Glue and any catalog that you register with Athena.
- **Athena JAVA SDK client** – Use catalog registration APIs, metadata APIs, and support for catalogs in the `StartQueryExecution` operation in the updated Athena Java SDK client.

Reference implementation

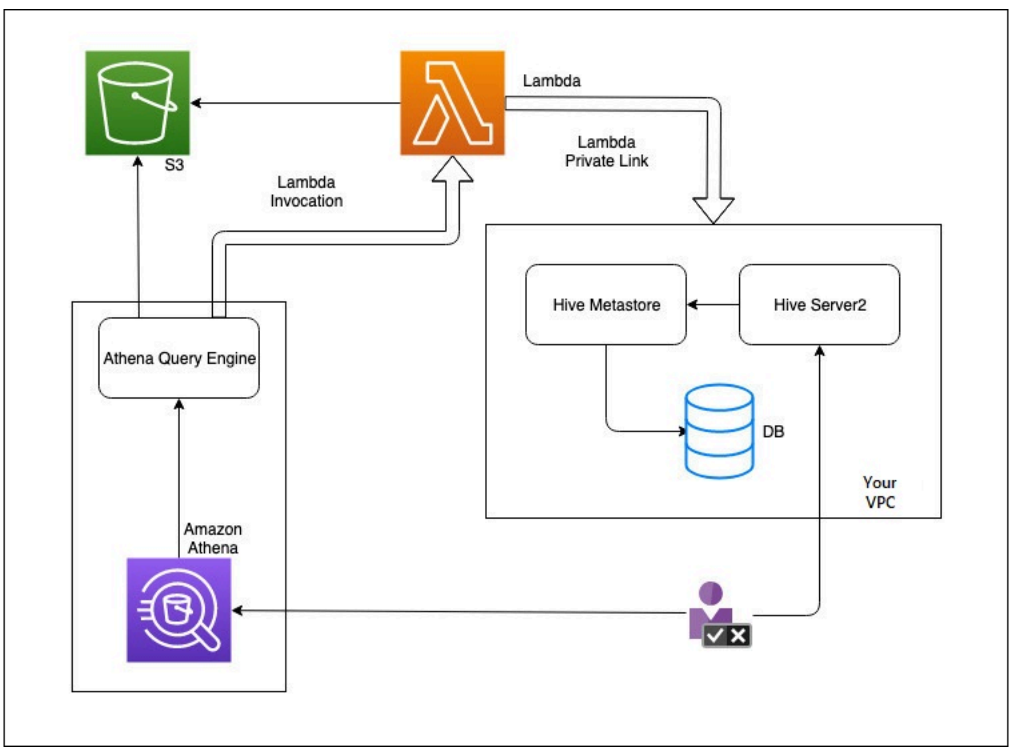
Athena provides a reference implementation for the Lambda function that connects to external Hive metastores. The reference implementation is provided on GitHub as an open source project at [Athena Hive metastore].

The reference implementation is available as the following two AWS SAM applications in the AWS Serverless Application Repository (SAR). You can use either of these applications in the SAR to create your own Lambda functions.

- **`AthenaHiveMetastoreFunction`** – Uber Lambda function `.jar` file. An "uber" JAR (also known as a fat JAR or JAR with dependencies) is a `.jar` file that contains both a Java program and its dependencies in a single file.
- **`AthenaHiveMetastoreFunctionWithLayer`** – Lambda layer and thin Lambda function `.jar` file.

Workflow

The following diagram shows how Athena interacts with your external Hive metastore.



In this workflow, your database-connected Hive metastore is inside your VPC. You use Hive Server2 to manage your Hive metastore using the Hive CLI.

The workflow for using external Hive metastores from Athena includes the following steps.

1. You create a Lambda function that connects Athena to the Hive metastore that is inside your VPC.
2. You register a unique catalog name for your Hive metastore and a corresponding function name in your account.
3. When you run an Athena DML or DDL query that uses the catalog name, the Athena query engine calls the Lambda function name that you associated with the catalog name.
4. Using AWS PrivateLink, the Lambda function communicates with the external Hive metastore in your VPC and receives responses to metadata requests. Athena uses the metadata from your external Hive metastore just like it uses the metadata from the default AWS Glue Data Catalog.

Considerations and limitations

When you use Athena Data Connector for External Hive Metastore, consider the following points:

- You can use CTAS to create a table on an external Hive metastore.
- You can use INSERT INTO to insert data into an external Hive metastore.
- DDL support for external Hive metastore is limited to the following statements.
  - ALTER DATABASE SET DBPROPERTIES
  - ALTER TABLE ADD COLUMNS
  - ALTER TABLE ADD PARTITION
  - ALTER TABLE DROP PARTITION
  - ALTER TABLE RENAME PARTITION
  - ALTER TABLE REPLACE COLUMNS
  - ALTER TABLE SET LOCATION
  - ALTER TABLE SET TBLPROPERTIES
  - CREATE DATABASE
  - CREATE TABLE
  - CREATE TABLE AS
  - DESCRIBE TABLE
  - DROP DATABASE
  - DROP TABLE

  - SHOW COLUMNS
  - SHOW CREATE TABLE
  - SHOW PARTITIONS
  - SHOW SCHEMAS
  - SHOW TABLES
  - SHOW TBLPROPERTIES
- The maximum number of registered catalogs that you can have is 1,000.
- Kerberos authentication for Hive metastore is not supported.
- To use the JDBC driver with an external Hive metastore or federated queries, include `MetadataRetrievalMethod=ProxyAPI` in your JDBC connection string. For information about the JDBC driver, see Connect to Amazon Athena with JDBC.
- The Hive hidden columns `$path`, `$bucket`, `$file_size`, `$file_modified_time`, `$partition`, `$row_id` cannot be used for fine-grained access control filtering.
- Hive hidden system tables like `example_table$partitions` or `example_table$properties` are not supported by fine-grained access control.

Permissions

Prebuilt and custom data connectors might require access to the following resources to function correctly. Check the information for the connector that you use to make sure that you have configured your VPC correctly. For information about required IAM permissions to run queries and create a data source connector in Athena, see Allow access to the Athena Data Connector for External Hive Metastore and Allow Lambda function access to external Hive metastores.

- **Amazon S3** – In addition to writing query results to the Athena query results location in Amazon S3, data connectors also write to a spill bucket in Amazon S3. Connectivity and permissions to this Amazon S3 location are required. For more information, see Spill location in Amazon S3 later in this topic.
- **Athena** – Access is required to check query status and prevent overscan.
- **AWS Glue** – Access is required if your connector uses AWS Glue for supplemental or primary metadata.
- **AWS Key Management Service**

- **Policies –** Hive metastore, Athena Query Federation, and UDFs require policies in addition to the AWS managed policy: AmazonAthenaFullAccess. For more information, see Identity and access management in Athena.

Spill location in Amazon S3

Because of the limit on Lambda function response sizes, responses larger than the threshold spill into an Amazon S3 location that you specify when you create your Lambda function. Athena reads these responses from Amazon S3 directly.

> **Note**
>
> Athena does not remove the response files on Amazon S3. We recommend that you set up a retention policy to delete response files automatically.

Connect Athena to an Apache Hive metastore

To connect Athena to an Apache Hive metastore, you must create and configure a Lambda function. For a basic implementation, you can perform all required steps starting from the Athena management console.

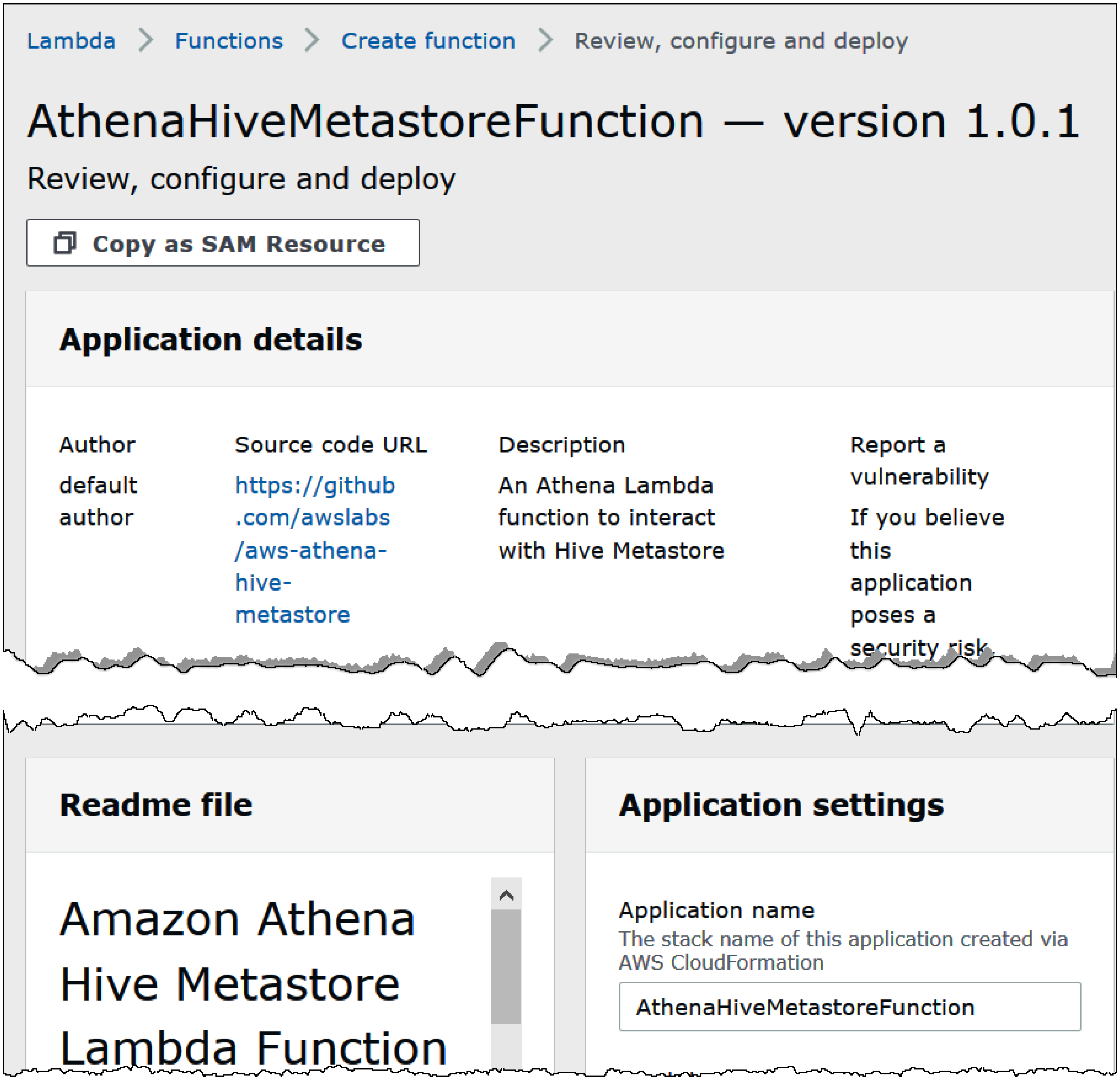
> **Note**
>
> The following procedure requires that you have permission to create a custom IAM role for the Lambda function. If you do not have permission to create a custom role, you can use the Athena reference implementation to create a Lambda function separately, and then use the AWS Lambda console to choose an existing IAM role for the function. For more information, see Connect Athena to a Hive metastore using an existing IAM execution role.

To connect Athena to a Hive metastore

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. Choose **Data sources and catalogs**.
4. On the upper right of the console, choose **Create data source**.
5. On the **Choose a data source** page, for **Data sources**, choose **S3 - Apache Hive metastore**.
6. Choose **Next**.
7. In the **Data source details** section, for **Data source name**, enter the name that you want to use in your SQL statements when you query the data source from Athena. The name can be up to 127 characters and must be unique within your account. It cannot be changed after you create it. Valid characters are a-z, A-Z, 0-9, _ (underscore), @ (at sign) and - (hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for data source names.
8. For **Lambda function**, choose **Create Lambda function**, and then choose **Create a new Lambda function in AWS Lambda**

   The **AthenaHiveMetastoreFunction** page opens in the AWS Lambda console. The page includes detailed information about the connector.

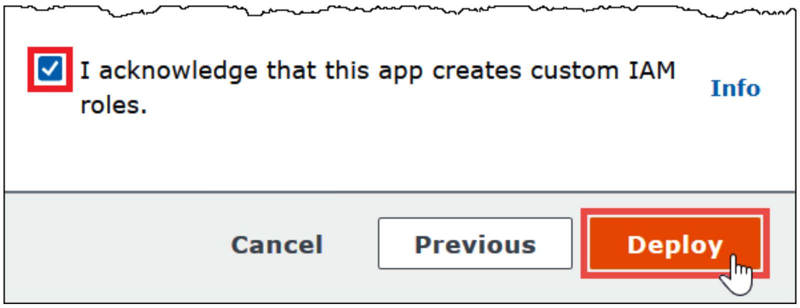


9. Under **Application settings**, enter the parameters for your Lambda function.

- **LambdaFuncName** – Provide a name for the function. For example, **myHiveMetastore**.
- **SpillLocation** – Specify an Amazon S3 location in this account to hold spillover metadata if the Lambda function response size exceeds 4 MB.
- **HMSUris** – Enter the URI of your Hive metastore host that uses the Thrift protocol at port 9083. Use the syntax `thrift://<host_name>:9083`.

- **LambdaMemory** – Specify a value from 128 MB to 3008 MB. The Lambda function is allocated CPU cycles proportional to the amount of memory that you configure. The default is 1024.
- **LambdaTimeout** – Specify the maximum permissible Lambda invocation run time in seconds from 1 to 900 (900 seconds is 15 minutes). The default is 300 seconds (5 minutes).
- **VPCSecurityGroupIds** – Enter a comma-separated list of VPC security group IDs for the Hive metastore.
- **VPCSubnetIds** – Enter a comma-separated list of VPC subnet IDs for the Hive metastore.

10. Select **I acknowledge that this app creates custom IAM roles**, and then choose **Deploy**.



When the deployment completes, your function appears in your list of Lambda applications. Now that the Hive metastore function has been deployed to your account, you can configure Athena to use it.

11. Return to the **Enter data source details** page of the Athena console.
12. In the **Lambda function** section, choose the refresh icon next to the Lambda function search box. Refreshing the list of available functions causes your newly created function to appear in the list.
13. Choose the name of the function that you just created in the Lambda console. The ARN of the Lambda function displays.
14. (Optional) For **Tags**, add key-value pairs to associate with this data source. For more information about tags, see Tag Athena resources.
15. Choose **Next**.
16. On the **Review and create** page, review the data source details, and then choose **Create data source**.
17. The **Data source details** section of the page for your data source shows information about your new connector.

You can now use the **Data source name** that you specified to reference the Hive metastore in your SQL queries in Athena. In your SQL queries, use the following example syntax, replacing `hms-catalog-1` with the catalog name that you specified earlier.

```
SELECT * FROM hms-catalog-1.CustomerData.customers
```

18. For information about viewing, editing, or deleting the data sources that you create, see Manage your data sources.

Use the AWS Serverless Application Repository to deploy a Hive data source connector

To deploy an Athena data source connector for Hive, you can use the AWS Serverless Application Repository instead of starting with the Athena console. Use the AWS Serverless Application Repository to find the connector that you want to use, provide the parameters that the connector requires, and then deploy the connector to your account. Then, after you deploy the connector, you use the Athena console to make the data source available to Athena.

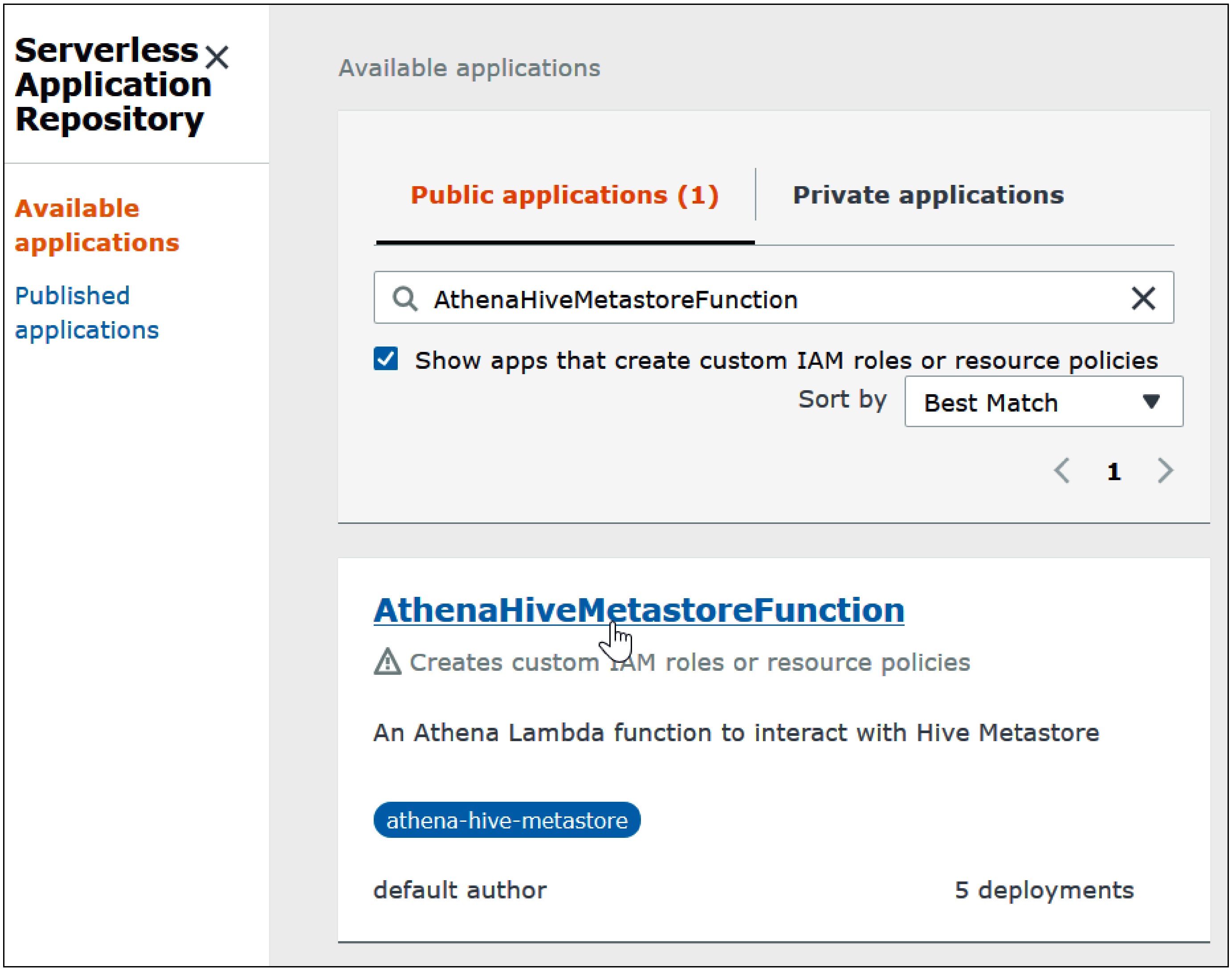
To use the AWS Serverless Application Repository to deploy a data source connector for Hive to your account

1. Sign in to the AWS Management Console and open the **Serverless App Repository**.
2. In the navigation pane, choose **Available applications**.
3. Select the option **Show apps that create custom IAM roles or resource policies**.
4. In the search box, enter **`Hive`**. The connectors that appear include the following two:

   - **AthenaHiveMetastoreFunction** – Uber Lambda function `.jar` file.
   - **AthenaHiveMetastoreFunctionWithLayer** – Lambda layer and thin Lambda function `.jar` file.

   The two applications have the same functionality and differ only in their implementation. You can use either one to create a Lambda function that connects Athena to your Hive metastore.

5. Choose the name of the connector that you want to use. This tutorial uses **AthenaHiveMetastoreFunction**.



6. Under **Application settings**, enter the parameters for your Lambda function.

  - **LambdaFuncName –** Provide a name for the function. For example, **myHiveMetastore**.
  - **SpillLocation –** Specify an Amazon S3 location in this account to hold spillover metadata if the Lambda function response size exceeds 4 MB.
  - **HMSUris –** Enter the URI of your Hive metastore host that uses the Thrift protocol at port 9083. Use the syntax `thrift://<host_name>:9083`.
  - **LambdaMemory –** Specify a value from 128 MB to 3008 MB. The Lambda function is allocated CPU cycles proportional to the amount of memory that you configure. The default is 1024.
  - **LambdaTimeout –** Specify the maximum permissible Lambda invocation run time in seconds from 1 to 900 (900 seconds is 15 minutes). The default is 300 seconds (5 minutes).

- **VPCSecurityGroupIds –** Enter a comma-separated list of VPC security group IDs for the Hive metastore.
- **VPCSubnetIds –** Enter a comma-separated list of VPC subnet IDs for the Hive metastore.

7. On the bottom right of the **Application details** page, select **I acknowledge that this app creates custom IAM roles**, and then choose **Deploy**.

At this point, you can configure Athena to use your Lambda function to connect to your Hive metastore. For steps, see Configure Athena to use a deployed Hive metastore connector.

Connect Athena to a Hive metastore using an existing IAM execution role

To connect your external Hive metastore to Athena with a Lambda function that uses an existing IAM role, you can use Athena's reference implementation of the Athena connector for external Hive metastore.

The three major steps are as follows:

1. **Clone and build** – Clone the Athena reference implementation and build the JAR file that contains the Lambda function code.
2. **AWS Lambda console** – In the AWS Lambda console, create a Lambda function, assign it an existing IAM execution role, and upload the function code that you generated.
3. **Amazon Athena console** – In the Amazon Athena console, create a data source name that you can use to refer to your external Hive metastore in your Athena queries.

If you already have permissions to create a custom IAM role, you can use a simpler workflow that uses the Athena console and the AWS Serverless Application Repository to create and configure a Lambda function. For more information, see Connect Athena to an Apache Hive metastore.

Prerequisites

- Git must be installed on your system.
- You must have Apache Maven installed.
- You have an IAM execution role that you can assign to the Lambda function. For more information, see Allow Lambda function access to external Hive metastores.

Clone and build the Lambda function

The function code for the Athena reference implementation is a Maven project located on GitHub at [awslabs/aws-athena-hive-metastore](). For detailed information about the project, see the corresponding README file on GitHub or the [Modify the Athena external Hive metastore connector]() topic in this documentation.

To clone and build the Lambda function code

1. Enter the following command to clone the Athena reference implementation:

   ```
   git clone https://github.com/awslabs/aws-athena-hive-metastore
   ```

2. Run the following command to build the `.jar` file for the Lambda function:

   ```
   mvn clean install
   ```

   After the project builds successfully, the following `.jar` file is created in the target folder of your project:

   `hms-lambda-func-1.0-SNAPSHOT-withdep.jar`

   In the next section, you use the AWS Lambda console to upload this file to your Amazon Web Services account.

Create and configure the Lambda function in the AWS Lambda console

In this section, you use the AWS Lambda console to create a function that uses an existing IAM execution role. After you configure a VPC for the function, you upload the function code and configure the environment variables for the function.

Create the Lambda function

In this step, you create a function in the AWS Lambda console that uses an existing IAM role.

To create a Lambda function that uses an existing IAM role

1. Sign in to the AWS Management Console and open the AWS Lambda console at [https://console.aws.amazon.com/lambda/]().
2. In the navigation pane, choose **Functions**.

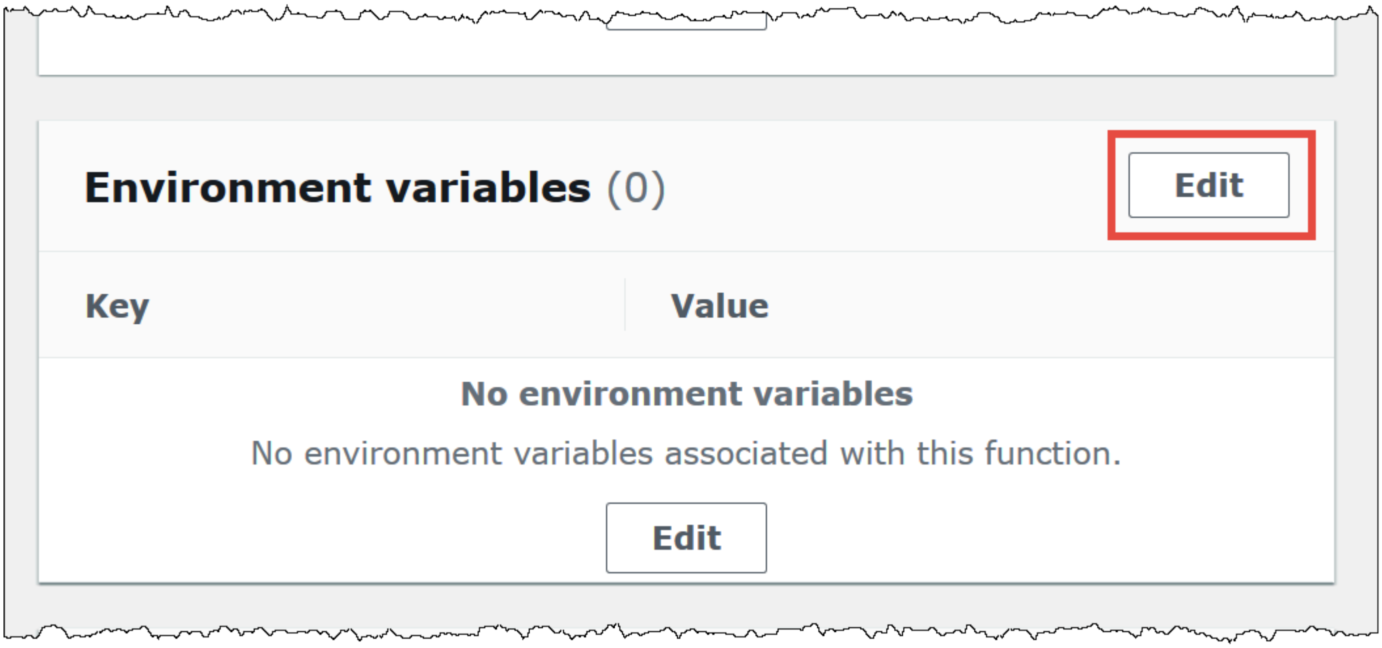
3. Choose **Create function**.
4. Choose **Author from scratch**.
5. For **Function name**, enter the name of your Lambda function (for example, **`EHMSBasedLambda`**).
6. For **Runtime**, choose **Java 8**.
7. Under **Permissions**, expand **Change default execution role**.
8. For **Execution role**, choose **Use an existing role**.
9. For **Existing role**, choose the IAM execution role that your Lambda function will use for Athena (this example uses a role called `AthenaLambdaExecutionRole`).
10. Expand **Advanced settings**.
11. Select **Enable Network**.
12. For **VPC**, choose the VPC that your function will have access to.
13. For **Subnets**, choose the VPC subnets for Lambda to use.
14. For **Security groups**, choose the VPC security groups for Lambda to use.
15. Choose **Create function**. The AWS Lambda console and opens the configuration page for your function and begins creating your function.

Upload the code and configure the Lambda function

When the console informs you that your function has been successfully created, you are ready to upload the function code and configure its environment variables.

To upload your Lambda function code and configure its environment variables

1. In the Lambda console, make sure that you are on the **Code** tab of the page of the function that you specfied.
2. For **Code source**, choose **Upload from**, and then choose **.zip or .jar file**.
3. Upload the `hms-lambda-func-1.0-SNAPSHOT-withdep.jar` file that you generated previously.
4. On your Lambda function page, choose the **Configuration** tab.
5. From the pane on the left, choose **Environment variables**.
6. In the **Environment variables** section, choose **Edit**.

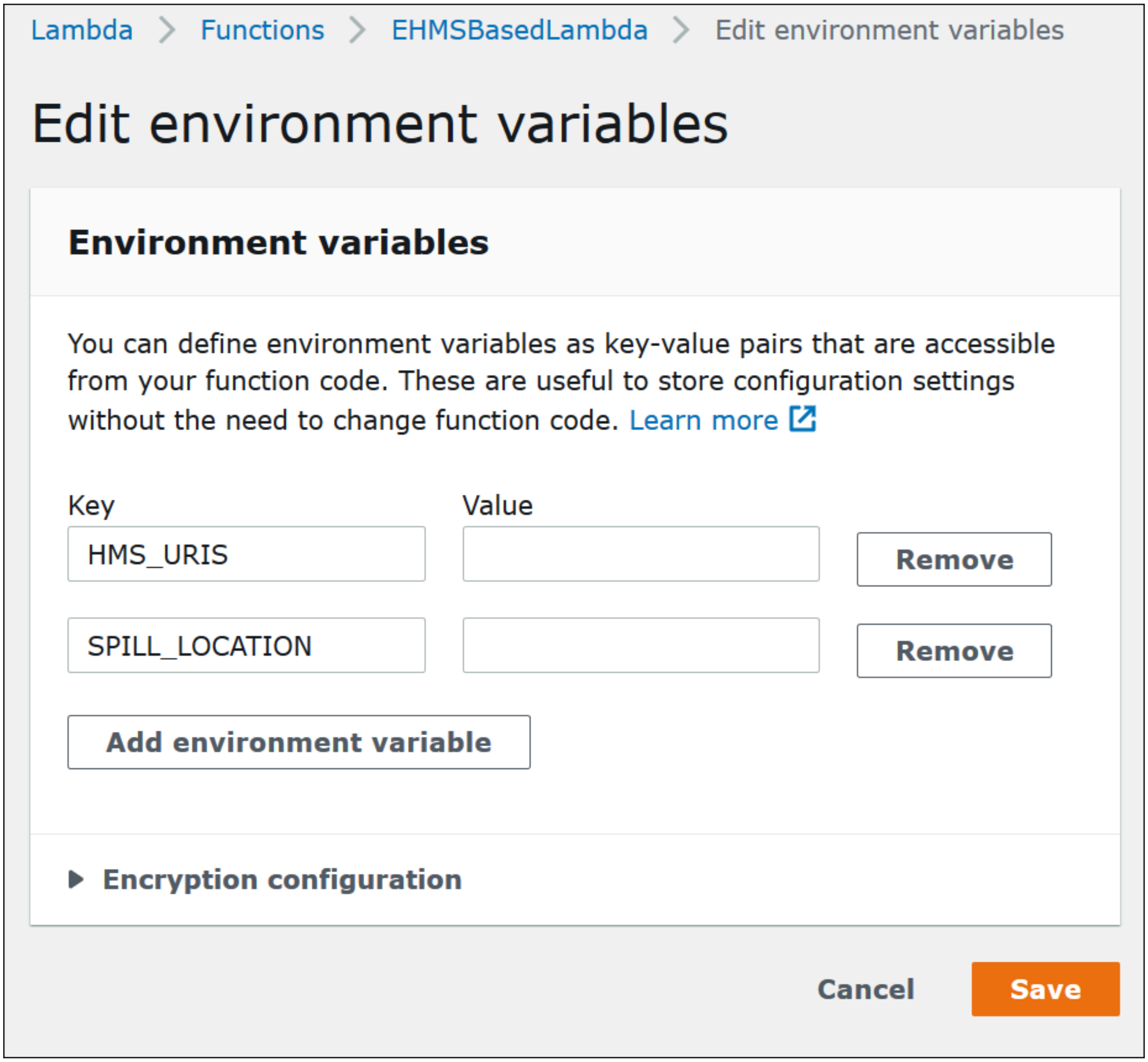


7. On the **Edit environment variables** page, use the **Add environment variable** option to add the following environment variable keys and values:

   - **HMS_URIS** – Use the following syntax to enter the URI of your Hive metastore host that uses the Thrift protocol at port 9083.

     ```
     thrift://<host_name>:9083
     ```

   - **SPILL_LOCATION** – Specify an Amazon S3 location in your Amazon Web Services account to hold spillover metadata if the Lambda function response size exceeds 4 MB.



8. Choose **Save**.

At this point, you are ready to configure Athena to use your Lambda function to connect to your Hive metastore. For steps, see [Configure Athena to use a deployed Hive metastore connector](#).

Configure Athena to use a deployed Hive metastore connector

After you have deployed a Lambda data source connector like `AthenaHiveMetastoreFunction` to your account, you can configure Athena to use it. To do so, you create a data source name that refers to your external Hive metastore to use in your Athena queries.

To connect Athena to your Hive metastore using an existing Lambda function

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. Choose **Data sources and catalogs**.
4. On the **Data sources and catalogs** page, choose **Create data source**.
5. On the **Choose a data source** page, for **Data sources**, choose **S3 - Apache Hive metastore**.
6. Choose **Next**.
7. In the **Data source details** section, for **Data source name**, enter the name that you want to use in your SQL statements when you query the data source from Athena (for example, `MyHiveMetastore`). The name can be up to 127 characters and must be unique within your account. It cannot be changed after you create it. Valid characters are a-z, A-Z, 0-9, _ (underscore), @ (at sign) and - (hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for data source names.
8. In the **Connection details** section, use the **Select or enter a Lambda function** box to choose the name of the function that you just created. The ARN of the Lambda function displays.
9. (Optional) For **Tags**, add key-value pairs to associate with this data source. For more information about tags, see [Tag Athena resources](#).
10. Choose **Next**.
11. On the **Review and create** page, review the data source details, and then choose **Create data source**.

12. The **Data source details** section of the page for your data source shows information about your new connector.

    You can now use the **Data source name** that you specified to reference the Hive metastore in your SQL queries in Athena.

    In your SQL queries, use the following example syntax, replacing `ehms-catalog` with the data source name that you specified earlier.

    ```
    SELECT * FROM ehms-catalog.CustomerData.customers
    ```

13. To view, edit, or delete the data sources that you create, see Manage your data sources.

Omit the catalog name in external Hive metastore queries

When you run DML and DDL queries on external Hive metastores, you can simplify your query syntax by omitting the catalog name if that name is selected in the query editor. Certain restrictions apply to this functionality.

DML statements

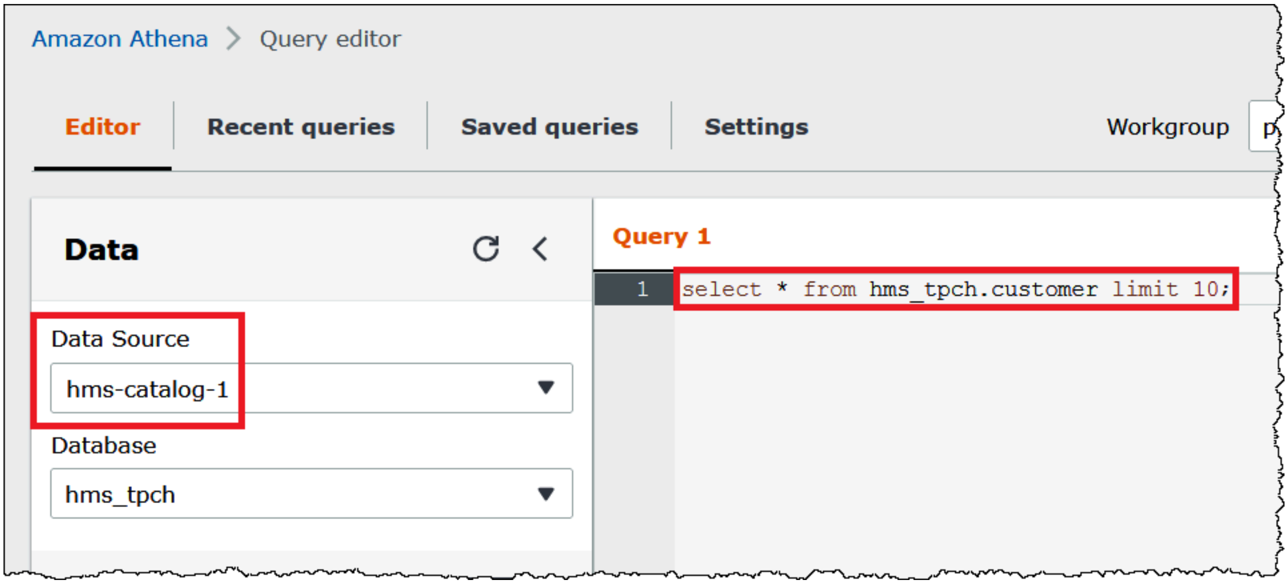
To run queries with registered catalogs

1. You can put the data source name before the database using the syntax `[[`*`data_source_name`*`].`*`database_name`*`].`*`table_name`*, as in the following example.

    ```
    select * from  "hms-catalog-1".hms_tpch.customer limit 10;
    ```

2. When the data source that you want to use is already selected in the query editor, you can omit the name from the query, as in the following example.

    ```
    select * from hms_tpch.customer limit 10:
    ```



3. When you use multiple data sources in a query, you can omit only the default data source name, and must specify the full name for any non-default data sources.

   For example, suppose `AwsDataCatalog` is selected as the default data source in the query editor. The `FROM` statement in the following query excerpt fully qualifies the first two data source names but omits the name for the third data source because it is in the AWS Glue data catalog.

```
...
FROM ehms01.hms_tpch.customer,
         "hms-catalog-1".hms_tpch.orders,
         hms_tpch.lineitem
...
```

DDL statements

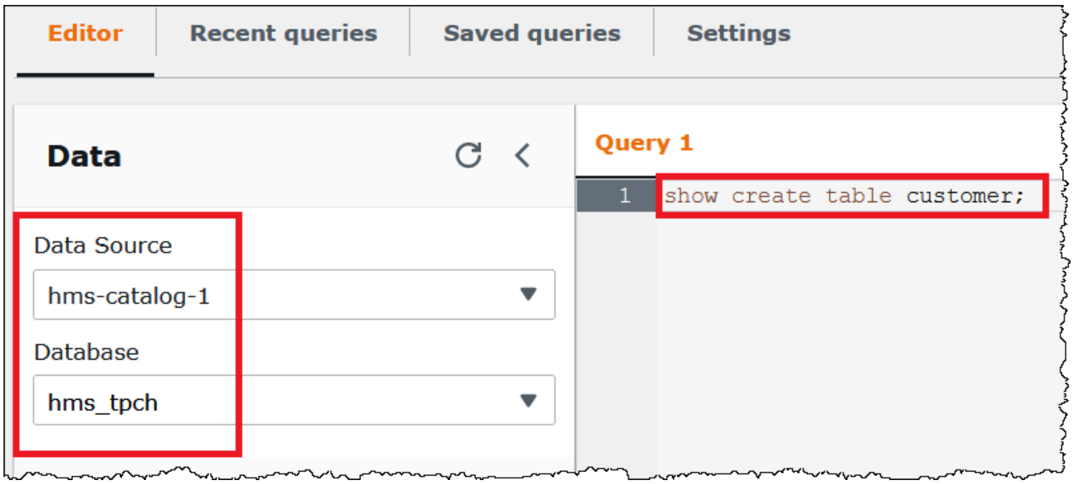
The following Athena DDL statements support catalog name prefixes. Catalog name prefixes in other DDL statements cause syntax errors.

```
SHOW TABLES [IN [catalog_name.]database_name] ['regular_expression']

SHOW TBLPROPERTIES [[catalog_name.]database_name.]table_name [('property_name')]

SHOW COLUMNS IN [[catalog_name.]database_name.]table_name
```

```
SHOW PARTITIONS [[catalog_name.]database_name.]table_name

SHOW CREATE TABLE [[catalog_name.][database_name.]table_name

DESCRIBE [EXTENDED | FORMATTED] [[catalog_name.][database_name.]table_name [PARTITION
 partition_spec] [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

As with DML statements, you can omit the datasource and database prefixes from the query when the data source and database are selected in the query editor.

In the following image, the `hms-catalog-1` data source and the `hms_tpch` database are selected in the query editor. The `show create table customer` statement succeeds even though the `hms-catalog-1` prefix and the `hms_tpch` database name are omitted from the query itself.



Specifying a default data source in a JDBC connection string

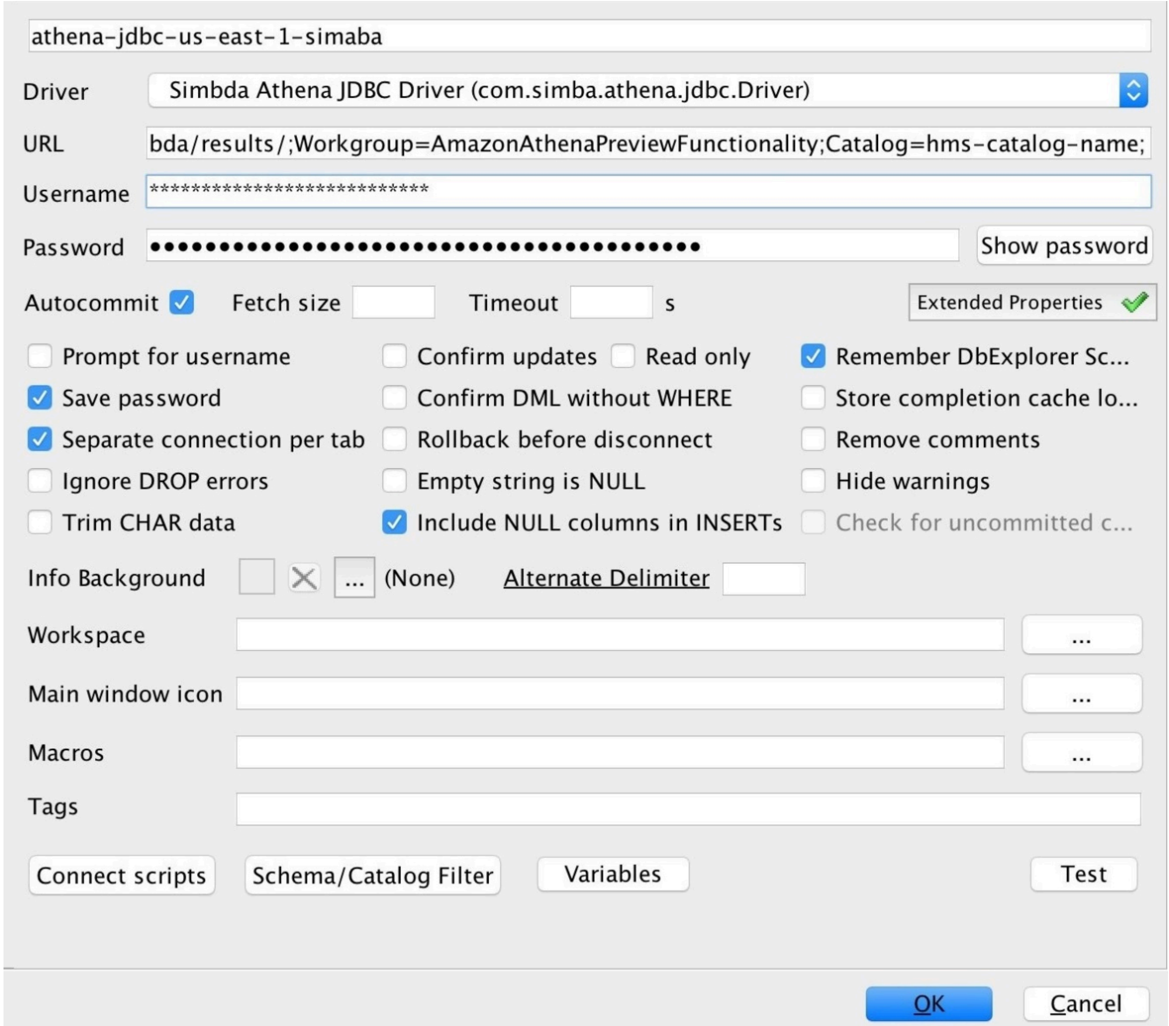
When you use the Athena JDBC Driver to connect Athena to an external Hive metastore, you can use the `Catalog` parameter to specify the default data source name in your connection string in a SQL editor like [SQL workbench](https://www.sql-workbench.eu/).

> **Note**
>
> To download the latest Athena JDBC drivers, see Using Athena with the JDBC driver.

The following connection string specifies the default data source *hms-catalog-name*.

```
    jdbc:awsathena://AwsRegion=us-east-1;S3OutputLocation=s3://amzn-s3-demo-bucket/
lambda/results/;Workgroup=AmazonAthenaPreviewFunctionality;Catalog=hms-catalog-name;
```

The following image shows a sample JDBC connection URL as configured in SQL Workbench.



Work with Hive views

You can use Athena to query existing views in your external Apache Hive metastores. Athena translates your views for you on-the-fly at runtime without changing the original view or storing the translation.

For example, suppose you have a Hive view like the following that uses a syntax not supported in Athena like `LATERAL VIEW explode()`:

```
CREATE VIEW team_view AS
SELECT team, score
FROM matches
```

```
LATERAL VIEW explode(scores) m AS score
```

Athena translates the Hive view query string into a statement like the following that Athena can run:

```
SELECT team, score
FROM matches
CROSS JOIN UNNEST(scores) AS m (score)
```

For information about connecting an external Hive metastore to Athena, see Use an external Hive metastore.

Considerations and limitations

When querying Hive views from Athena, consider the following points:

- Athena does not support creating Hive views. You can create Hive views in your external Hive metastore, which you can then query from Athena.
- Athena does not support custom UDFs for Hive views.
- Due to a known issue in the Athena console, Hive views appear under the list of tables instead of the list of views.
- Although the translation process is automatic, certain Hive functions are not supported for Hive views or require special handling. For more information, see the following section.

Hive function support limitations

This section highlights the Hive functions that Athena does not support for Hive views or that require special treatment. Currently, because Athena primarily supports functions from Hive 2.2.0, functions that are available only in higher versions (such as Hive 4.0.0) are not available. For a full list of Hive functions, see Hive language manual UDF.

Aggregate functions

Aggregate functions that require special handling

The following aggregate function for Hive views requires special handling.

- **Avg** – Instead of `avg(INT i)`, use `avg(CAST(i AS DOUBLE))`.

Aggregate functions not supported

The following Hive aggregate functions are not supported in Athena for Hive views.

```
covar_pop
histogram_numeric
ntile
percentile
percentile_approx
```

Regression functions like `regr_count`, `regr_r2`, and `regr_sxx` are not supported in Athena for Hive views.

Date functions not supported

The following Hive date functions are not supported in Athena for Hive views.

```
date_format(date/timestamp/string ts, string fmt)
day(string date)
dayofmonth(date)
extract(field FROM source)
hour(string date)
minute(string date)
month(string date)
quarter(date/timestamp/string)
second(string date)
weekofyear(string date)
year(string date)
```

Masking functions not supported

Hive masking functions like `mask()`, and `mask_first_n()` are not supported in Athena for Hive views.

Miscellaneous functions

Miscellaneous functions that require special handling

The following miscellaneous functions for Hive views require special handling.

- **md5** – Athena supports `md5(binary)` but not `md5(varchar)`.
- **Explode** – Athena supports `explode` when it is used in the following syntax:

```
LATERAL VIEW [OUTER] EXPLODE(<argument>)
```

- **Posexplode** – Athena supports `posexplode` when it is used in the following syntax:

```
LATERAL VIEW [OUTER] POSEXPLODE(<argument>)
```

In the `(pos, val)` output, Athena treats the `pos` column as `BIGINT`. Because of this, you may need to cast the `pos` column to `BIGINT` to avoid a stale view. The following example illustrates this technique.

```
SELECT CAST(c AS BIGINT) AS c_bigint, d
FROM table LATERAL VIEW POSEXPLODE(<argument>) t AS c, d
```

Miscellaneous functions not supported

The following Hive functions are not supported in Athena for Hive views.

```
aes_decrypt
aes_encrypt
current_database
current_user
inline
java_method
logged_in_user
reflect
sha/sha1/sha2
stack
version
```

Operators

Operators that require special handling

The following operators for Hive views require special handling.

- **Mod operator (%)** – Because the `DOUBLE` type implicitly casts to `DECIMAL(x,y)`, the following syntax can cause a View is stale error message:

```
a_double % 1.0 AS column
```

To work around this issue, use CAST, as in the following example.

```
CAST(a_double % 1.0 as DOUBLE) AS column
```

- **Division operator (/)** – In Hive, `int` divided by `int` produces a `double`. In Athena, the same operation produces a truncated `int`.

Operators not supported

Athena does not support the following operators for Hive views.

~A – bitwise NOT

A ^ b – bitwise XOR

A & b – bitwise AND

A | b – bitwise OR

A <=> b – Returns same result as the equals (=) operator for non-null operands. Returns TRUE if both are NULL, FALSE if one of them is NULL.

String functions

String functions that require special handling

The following Hive string functions for Hive views require special handling.

- **chr(bigint|double a)** – Hive allows negative arguments; Athena does not.
- **instr(string str, string substr)** – Because the Athena mapping for the `instr` function returns BIGINT instead of INT, use the following syntax:

```
CAST(instr(string str, string substr) as INT)
```

  Without this step, the view will be considered stale.
- **length(string a)** – Because the Athena mapping for the `length` function returns BIGINT instead of INT, use the following syntax so that the view will not be considered stale:

```
CAST(length(string str) as INT)
```

String functions not supported

The following Hive string functions are not supported in Athena for Hive views.

```
ascii(string str)
character_length(string str)
decode(binary bin, string charset)
encode(string src, string charset)
elt(N int,str1 string,str2 string,str3 string,...)
field(val T,val1 T,val2 T,val3 T,...)
find_in_set(string str, string strList)
initcap(string A)
levenshtein(string A, string B)
locate(string substr, string str[, int pos])
octet_length(string str)
parse_url(string urlString, string partToExtract [, string keyToExtract])
printf(String format, Obj... args)
quote(String text)
regexp_extract(string subject, string pattern, int index)
repeat(string str, int n)
sentences(string str, string lang, string locale)
soundex(string A)
space(int n)
str_to_map(text[, delimiter1, delimiter2])
substring_index(string A, string delim, int count)
```

XPath functions not supported

Hive XPath functions like `xpath`, `xpath_short`, and `xpath_int` are not supported in Athena for Hive views.

Troubleshooting

When you use Hive views in Athena, you may encounter the following issues:

- **View *<view name>* is stale** – This message usually indicates a type mismatch between the view in Hive and Athena. If the same function in the Hive LanguageManual UDF and Presto functions and operators documentation has different signatures, try casting the mismatched data type.
- **Function not registered** – Athena does not currently support the function. For details, see the information earlier in this document.

Use the AWS CLI with Hive metastores

You can use aws athena CLI commands to manage the Hive metastore data catalogs that you use with Athena. After you have defined one or more catalogs to use with Athena, you can reference those catalogs in your aws athena DDL and DML commands.

Using the AWS CLI to manage Hive metastore catalogs

Registering a catalog: Create-data-catalog

To register a data catalog, you use the create-data-catalog command. Use the name parameter to specify the name that you want to use for the catalog. Pass the ARN of the Lambda function to the metadata-function option of the parameters argument. To create tags for the new catalog, use the tags parameter with one or more space-separated Key=*key*,Value=*value* argument pairs.

The following example registers the Hive metastore catalog named hms-catalog-1. The command has been formatted for readability.

```
$ aws athena create-data-catalog
 --name "hms-catalog-1"
 --type "HIVE"
 --description "Hive Catalog 1"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3,sdk-version=1.0"
 --tags Key=MyKey,Value=MyValue
 --region us-east-1
```

Showing catalog details: Get-data-catalog

To show the details of a catalog, pass the name of the catalog to the get-data-catalog command, as in the following example.

```
$ aws athena get-data-catalog --name "hms-catalog-1" --region us-east-1
```

The following sample result is in JSON format.

```
{
    "DataCatalog": {
        "Name": "hms-catalog-1",
        "Description": "Hive Catalog 1",
```

```
        "Type": "HIVE",
        "Parameters": {
            "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
            "sdk-version": "1.0"
        }
    }
}
```

Listing registered catalogs: List-data-catalogs

To list the registered catalogs, use the `list-data-catalogs` command and optionally specify a Region, as in the following example. The catalogs listed always include AWS Glue.

```
$ aws athena list-data-catalogs --region us-east-1
```

The following sample result is in JSON format.

```
{
    "DataCatalogs": [
        {
            "CatalogName": "AwsDataCatalog",
            "Type": "GLUE"
        },
        {
            "CatalogName": "hms-catalog-1",
            "Type": "HIVE",
            "Parameters": {
                "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
                "sdk-version": "1.0"
            }
        }
    ]
}
```

Updating a catalog: Update-data-catalog

To update a data catalog, use the `update-data-catalog` command, as in the following example. The command has been formatted for readability.

```
$ aws athena update-data-catalog
```

```
 --name "hms-catalog-1"
 --type "HIVE"
 --description "My New Hive Catalog Description"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new,sdk-version=1.0"
 --region us-east-1
```

Deleting a catalog: Delete-data-catalog

To delete a data catalog, use the `delete-data-catalog` command, as in the following example.

```
$ aws athena delete-data-catalog --name "hms-catalog-1" --region us-east-1
```

Showing database details: Get-database

To show the details of a database, pass the name of the catalog and the database to the `get-database` command, as in the following example.

```
$ aws athena get-database --catalog-name hms-catalog-1 --database-name mydb
```

The following sample result is in JSON format.

```
{
    "Database": {
        "Name": "mydb",
        "Description": "My database",
        "Parameters": {
            "CreatedBy": "Athena",
            "EXTERNAL": "TRUE"
        }
    }
}
```

Listing databases in a catalog: List-databases

To list the databases in a catalog, use the `list-databases` command and optionally specify a Region, as in the following example.

```
$ aws athena list-databases --catalog-name AwsDataCatalog --region us-west-2
```

The following sample result is in JSON format.


```
{
    "DatabaseList": [
        {
            "Name": "default"
        },
        {
            "Name": "mycrawlerdatabase"
        },
        {
            "Name": "mydatabase"
        },
        {
            "Name": "sampledb",
            "Description": "Sample database",
            "Parameters": {
                "CreatedBy": "Athena",
                "EXTERNAL": "TRUE"
            }
        },
        {
            "Name": "tpch100"
        }
    ]
}
```


Showing table details: Get-table-metadata

To show the metadata for a table, including column names and datatypes, pass the name of the catalog, database, and table name to the `get-table-metadata` command, as in the following example.

```
$ aws athena get-table-metadata --catalog-name AwsDataCatalog --database-name mydb --
table-name cityuseragent
```

The following sample result is in JSON format.


```
{
    "TableMetadata": {
        "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
```

```
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://amzn-s3-demo-bucket/",
                "numFiles": "1",
                "numRows": "-1",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
            }
        }
}
```

Showing metadata for all tables in a database: List-table-metadata

To show the metadata for all tables in a database, pass the name of the catalog and database name to the `list-table-metadata` command. The `list-table-metadata` command is similar to the `get-table-metadata` command, except that you do not specify a table name. To limit the number of results, you can use the `--max-results` option, as in the following example.

```
$ aws athena list-table-metadata --catalog-name AwsDataCatalog --database-name sampledb
 --region us-east-1 --max-results 2
```

The following sample result is in JSON format.


```
{
    "TableMetadataList": [
        {
            "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://amzn-s3-demo-bucket/",
                "numFiles": "1",
                "numRows": "-1",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
            }
        },
        {
            "Name": "clearinghouse_data",
            "CreateTime": 1589255544.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
```

```
                {
                    "Name": "location",
                    "Type": "string"
                },
                {
                    "Name": "stock_count",
                    "Type": "int"
                },
                {
                    "Name": "quantity_shipped",
                    "Type": "int"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "location": "s3://amzn-s3-demo-bucket/",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "transient_lastDdlTime": "1589255544"
            }
        }
    ],
    "NextToken":
 "eyJsYXN0RXZhbHVhdGVkS2V5Ijp7IkhBU0hfS0VZIjp7InMiOiJ0Ljk0YWZjYjk1MjJjNTQ1YmU4Y2I5OWE5NTg0MjFjY
}
```

Running DDL and DML statements

When you use the AWS CLI to run DDL and DML statements, you can pass the name of the Hive metastore catalog in one of two ways:

- Directly into the statements that support it.
- To the `--query-execution-context Catalog` parameter.

DDL statements

The following example passes in the catalog name directly as part of the `show create table` DDL statement. The command has been formatted for readability.

```
$ aws athena start-query-execution
 --query-string "show create table hms-catalog-1.hms_tpch_partitioned.lineitem"
 --result-configuration "OutputLocation=s3://amzn-s3-demo-bucket/lambda/results"
```

The following example DDL `show create table` statement uses the `Catalog` parameter of `--query-execution-context` to pass the Hive metastore catalog name `hms-catalog-1`. The command has been formatted for readability.

```
$ aws athena start-query-execution
 --query-string "show create table lineitem"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://amzn-s3-demo-bucket/lambda/results"
```

DML statements

The following example DML `select` statement passes the catalog name into the query directly. The command has been formatted for readability.

```
$ aws athena start-query-execution
 --query-string "select * from hms-catalog-1.hms_tpch_partitioned.customer limit 100"
 --result-configuration "OutputLocation=s3://amzn-s3-demo-bucket/lambda/results"
```

The following example DML `select` statement uses the `Catalog` parameter of `--query-execution-context` to pass in the Hive metastore catalog name `hms-catalog-1`. The command has been formatted for readability.

```
$ aws athena start-query-execution
 --query-string "select * from customer limit 100"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://amzn-s3-demo-bucket/lambda/results"
```

Modify the Athena external Hive metastore connector

If you have special requirements, you can modify the Athena connector for external Hive metastore for your own use. Athena provides a reference implementation of the connector on GitHub.com at https://github.com/awslabs/aws-athena-hive-metastore. Most use cases do not require you to modify the reference implementation. However, if necessary, you can modify the source code and build the artifacts yourself.

The reference implementation is an Apache Maven project that has the following modules:

- **`hms-service-api`** – Contains the API operations between the Lambda function and the Athena service clients. These API operations are defined in the `HiveMetaStoreService` interface. Because this is a service contract, you should not change anything in this module.
- **`hms-lambda-handler`** – A set of default Lambda handlers that process all Hive metastore API calls. The class `MetadataHandler` is the dispatcher for all API calls. You do not need to change this package.
- **`hms-lambda-func`** – An example Lambda function that has the following components.
  - **`HiveMetaStoreLambdaFunc`** – An example Lambda function that extends `MetadataHandler`.
  - **`ThriftHiveMetaStoreClient`** – A Thrift client that communicates with Hive metastore. This client is written for Hive 2.3.0. If you use a different Hive version, you might need to update this class to ensure that the response objects are compatible.
  - **`ThriftHiveMetaStoreClientFactory`** – Controls the behavior of the Lambda function. For example, you can provide your own set of handler providers by overriding the `getHandlerProvider()` method.
  - `hms.properties` – Configures the Lambda function. Most cases require updating the following two properties only.
    - `hive.metastore.uris` – the URI of the Hive metastore in the format `thrift://<host_name>:9083`.
    - `hive.metastore.response.spill.location`: The Amazon S3 location to store response objects when their sizes exceed a given threshold (for example, 4 MB). The threshold is defined in the property `hive.metastore.response.spill.threshold`. Changing the default value is not recommended.

> **Note**
>
> These two properties can be overridden by the Lambda environment variables `HMS_URIS` and `SPILL_LOCATION`. Use these variables instead of recompiling the source code for

the Lambda function when you want to use the function with a different Hive metastore or spill location.

- **`hms-lambda-layer`** – A Maven assembly project that puts `hms-service-api`, `hms-lambda-handler`, and their dependencies into a `.zip` file. The `.zip` file is registered as a Lambda layer for use by multiple Lambda functions.
- **`hms-lambda-rnp`** – Records the responses from a Lambda function and then uses them to replay the response. You can use this model to simulate Lambda responses for testing.

Building the artifacts yourself

After you modify the source code, you can build the artifacts yourself and upload them to an Amazon S3 location.

Before you build the artifacts, update the properties `hive.metastore.uris` and `hive.metastore.response.spill.location` in the `hms.properties` file in the `hms-lambda-func` module.

To build the artifacts, you must have Apache Maven installed and run the command `mvn install`. This generates the layer `.zip` file in the output folder called `target` in the module `hms-lambda-layer` and the Lambda function `.jar` file in the module `hms-lambd-func`.

Manage your data sources

You can use the **Data sources and catalogs** page of the Athena console to manage the data sources that you create.

To view a data source

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Data sources and catalogs**.
4. From the list of data sources, choose the name of the data source that you want to view.

> **Note**
>
> The items in the **Data source name** column correspond to the output of the ListDataCatalogs API action and the list-data-catalogs CLI command.

To edit a data source

1. On the **Data sources and catalogs** page, do one of the following:
   - Select the button next to the catalog name, and then choose **Actions**, **Edit**.
   - Choose the name of the data source. Then on the details page, choose **Actions**, **Edit**.
2. On the **Edit** page, you can choose a different Lambda function for the data source, change the description, or add custom tags. For more information about tags, see Tag Athena resources.
3. Choose **Save**.
4. To edit your **AwsDataCatalog** data source, choose the **AwsDataCatalog** link to open its details page. Then, on the details page, choose the link to the AWS Glue console where you can edit your catalog.

To share a data source

For information about sharing data sources, visit the following links.

- For non-Hive Lambda-based data sources, see Enable cross-account federated queries.
- For AWS Glue Data Catalogs, see Configure cross-account access to AWS Glue data catalogs.

To delete a data source

1. On the **Data sources and catalogs** page, do one of the following:
   - Select the button next to the catalog name, and then choose **Actions**, **Delete**.
   - Choose the name of the data source, and then, on the details page, choose **Actions**, **Delete**.

   > **Note**
   >
   > The **AwsDataCatalog** is the default data source in your account and cannot be deleted.

   You are warned that when you delete a data source, its corresponding data catalog, tables, and views are removed from the query editor. Saved queries that used the data source will no longer run in Athena.
2. To confirm the deletion, type the name of the data source, and then choose **Delete**.

Connect to Amazon Athena with ODBC and JDBC drivers

To explore and visualize your data with business intelligence tools, download, install, and configure an ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity) driver.

Topics

- Connect to Amazon Athena with JDBC
- Connect to Amazon Athena with ODBC

See also the following AWS Knowledge Center and AWS Big Data Blog topics:

- How can I use my IAM role credentials or switch to another IAM role when connecting to Athena using the JDBC driver?

- Setting up trust between ADFS and AWS and using Active Directory credentials to connect to Amazon Athena with ODBC driver

Connect to Amazon Athena with JDBC

Amazon Athena offers two JDBC drivers, versions 2.x and 3.x. The Athena JDBC 3.x driver is the new generation driver offering better performance and compatibility. The JDBC 3.x driver supports reading query results directly from Amazon S3, which improves the performance of applications that consume large query results. The new driver also has fewer third-party dependencies, which makes integration with BI tools and custom applications easier. In most cases, you can use the new driver with no or minimal changes to existing configuration.

- To download the JDBC 3.x driver, see Athena JDBC 3.x driver.
- To download the JDBC 2.x driver, see Athena JDBC 2.x driver.

Topics

Athena JDBC 3.x driver

You can use the Athena JDBC driver to connect to Amazon Athena from many third-party SQL client tools and from custom applications.

System Requirements

- Java 8 (or higher) runtime environment
- At least 20 MB of available disk space

Considerations and limitations

Following are some considerations and limitations for the Athena JDBC 3.x driver.

- **Logging –** The 3.x driver uses SLF4J, which is an abstraction layer that enables the use of any one of several logging systems at runtime.

- **Encryption** – When using the Amazon S3 fetcher with the `CSE_KMS` encryption option, the Amazon S3 client can't decrypt results stored in an Amazon S3 bucket. If you require `CSE_KMS` encryption, you can continue to use the streaming fetcher. Support for `CSE_KMS` encryption with the Amazon S3 fetcher is planned.

JDBC 3.x driver download

This section contains download and license information for the JDBC 3.x driver.

Important

When you use the JDBC 3.x driver, be sure to note the following requirements:

- **Open port 444** – Keep port 444, which Athena uses to stream query results, open to outbound traffic. When you use a PrivateLink endpoint to connect to Athena, ensure that the security group attached to the PrivateLink endpoint is open to inbound traffic on port 444.
- **athena:GetQueryResultsStream policy** – Add the `athena:GetQueryResultsStream` policy action to the IAM principals that use the JDBC driver. This policy action is not exposed directly with the API. It is used only with the ODBC and JDBC drivers as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess.

To download the Amazon Athena 3.x JDBC driver, visit the following links.

JDBC driver uber jar

The following download packages the driver and all its dependencies in the same `.jar` file. This download is commonly used for third-party SQL clients.

3.5.0 uber jar

JDBC driver lean jar

The following download is a `.zip` file that contains the lean `.jar` for the driver and separate `.jar` files for the driver's dependencies. This download is commonly used for custom applications that might have dependencies that conflict with the dependencies that the driver uses. This download is useful if you want to choose which of the driver dependencies to include with the lean jar, and which to exclude if your custom application already contains one or more of them.

3.5.0 lean jar

License

The following link contains the license agreement for the JDBC 3.x driver.

License

Topics

- Get started with the JDBC 3.x driver
- Amazon Athena JDBC 3.x connection parameters
- Other JDBC 3.x configuration
- Amazon Athena JDBC 3.x release notes
- Previous versions of the Athena JDBC 3.x driver

Get started with the JDBC 3.x driver

Use the information in this section to get started with the Amazon Athena JDBC 3.x driver.

Topics

- Installation Instructions
- Running the driver
- Configuring the driver
- Upgrading from the Athena JDBC v2 driver

Installation Instructions

You can use the JDBC 3.x driver in custom application or from a third-party SQL client.

In a custom application

Download the `.zip` file that contains the driver jar and its dependencies. Each dependency has its own `.jar` file. Add the driver jar as a dependency in your custom application. Selectively add the dependencies of the driver jar based on whether you have already added those dependencies to your application from another source.

In a third-party SQL client

Download the driver uber jar file and add it to the third-party SQL client following the instructions for that client.

Running the driver

To run the driver, you can use a custom application or a third-party SQL client.

In a custom application

Use the JDBC interface to interact with the JDBC driver from a program. The following code shows a sample custom Java application.

```
public static void main(String args[]) throws SQLException {
    Properties connectionParameters = new Properties();
    connectionParameters.setProperty("Workgroup", "primary");
    connectionParameters.setProperty("Region", "us-east-2");
    connectionParameters.setProperty("Catalog", "AwsDataCatalog");
    connectionParameters.setProperty("Database","sampledatabase");
    connectionParameters.setProperty("OutputLocation","s3://amzn-s3-demo-bucket");
    connectionParameters.setProperty("CredentialsProvider","DefaultChain");
    String url = "jdbc:athena://";
    AthenaDriver driver = new AthenaDriver();
    Connection connection = driver.connect(url, connectionParameters);
    Statement statement = connection.createStatement();
    String query = "SELECT * from sample_table LIMIT 10";
    ResultSet resultSet = statement.executeQuery(query);
    printResults(resultSet); // A custom-defined method for iterating over a
                             // result set and printing its contents
}
```

In a third-party SQL client

Follow the documentation for the SQL client that you are using. Typically, you use the SQL client's graphical user interface to enter and submit the query, and the query results are displayed in the same interface.

Configuring the driver

You can use connection parameters to configure the Amazon Athena JDBC driver. For supported connection parameters, see [Amazon Athena JDBC 3.x connection parameters](#).

In a custom application

To set the connection parameters for the JDBC driver in a custom application, do one of the following:

- Add the parameter names and their values to a `Properties` object. When you call `Connection#connect`, pass that object along with the URL. For an example, see the sample Java application in Running the driver.
- In the connection string (the URL), use the following format to add the parameter names and their values directly after the protocol prefix.

```
<parameterName>=<parameterValue>;
```

  Use a semi-colon at the end of each parameter name/parameter value pair, and leave no white space after the semicolon, as in the following example.

```
String url = "jdbc:athena://WorkGroup=primary;Region=us-east-1;...;";AthenaDriver
 driver = new AthenaDriver();Connection connection = driver.connect(url, null);
```

> **Note**
>
> If a parameter is specified both in the connection string and in the `Properties` object, the value in the connection string takes precedence. Specifying the same parameter in both places is not recommended.

- Add the parameter values as arguments to the methods of `AthenaDataSource`, as in the following example.

```
AthenaDataSource dataSource = new AthenaDataSource();
    dataSource.setWorkGroup("primary");
    dataSource.setRegion("us-east-2");
    ...
    Connection connection = dataSource.getConnection();
    ...
```

In a third-party SQL client

Follow the instructions of the SQL client that you are using. Typically, the client provides a graphical user interface to input the parameter names and their values.

Upgrading from the Athena JDBC v2 driver

Most of the JDBC version 3 connection parameters are backwards-compatible with the version 2 (Simba) JDBC driver. This means that a version 2 connection string can be reused with version 3 of the driver. However, some connection parameters have changed. These changes are described here. When you upgrade to the version 3 JDBC driver, update your existing configuration if necessary.

Driver class

Some BI tools ask you to provide the driver class from the JDBC driver `.jar` file. Most tools find this class automatically. The fully qualified name of the class in the version 3 driver is `com.amazon.athena.jdbc.AthenaDriver`. In the version 2 driver, the class was `com.simba.athena.jdbc.Driver`.

Connection string

The version 3 driver uses `jdbc:athena://` for the protocol at the beginning of the JDBC connection string URL. The version 3 driver also supports the version 2 protocol `jdbc:awsathena://`, but the use of the version 2 protocol is deprecated. To avoid undefined behaviors, version 3 does not accept connection strings that start with `jdbc:awsathena://` if version 2 (or any other driver that accepts connection strings that start with `jdbc:awsathena://`) has been registered with the DriverManager class.

Credentials providers

The version 2 driver uses fully qualified names to identify different credentials providers (for example, `com.simba.athena.amazonaws.auth.DefaultAWSCredentialsProviderChain`. The version 3 driver uses shorter names (for example, `DefaultChain`). The new names are described in the corresponding sections for each credentials provider.

Custom credentials providers written for the version 2 driver need to be modified for the version 3 driver to implement the AwsCredentialsProvider interface from the new AWS SDK for Java instead of the AWSCredentialsProvider interface from the previous AWS SDK for Java.

The `PropertiesFileCredentialsProvider` is not supported in the JDBC 3.x driver. The provider was used in the JDBC 2.x driver but belongs to the previous version of the AWS SDK for

Java which is approaching end of support. To achieve the same functionality in the JDBC 3.x driver, use the AWS configuration profile credentials provider instead.

Log level

The following table shows the differences in the `LogLevel` parameters in the JDBC version 2 and version 3 drivers.

JDBC driver version	Parameter name	Parameter type	Default value	Possible values	Connection string example
v2	`LogLevel`	Optional	0	0-6	`LogLevel=6;`
v3	`LogLevel`	Optional	TRACE	OFF, ERROR, WARN, INFO, DEBUG, TRACE	`LogLevel=INFO;`

Query ID retrieval

In the version 2 driver, you unwrap a `Statement` instance to `com.interfaces.core.IStatementQueryInfoProvider`, an interface that has two methods: `#getPReparedQueryId` and `#getQueryId`. You can use these methods to obtain the query execution ID of a query that has run.

In the version 3 driver, you unwrap `Statement`, `PreparedStatement`, and `ResultSet` instances to the `com.amazon.athena.jdbc.AthenaResultSet` interface. The interface has one method: `#getQueryExecutionId`.

Amazon Athena JDBC 3.x connection parameters

Supported connection parameters are divided here into three sections: Basic connection parameters, Advanced connection parameters, and Authentication connection parameters. The Advanced connection parameters and Authentication connection parameters sections have subsections that group related parameters together.

Topics

- Basic connection parameters
- Advanced connection parameters
- Authentication connection parameters

Basic connection parameters

The following sections describe the basic connection parameters for the JDBC 3.x driver.

Region

The AWS Region where queries will be run. For a list of regions, see Amazon Athena endpoints and quotas.

Parameter name	Alias	Parameter type	Default value
Region	AwsRegion (deprecated)	Mandatory (but if not provided, will be searched using the DefaultAwsRegionProviderChain)	none

Catalog

The catalog that contains the databases and the tables that will be accessed with the driver. For information about catalogs, see DataCatalog.

Parameter name	Alias	Parameter type	Default value
Catalog	none	Optional	AwsDataCatalog

Database

The database where queries will run. Tables that are not explicitly qualified with a database name are resolved to this database. For information about databases, see Database.

Parameter name	Alias	Parameter type	Default value
Database	Schema	Optional	default

Workgroup

The workgroup in which queries will run. For information about workgroups, see WorkGroup.

Parameter name	Alias	Parameter type	Default value
WorkGroup	none	Optional	primary

Output location

The location in Amazon S3 where query results will be stored. For information about output location, see ResultConfiguration.

Parameter name	Alias	Parameter type	Default value
OutputLocation	S3OutputLocation (deprecated)	Mandatory (unless the workgroup specifies an output location)	none

Advanced connection parameters

The following sections describe the advanced connection parameters for the JDBC 3.x driver.

Topics

- Connection test
- Number of retries
- Network timeout

Result encryption parameters

Note the following points:

- The AWS KMS Key must be specified when `EncryptionOption` is `SSE_KMS` or `CSE_KMS`.
- The AWS KMS Key cannot be specified when `EncryptionOption` is not specified or when `EncryptionOption` is `SSE_S3`.

Encryption option

The type of encryption to be used for query results as they are stored in Amazon S3. For information about query result encryption, see EncryptionConfiguration in the *Amazon Athena API Reference*.

Parameter name	Alias	Parameter type	Default value	Possible values
EncryptionOption	S3OutputEncOption (deprecated)	Optional	none	SSE_S3, SSE_KMS, CSE_KMS

KMS Key

The KMS key ARN or ID, if `SSE_KMS` or `CSE_KMS` is chosen as the encryption option. For more information, see EncryptionConfiguration in the *Amazon Athena API Reference*.

Parameter name	Alias	Parameter type	Default value
KmsKey	S3OutputEncKMSKey (deprecated)	Optional	none

Result fetching parameters

Result fetcher

The fetcher that will be used to download query results.

The default result fetcher, `auto`, downloads query results directly from Amazon S3 without using the Athena APIs. When direct S3 download is not possible, like when query results are encrypted with the `CSE_KMS` option, it automatically falls back to use the `GetQueryResultsStream` API.

Using the `auto` fetcher is recommended in most situations. If your IAM policies, or S3 bucket policies use the s3:CalledVia condition to limit access to S3 objects requests from Athena, the `auto` fetcher first attempts to download the results from S3 and then falls back to use the `GetQueryResultsStream` API. In this situation, you can set the ResultFetcher to `GetQueryResultsStream` to avoid an extra API call.

Parameter name	Alias	Parameter type	Default value	Possible values
ResultFetcher	none	Optional	auto	auto, S3, GetQueryResults, GetQueryResultsStream

Fetch size

The value of this parameter is used as the minimum for internal buffers and as the target page size when fetching results. The value 0 (zero) means that the driver should use its defaults as described below. The maximum value is 1,000,000.

Parameter name	Alias	Parameter type	Default value
FetchSize	RowsToFetchPerBlock (deprecated)	Optional	0

- The `GetQueryResults` fetcher will always use a page size of 1,000, which is the maximum value supported by the API call. When the fetch size is higher than 1,000, multiple successive API calls are made to fill the buffer above the minimum.
- The `GetQueryResultsStream` fetcher will use the configured fetch size as the page size, or 10,000 by default.
- The `S3` fetcher will use the configured fetch size as the page size, or 10,000 by default.

Result configuration parameters

Expected bucket owner

The account ID of the expected s3 bucket owner. If the account ID that you provide does not match the actual owner of the bucket, the request fails. For more information about verifying s3 bucket owner, see Verifying bucket ownership.

Parameter name	Alias	Parameter type	Default value
ExpectedBucketOwner	none	Optional	none

Acl option

Indicates that an Amazon S3 canned ACL should be set to control ownership of stored query results. For more information about `AclOption`, see AclConfiguration.

Parameter name	Alias	Parameter type	Default value	Possible values
AclOption	none	Optional	none	BUCKET_OWNER_FULL_CONTROL

Query result reuse parameters

Enable result reuse

Specifies whether previous results for the same query can be reused when a query is run. For information about query result reuse, see ResultReuseByAgeConfiguration.

Parameter name	Alias	Parameter type	Default value
EnableResultReuseByAge	none	Optional	FALSE

Result reuse max age

The maximum age, in minutes, of a previous query result that Athena should consider for reuse. For information about result reuse max age, see ResultReuseByAgeConfiguration.

Parameter name	Alias	Parameter type	Default value
MaxResultReuseAgeInMinutes	none	Optional	60

Query execution polling parameters

Minimum query execution polling interval

The minimum time, in milliseconds, to wait before polling Athena for the query execution status.

Parameter name	Alias	Parameter type	Default value
MinQueryExecutionPollingIntervalMillis	MinQueryExecutionPollingInterval (deprecated)	Optional	100

Maximum query execution polling interval

The maximum time, in milliseconds, to wait before polling Athena for the query execution status.

Parameter name	Alias	Parameter type	Default value
MaxQueryExecutionPollingIntervalMillis	MaxQueryExecutionPollingInterval (deprecated)	Optional	5000

Query execution polling interval multiplier

The factor for increasing the polling period. By default, polling will begin with the value for `MinQueryExecutionPollingIntervalMillis` and double with each poll until it reaches the value for `MaxQueryExecutionPollingIntervalMillis`.

Parameter name	Alias	Parameter type	Default value
QueryExecutionPollingIntervalMultiplier	none	Optional	2

Endpoint override parameters

Athena endpoint override

The endpoint that the driver will use to make API calls to Athena.

Note the following points:

- If the `https://` or `http://` protocols are not specified in the provided URL, the driver inserts the `https://` prefix.
- If this parameter is not specified, the driver uses a default endpoint.

Parameter name	Alias	Parameter type	Default value
AthenaEndpoint	EndpointOverride (deprecated)	Optional	none

Athena streaming service endpoint override

The endpoint that the driver will use to download query results when it uses the Athena streaming service. The Athena streaming service is available on port 444.

Note the following points:

- If the `https://` or `http://` protocols are not specified in the provided URL, the driver inserts the `https://` prefix.
- If a port is not specified in the provided URL, the driver inserts the streaming service port 444.
- If the `AthenaStreamingEndpoint` parameter is not specified, the driver uses the `AthenaEndpoint` override. If neither the `AthenaStreamingEndpoint` nor the `AthenaEndpoint` override is specified, the driver uses a default streaming endpoint.

Parameter name	Alias	Parameter type	Default value
AthenaStreamingEndpoint	StreamingEndpointOverride (deprecated)	Optional	none

LakeFormation endpoint override

The endpoint that the driver will use for the Lake Formation service when using the AWS Lake Formation AssumeDecoratedRoleWithSAML API to retrieve temporary credentials. If this parameter is not specified, the driver uses a default Lake Formation endpoint.

Note the following points:

- If the `https://` or `http://` protocols are not specified in the provided URL, the driver inserts the `https://` prefix.

Parameter name	Alias	Parameter type	Default value
LakeFormationEndpoint	LfEndpointOverride (deprecated)	Optional	none

S3 endpoint override

The endpoint that the driver will use to download query results when it uses the Amazon S3 fetcher. If this parameter is not specified, the driver uses a default Amazon S3 endpoint.

Note the following points:

- If the `https://` or `http://` protocols are not specified in the provided URL, the driver inserts the `https://` prefix.

Parameter name	Alias	Parameter type	Default value
S3Endpoint	None	Optional	none

STS endpoint override

The endpoint that the driver will use for the AWS STS service when using the AWS STS AssumeRoleWithSAML API to retrieve temporary credentials. If this parameter is not specified, the driver uses a default AWS STS endpoint.

Note the following points:

- If the `https://` or `http://` protocols are not specified in the provided URL, the driver inserts the `https://` prefix.

Parameter name	Alias	Parameter type	Default value
StsEndpoint	StsEndpointOverride(deprecated)	Optional	none

Proxy configuration parameters

Proxy host

The URL of the proxy host. Use this parameter if you require Athena requests to go through a proxy.

> **Note**
>
> Make sure to include the protocol `https://` or `http://` at the beginning of the URL for `ProxyHost`.

Parameter name	Alias	Parameter type	Default value
ProxyHost	none	Optional	none

Proxy port

The port to be used on the proxy host. Use this parameter if you require Athena requests to go through a proxy.

Parameter name	Alias	Parameter type	Default value
ProxyPort	none	Optional	none

Proxy username

The username to authenticate on the proxy server. Use this parameter if you require Athena requests to go through a proxy.

Parameter name	Alias	Parameter type	Default value
ProxyUsername	ProxyUID (deprecated)	Optional	none

Proxy password

The password to authenticate on the proxy server. Use this parameter if you require Athena requests to go through a proxy.

Parameter name	Alias	Parameter type	Default value
ProxyPassword	ProxyPWD (deprecated)	Optional	none

Proxy-exempt hosts

A set of host names that the driver connects to without using a proxy when proxying is enabled (that is, when the `ProxyHost` and `ProxyPort` connection parameters are set). The hosts should be separated by the pipe (`|`) character (for example, `host1.com|host2.com`).

Parameter name	Alias	Parameter type	Default value
ProxyExemptHosts	NonProxyHosts	Optional	none

Proxy enabled for identity providers

Specifies whether a proxy should be used when the driver connects to an identity provider.

Parameter name	Alias	Parameter type	Default value
ProxyEnabledForIdP	UseProxyForIdP	Optional	FALSE

Logging parameters

This section describes parameters related to logging.

Log level

Specifies the level for the driver logging. Nothing is logged unless the `LogPath` parameter is also set.

Note

We recommend setting only the `LogPath` parameter unless you have special requirements. Setting only the `LogPath` parameter enables logging and uses the default `TRACE` log level. The `TRACE` log level provides the most detailed logging.

Parameter name	Alias	Parameter type	Default value	Possible values
LogLevel	none	Optional	TRACE	*OFF, ERROR, WARN, INFO, DEBUG, TRACE*

Log path

The path to a directory on the computer that runs the driver where driver logs will be stored. A log file with a unique name will be created within the specified directory. If set, enables driver logging.

Parameter name	Alias	Parameter type	Default value
LogPath	none	Optional	none

Application name

The name of the application that uses the driver. If a value for this parameter is specified, the value is included in the user agent string of the API calls that the driver makes to Athena.

Note

You can also set the application name by calling `setApplicationName` on the `DataSource` object.

Parameter name	Alias	Parameter type	Default value
ApplicationName	none	Optional	none

Connection test

If set to TRUE, the driver performs a connection test each time a JDBC connection is created, even if a query is not executed on the connection.

Parameter name	Alias	Parameter type	Default value
ConnectionTest	none	Optional	TRUE

> **Note**
>
> A connection test submits a SELECT 1 query to Athena to verify that the connection has been configured correctly. This means that two files will be stored in Amazon S3 (the result set and metadata), and additional charges can apply in accordance with the Amazon Athena pricing policy.

Number of retries

The maximum number of times the driver should resend a retriable request to Athena.

Parameter name	Alias	Parameter type	Default value
NumRetries	MaxErrorRetry (deprecated)	Optional	none

Network timeout

The network timeout controls the amount of time that the driver waits for a network connection to be established. This includes the time it takes to send API requests. In rare circumstances, it may be useful to change the network timeout. For example, you might want to increase the timeout for long garbage collection pauses. Setting this connection parameter is equivalent to using the `setNetworkTimeout` method on a `Connection` object.

Parameter name	Alias	Parameter type	Default value
NetworkTimeoutMillis	none	Optional	none

Authentication connection parameters

The Athena JDBC 3.x driver supports several authentication methods. The connection parameters that are required depend on the authentication method that you use.

Topics

- IAM credentials
- Default credentials
- AWS configuration profile credentials
- Instance profile credentials
- Custom credentials
- JWT credentials
- Azure AD credentials
- Okta credentials
- Ping credentials
- AD FS credentials
- Browser Azure AD credentials
- Browser SAML credentials
- DataZone IdC Credentials Provider
- DataZone IAM Credentials Provider

IAM credentials

You can use your IAM credentials with the JDBC driver to connect to Amazon Athena by setting the following connection parameters.

User

Your AWS access key ID. For information about access keys, see AWS security credentials in the *IAM User Guide*.

Parameter name	Alias	Parameter type	Default value
User	AccessKeyId	Required	none

Password

Your AWS secret key ID. For information about access keys, see AWS security credentials in the *IAM User Guide*.

Parameter name	Alias	Parameter type	Default value
Password	SecretAccessKey	Optional	none

Session token

If you use temporary AWS credentials, you must specify a session token. For information about temporary credentials, see Temporary security credentials in IAM in the *IAM User Guide*.

Parameter name	Alias	Parameter type	Default value
SessionToken	none	Optional	none

Default credentials

You can use the default credentials that you configure on your client system to connect to Amazon Athena by setting the following connection parameters. For information about using default credentials, see Using the Default Credential Provider Chain in the *AWS SDK for Java Developer Guide*.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `DefaultChain`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	*AWSCredentialsProviderClass (deprecated)*	Required	none	`DefaultChain`

AWS configuration profile credentials

You can use credentials stored in an AWS configuration profile by setting the following connection parameters. AWS configuration profiles are typically stored in files in the `~/.aws` directory). For information about AWS configuration profiles, see [Use profiles] in the *AWS SDK for Java Developer Guide*.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `ProfileCredentials`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	*AWSCredentialsProviderClass (deprecated)*	Required	none	`ProfileCredentials`

Profile name

The name of the AWS configuration profile whose credentials should be used to authenticate the request to Athena.

Parameter name	Alias	Parameter type	Default value
ProfileName	none	Required	none

> **Note**
>
> The profile name can also be specified as the value of the `CredentialsProviderArguments` parameter, although this use is deprecated.

Instance profile credentials

This authentication type is used on Amazon EC2 instances. An *instance profile* is a profile attached to an Amazon EC2 instance. Using an instance profile credentials provider delegates the management of AWS credentials to the Amazon EC2 Instance Metadata Service. This removes the need for developers to store credentials permanently on the Amazon EC2 instance or worry about rotating or managing temporary credentials.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `InstanceProfile`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	*AWSCredentialsProviderClass (deprecated)*	Required	none	`InstanceProfile`

Custom credentials

You can use this authentication type to provide your own credentials by using a Java class that implements the [AwsCredentialsProvider](https://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/auth/credentials/AwsCredentialsProvider.html) interface.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to the fully qualified class name of the custom class that implements the AwsCredentialsProvider interface. At runtime, that class must be on the Java class path of the application that uses the JDBC driver.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	*AWSCredentialsProviderClass (deprecated)*	Required	none	The fully qualified class name of the custom implementation of `AwsCredentialsProvider`

Credentials provider arguments

A comma-separated list of string arguments for the custom credentials provider constructor.

Parameter name	Alias	Parameter type	Default value
CredentialsProviderArguments	AwsCredentialsProviderArguments (deprecated)	Optional	none

JWT credentials

With this authentication type, you can use a JSON web token (JWT) obtained from an external identity provider as a connection parameter to authenticate with Athena. The external credentials provider must already be federated with AWS.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `JWT`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`JWT`

JWT web identity token

The JWT token obtained from an external federated identity provider. This token will be used to authenticate with Athena.

Parameter name	Alias	Parameter type	Default value
JwtWebIdentityToken	web_identity_token (deprecated)	Required	none

JWT role ARN

The Amazon Resource Name (ARN) of the role to assume. For information about assuming roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
JwtRoleArn	role_arn (deprecated)	Required	none

JWT role session name

The name of the session when you use JWT credentials for authentication. The name can be any name that you choose.

Parameter name	Alias	Parameter type	Default value
JwtRoleSessionName	role_session_name (deprecated)	Required	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRoleWithWebIdentity in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Azure AD credentials

A SAML-based authentication mechanism that enables authentication to Athena using the Azure AD identity provider. This method assumes that a federation has already been set up between Athena and Azure AD.

> **Note**
>
> Some of the parameter names in this section have aliases. The aliases are functional equivalents of the parameter names and have been provided for backward compatibility with the JDBC 2.x driver. Because the parameter names have been improved to follow a clearer, more consistent naming convention, we recommend that you use them instead of the aliases, which have been deprecated.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `AzureAD`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`AzureAD`

User

The email address of the Azure AD user to use for authentication with Azure AD.

Parameter name	Alias	Parameter type	Default value
User	UID (deprecated)	Required	none

Password

The password for the Azure AD user.

Parameter name	Alias	Parameter type	Default value
Password	PWD (deprecated)	Required	none

Azure AD tenant ID

The tenant ID of your Azure AD application.

Parameter name	Alias	Parameter type	Default value
AzureAdTenantId	tenant_id (deprecated)	Required	none

Azure AD client ID

The client ID of your Azure AD application.

Parameter name	Alias	Parameter type	Default value
AzureAdClientId	client_id (deprecated)	Required	none

Azure AD client secret

The client secret of your Azure AD application.

Parameter name	Alias	Parameter type	Default value
AzureAdClientSecret	client_secret (deprecated)	Required	none

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see [AssumeRole] in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Lake Formation enabled

Specifies whether to use the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

Okta credentials

A SAML-based authentication mechanism that enables authentication to Athena using the Okta identity provider. This method assumes that a federation has already been set up between Athena and Okta.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `Okta`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`Okta`

User

The email address of the Okta user to use for authentication with Okta.

Parameter name	Alias	Parameter type	Default value
User	UID (deprecated)	Required	none

Password

The password for the Okta user.

Parameter name	Alias	Parameter type	Default value
Password	PWD (deprecated)	Required	none

Okta host name

The URL for your Okta organization. You can extract the `idp_host` parameter from the **Embed Link** URL in your Okta application. For steps, see [Retrieve ODBC configuration information from Okta](). The first segment after `https://`, up to and including `okta.com`, is your IdP host (for example, `trial-1234567.okta.com` for a URL that starts with `https://trial-1234567.okta.com`).

Parameter name	Alias	Parameter type	Default value
OktaHostName	IdP_Host (deprecated)	Required	none

Okta application ID

The two-part identifier for your application. You can extract the application ID from the **Embed Link** URL in your Okta application. For steps, see Retrieve ODBC configuration information from Okta. The application ID is the last two segments of the URL, including the forward slash in the middle. The segments are two 20-character strings with a mix of numbers and upper and lowercase letters (for example, `Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4`).

Parameter name	Alias	Parameter type	Default value
OktaAppId	App_ID (deprecated)	Required	none

Okta application name

The name of your Okta application.

Parameter name	Alias	Parameter type	Default value
OktaAppName	App_Name (deprecated)	Required	none

Okta MFA type

If you have set up Okta to require multi-factor authentication (MFA), you need to specify the Okta MFA type and additional parameters depending on the second factor that you want to use.

Okta MFA type is the second authentication factor type (after the password) to use to authenticate with Okta. Supported second factors include push notifications delivered through the Okta Verify app and temporary one-time passwords (TOTPs) generated by Okta Verify, Google Authenticator, or sent through SMS. Individual organization security policies determine whether or not MFA is required for user login.

Parameter name	Alias	Parameter type	Default value	Possible values
OktaMfaType	okta_mfa_type (deprecated)	Required, if Okta is set up to require MFA	none	`oktaverifywithpush` , `oktaverifywithtotp` , `googleauthenticator` , `smsauthentication`

Okta phone number

The phone number to which Okta will send a temporary one-time password using SMS when the `smsauthentication` MFA type is chosen. The phone number must be a US or Canadian phone number.

Parameter name	Alias	Parameter type	Default value
OktaPhoneNumber	okta_phone_number (deprecated)	Required, if OktaMfaType is `smsauthentication`	none

Okta MFA wait time

The duration, in seconds, to wait for the user to acknowledge a push notification from Okta before the driver throws a timeout exception.

Parameter name	Alias	Parameter type	Default value
OktaMfaWaitTime	okta_mfa_wait_time (deprecated)	Optional	60

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Lake Formation enabled

Specifies whether to use the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

Ping credentials

A SAML-based authentication mechanism that enables authentication to Athena using the Ping Federate identity provider. This method assumes that a federation has already been set up between Athena and Ping Federate.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `Ping`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`Ping`

User

The email address of the Ping Federate user to use for authentication with Ping Federate.

Parameter name	Alias	Parameter type	Default value
User	UID (deprecated)	Required	none

Password

The password for the Ping Federate user.

Parameter name	Alias	Parameter type	Default value
Password	PWD (deprecated)	Required	none

PingHostName

The address for your Ping server. To find your address, visit the following URL and view the **SSO Application Endpoint** field.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

Parameter name	Alias	Parameter type	Default value
PingHostName	IdP_Host (deprecated)	Required	none

PingPortNumber

The port number to use to connect to your IdP host.

Parameter name	Alias	Parameter type	Default value
PingPortNumber	IdP_Port (deprecated)	Required	none

PingPartnerSpId

The service provider address. To find the service provider address, visit the following URL and view the **SSO Application Endpoint** field.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

Parameter name	Alias	Parameter type	Default value
PingPartnerSpId	Partner_SPID (deprecated)	Required	none

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see [AssumeRole](#) in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Lake Formation enabled

Specifies whether to use the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

AD FS credentials

A SAML-based authentication mechanism that enables authentication to Athena using Microsoft Active Directory Federation Services (AD FS). This method assumes that the user has already set up a federation between Athena and AD FS.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to ADFS.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	ADFS

User

The email address of the AD FS user to use for authentication with AD FS.

Parameter name	Alias	Parameter type	Default value
User	UID (deprecated)	Required for form-based authentication. Optional for Windows Integrated Authentication.	none

Password

The password for the AD FS user.

Parameter name	Alias	Parameter type	Default value
Password	PWD (deprecated)	Required for form-based authentication. Optional for Windows Integrated Authentication.	none

ADFS host name

The address for your AD FS server.

Parameter name	Alias	Parameter type	Default value
AdfsHostName	IdP_Host (deprecated)	Required	none

ADFS port number

The port number to use to connect to your AD FS server.

Parameter name	Alias	Parameter type	Default value
AdfsPortNumber	IdP_Port (deprecated)	Required	none

ADFS relying party

The trusted relying party. Use this parameter to override the AD FS relying party endpoint URL.

Parameter name	Alias	Parameter type	Default value
AdfsRelyingParty	LoginToRP (deprecated)	Optional	`urn:amazon:webservices`

ADFS WIA enabled

Boolean. Use this parameter to enable Windows Integrated Authentication (WIA) with AD FS.

Parameter name	Alias	Parameter type	Default value
AdfsWiaEnabled	none	Optional	`FALSE`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see [`AssumeRole`] in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see [`AssumeRole`] in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	`3600`

Lake Formation enabled

Specifies whether to use the [`AssumeDecoratedRoleWithSAML`] Lake Formation API action to retrieve temporary IAM credentials instead of the [`AssumeRoleWithSAML`] AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

Browser Azure AD credentials

Browser Azure AD is a SAML-based authentication mechanism that works with the Azure AD identity provider and supports multi-factor authentication. Unlike the standard Azure AD authentication mechanism, this mechanism does not require a user name, password, or client secret in the connection parameters. Like the standard Azure AD authentication mechanism, Browser Azure AD also assumes the user has already set up federation between Athena and Azure AD.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `BrowserAzureAD`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`BrowserAzureAD`

Azure AD tenant ID

The tenant ID of your Azure AD application

Parameter name	Alias	Parameter type	Default value
AzureAdTenantId	tenant_id (deprecated)	Required	none

Azure AD client ID

The client ID of your Azure AD application

Parameter name	Alias	Parameter type	Default value
AzureAdClientId	client_id (deprecated)	Required	none

Identity provider response timeout

The duration, in seconds, before the driver stops waiting for the SAML response from Azure AD.

Parameter name	Alias	Parameter type	Default value
IdpResponseTimeout	idp_response_timeout (deprecated)	Optional	120

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Lake Formation enabled

Specifies whether to use the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

Browser SAML credentials

Browser SAML is a generic authentication plugin that can work with SAML-based identity providers and supports multi-factor authentication.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `BrowserSaml`.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	`BrowserSaml`

Single sign-on login URL

The single sign-on URL for your application on the SAML-based identity provider.

Parameter name	Alias	Parameter type	Default value
SsoLoginUrl	login_url (deprecated)	Required	none

Listen port

The port number that is used to listen for the SAML response. This value should match the URL with which you configured the SAML-based identity provider (for example, `http://localhost:7890/athena`).

Parameter name	Alias	Parameter type	Default value
ListenPort	listen_port (deprecated)	Optional	`7890`

Identity provider response timeout

The duration, in seconds, before the driver stops waiting for the SAML response from Azure AD.

Parameter name	Alias	Parameter type	Default value
IdpResponseTimeout	idp_response_timeout (deprecated)	Optional	120

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see [AssumeRole](#) in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
PreferredRole	preferred_role (deprecated)	Optional	none

Role session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Parameter name	Alias	Parameter type	Default value
RoleSessionDuration	Duration (deprecated)	Optional	3600

Lake Formation enabled

Specifies whether to use the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action.

Parameter name	Alias	Parameter type	Default value
LakeFormationEnabled	none	Optional	FALSE

DataZone IdC Credentials Provider

An authentication mechanism that enables connecting to DataZone-governed data in Athena using IAM Identity Center.

Credentials provider

The credentials provider that will be used to authenticate requests to AWS. Set the value of this parameter to `DataZoneIdc`. Note that the `AWSCredentialsProviderClass` alias is deprecated; use the `CredentialsProvider` parameter name instead.

Parameter name	Alias	Parameter type	Default value	Value to use
CredentialsProvider	AWSCredentialsProviderClass (deprecated)	Required	none	DataZoneIdc

DataZone domain identifier

Identifier of the DataZone domain to use.

Parameter name	Alias	Parameter type	Default value
DataZoneDomainId	none	Required	none

DataZone environment identifier

Identifier of the DataZone environment to use.

Parameter name	Alias	Parameter type	Default value
DataZoneEnvironmentId	none	Required	none

DataZone domain region

The AWS Region where your DataZone domain is provisioned.

Parameter name	Alias	Parameter type	Default value
DataZoneDomainRegion	none	Required	none

Region

The AWS Region where your DataZone environment and Athena workgroup are provisioned.

Parameter name	Alias	Parameter type	Default value
Region	none	Required	none

IAM Identity Center issuer URL

The issuer URL of the IAM Identity Center instance that the DataZone domain uses.

Parameter name	Alias	Parameter type	Default value
IdentityCenterIssuerUrl	none	Required	none

DataZone endpoint override

The DataZone API endpoint to be used instead of the default for the provided AWS Region.

Parameter name	Alias	Parameter type	Default value
DataZoneEndpointOverride	none	Optional	none

Enable token caching

When enabled, allows the same IAM Identity Center access token to be used across driver connections. This prevents SQL tools that create multiple driver connections from launching multiple browser windows. If you enable this parameter, we recommend that you close the SQL tool immediately after using it to clear the token cache and require re-authentication.

Parameter name	Alias	Parameter type	Default value
EnableTokenCaching	none	Optional	FALSE

Listen port

The port number that listens for the IAM Identity Center response.

Parameter name	Alias	Parameter type	Default value
ListenPort	none	Optional	8000

Identity provider response time out

The duration, in seconds, before the driver stops waiting for the response from IAM Identity Center.

Parameter name	Alias	Parameter type	Default value
IdpResponseTimeout	none	Optional	120

DataZone IAM Credentials Provider

An authentication mechanism that uses IAM credentials to connect to DataZone-governed data in Athena.

DataZone domain identifier

Identifier of the DataZone domain to use.

Parameter name	Alias	Parameter type	Default value
DataZoneDomainId	none	Required	none

DataZone environment identifier

Identifier of the DataZone environment to use.

Parameter name	Alias	Parameter type	Default value
DataZoneEnvironmentId	none	Required	none

DataZone domain region

The AWS Region where your DataZone domain is provisioned.

Parameter name	Alias	Parameter type	Default value
DataZoneDomainRegion	none	Required	none

DataZone endpoint override

The DataZone API endpoint to use instead of the endpoint default for the provided AWS Region.

Parameter name	Alias	Parameter type	Default value
DataZoneEndpointOverride	none	Optional	none

User

Your AWS access key ID. For more information about access keys, see AWS security credentials in the *IAM User Guide*.

Parameter name	Alias	Parameter type	Default value
User	AccessKeyId	Optional	none

Password

Your AWS secret key ID. For more information about access keys, see AWS security credentials in the *IAM User Guide*.

Parameter name	Alias	Parameter type	Default value
Password	SecretAccessKey	Optional	none

Other JDBC 3.x configuration

The following sections describe some additional configuration settings for the JDBC 3.x driver.

Network timeout

The network timeout controls the amount of time in milliseconds that the driver waits for a network connection to be established. This includes the time it takes to send API requests. After this time, the driver throws a timeout exception. In rare circumstances, it may be useful to change the network timeout. For example, you might want to increase the timeout for long garbage collection pauses.

To set it, call the `setNetworkTimeout` method on a JDBC `Connection` object. This value can be changed during the lifecycle of the JDBC connection. For more information, see setNetworkTimeout in the Oracle JDBC API documentation. Using the `setNetworkTimeout` method is equivalent to setting the Network timeout connection parameter.

The following example sets the network timeout to 5000 milliseconds.

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
connection.setNetworkTimeout(null, 5000);
...
```

Query timeout

The amount of time, in seconds, the driver will wait for a query to complete on Athena after a query has been submitted. After this time, the driver attempts to cancel the submitted query and throws a timeout exception.

The query timeout cannot be set as a connection parameter. To set it, call the `setQueryTimeout` method on a JDBC `Statement` object. This value can be changed during the lifecycle of a JDBC statement. The default value of this parameter is `0` (zero). A value of `0` means that queries can run until they complete (subject to Service Quotas).

The following example sets the query timeout to 5 seconds.

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
```

```
Statement statement = connection.createStatement();
statement.setQueryTimeout(5);
...
```

Amazon Athena JDBC 3.x release notes

These release notes provide details of improvements and fixes in the Amazon Athena JDBC 3.x driver.

3.5.0

Released 2025-03-18

Improvements

- **Result configuration parameters** – Added support for two new connection parameters `ExpectedBucketOwner` and `AclOption`. For more information, see Result configuration parameters.
- **AWS SDK version** – The AWS SDK version used in the driver has been updated to 2.30.22.

3.4.0

Released 2025-02-18

Improvements

- **Result Fetcher** – The driver now automatically selects the fastest method to download query results. This removes the need to manually configure the fetcher in most situations. For more information, see Result fetching parameters.

Fixes

- **ResultSet** – The driver now handles iterating over the result sets of DDL statements that don't produce result objects on S3. It also returns an empty `ResultSet` object instead of null when `GetQueryResultsStream` returns a completely empty page.
- **ResultsStream** – The result streaming has been optimized by removing unnecessary calls to count the number of rows in internal buffers.
- **getTables** – The `GetTables` call has been optimized by handling table types based on `ListTableMetadata` and `GetTableMetadata` responses.

3.3.0

Released 2024-10-30

Improvements

- **DataZone authentication –** Added support for the DataZone authentication plugins `DataZoneIdC` and `DataZoneIAM`. For more information, see DataZone IdC Credentials Provider and DataZone IAM Credentials Provider.
- **Network timeout –** The network timeout can now be set using the `NetworkTimeoutMillis` connection parameter. Previously it could be set only on the `Connection` object itself. For more information, see Network timeout.

Fixes

- **S3 empty object handling –** The driver now handles empty objects in the S3 fetcher instead of throwing an Amazon S3 Range Not Satisfiable exception.
- **Logging –** The driver no longer logs the message Items requested for query execution [...], but subscription is cancelled after consuming query results.
- **Empty parameter strings –** The driver now handles empty strings present in a connection parameter as if the parameter were not present. This resolves issues that occurred when some BI tools inadvertently passed empty strings that caused unintended authentication attempts.

3.2.2

Released 2024-07-29

Improvements

- **Data type mapping –** Improved the compliance with the JDBC spec by changing how the driver maps the `tinyint`, `smallint`, `row`, and `struct` data types to Java objects.
- **AWS SDK version update –** The AWS SDK version used in the driver has been updated to 2.26.23.

Fixes

- **Comments –** Fixed an issue with line comments at the end of a statement.

- **Database listing** – Fixed an issue in which listing databases could enter an infinite loop when the last page returned by the paginated `ListDatabases` API was empty.

3.2.1

Released 2024-07-03

Improvements

- **JWT credentials provider** – Added support for user-specified session durations. For more information, see Role session duration.

Fixes

- **Thread pool** – Created one `ThreadPoolExecutor` per connection for asynchronous tasks to avoid using the `ForkJoin` pool.
- **Credential providers** – The proxy host is now parsed to get the scheme and host when the HTTP client is configured for external IdPs.
- **Default credentials provider** – Ensured the default credentials provider can't be closed by client code.
- **getColumns** – Fixed an `ORDINAL_COLUMN` column property issue in the `DatabaseMetaData#getColumns` method.
- **ResultSet** – Added support for `Infinity`, `-Infinity`, and `NaN` to `ResultSet.` Fixed a discrepancy between the column type returned from catalog operations and the result set of a completed query.

3.2.0

Released 2024-04-26

Improvements

- **Catalog operation performance** – Performance has been improved for catalog operations that do not use wildcard characters.
- **Minimum polling interval change** – The minimum polling interval default has been modified to reduce the number of API calls the driver makes to Athena. Query completions are still detected as soon as possible.

- **BI tool discoverability** – The driver has been made more easily discoverable for business intelligence tools.
- **Data type mapping** – Data type mapping to the Athena `binary`, `array`, and `struct` DDL data types has been improved.
- **AWS SDK version** – The AWS SDK version used in the driver has been updated to 2.25.34.

Fixes

- **Federated catalog table listings** – Fixed an issue that caused federated catalogs to return an empty list of tables.
- **getSchemas** – Fixed an issue that caused the JDBC [DatabaseMetaData#getSchemas](#) method to fetch databases only from the default catalog instead of from all catalogs.
- **getColumns** – Fixed an issue that caused a null catalog to be returned when the JDBC [DatabaseMetaData#getColumns](#) method was called with a null catalog name.

3.1.0

Released 2024-02-15

Improvements

- Support added for Microsoft Active Directory Federation Services (AD FS) Windows Integrated Authentication and form-based authentication.
- For backwards compatibility with version 2.x, the `awsathena` JDBC sub-protocol is now accepted but produces a deprecation warning. Use the `athena` JDBC sub-protocol instead.
- `AwsDataCatalog` is now the default for the catalog parameter, and `default` is the default for the database parameter. These changes ensure that correct values for the current catalog and database are returned instead of null.
- In conformance with the JDBC specification, `IS_AUTOINCREMENT` and `IS_GENERATEDCOLUMN` now return an empty string instead of `NO`.
- The Athena `int` data type now maps to the same JDBC type as Athena `integer` instead of to `other`.
- When the column metadata from Athena does not contain the optional `precision` and `scale` fields, the driver now returns zero for the corresponding values in a `ResultSet` column.
- The AWS SDK version has been updated to 2.21.39.

Fixes

- Fixed an issue with `GetQueryResultsStream` that caused an exception to occur when plain text results from Athena had a column count inconsistent with the column count in Athena result metadata.

3.0.0

Released 2023-11-16

The Athena JDBC 3.x driver is the new generation driver offering better performance and compatibility. The JDBC 3.x driver supports reading query results directly from Amazon S3, which improves the performance of applications that consume large query results. The new driver also has fewer third-party dependencies, which makes integration with BI tools and custom applications easier.

Previous versions of the Athena JDBC 3.x driver

We highly recommended that you use the latest version of the JDBC 3.x driver. The latest version of the driver contains the most recent improvements and fixes. Use an older version only if your application experiences incompatibilities with the latest version.

JDBC driver uber jar

The following download packages the driver and all its dependencies in the same `.jar` file. This download is commonly used for third-party SQL clients.

- 3.4.0 uber jar
- 3.3.0 uber jar
- 3.2.2 uber jar
- 3.2.1 uber jar
- 3.2.0 uber jar
- 3.1.0 uber jar
- 3.0.0 uber jar

JDBC driver lean jar

The following download is a `.zip` file that contains the lean `.jar` for the driver and separate `.jar` files for the driver's dependencies. This download is commonly used for custom applications that might have dependencies that conflict with the dependencies that the driver uses. This download is useful if you want to choose which of the driver dependencies to include with the lean jar, and which to exclude if your custom application already contains one or more of them.

- 3.4.0 lean jar
- 3.3.0 lean jar
- 3.2.2 lean jar
- 3.2.1 lean jar
- 3.2.0 lean jar
- 3.1.0 lean jar
- 3.0.0 lean jar

Athena JDBC 2.x driver

You can use a JDBC connection to connect Athena to business intelligence tools and other applications, such as SQL workbench. To do this, use the Amazon S3 links on this page to download, install, and configure the Athena JDBC 2.x driver. For information about building the JDBC connection URL, see the downloadable JDBC driver installation and configuration guide. For permissions information, see Control access through JDBC and ODBC connections. To submit feedback regarding the JDBC driver, email athena-feedback@amazon.com. Starting with version 2.0.24, two versions of the driver are available: one that includes the AWS SDK, and one that does not.

> **Important**
>
> When you use the JDBC driver, be sure to note the following requirements:
>
> - **Open port 444** – Keep port 444, which Athena uses to stream query results, open to outbound traffic. When you use a PrivateLink endpoint to connect to Athena, ensure that the security group attached to the PrivateLink endpoint is open to inbound traffic on port 444. If port 444 is blocked, you may receive the error message [Simba][AthenaJDBC] (100123) An error has occurred. Exception during column initialization.

- **athena:GetQueryResultsStream policy –** Add the `athena:GetQueryResultsStream` policy action to the IAM principals that use the JDBC driver. This policy action is not exposed directly with the API. It is used only with the ODBC and JDBC drivers as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess.
- **Using the JDBC driver for multiple data catalogs –** To use the JDBC driver for multiple data catalogs with Athena (for example, when using an external Hive metastore or federated queries), include `MetadataRetrievalMethod=ProxyAPI` in your JDBC connection string.
- **4.1 drivers –** Starting in 2023, driver support for JDBC version 4.1 is discontinued. No further updates will be released. If you are using a JDBC 4.1 driver, migration to the 4.2 driver is highly recommended.

JDBC 2.x driver with AWS SDK

The JDBC driver version 2.2.1 complies with the JDBC API 4.2 data standard and requires JDK 8.0 or later. For information about checking the version of Java Runtime Environment (JRE) that you use, see the Java documentation.

Use the following link to download the JDBC 4.2 driver `.jar` file.

- AthenaJDBC42-2.2.1.1000.jar

The following `.zip` file download contains the `.jar` file for JDBC 4.2 and includes the AWS SDK and the accompanying documentation, release notes, licenses, and agreements.

- SimbaAthenaJDBC-2.2.1.1000.zip

JDBC 2.x driver without AWS SDK

The JDBC driver version 2.2.1 complies with the JDBC API 4.2 data standard and requires JDK 8.0 or later. For information about checking the version of Java Runtime Environment (JRE) that you use, see the Java documentation.

Use the following link to download the JDBC 4.2 driver `.jar` file without the AWS SDK.

- AthenaJDBC42-2.2.1.1001.jar

The following `.zip` file download contains the `.jar` file for JDBC 4.2 and the accompanying documentation, release notes, licenses, and agreements. It does not include the AWS SDK.

- SimbaAthenaJDBC-2.2.1.1001.zip

JDBC 2.x driver release notes, license agreement, and notices

After you download the version you need, read the release notes, and review the License Agreement and Notices.

- Release notes
- License agreement
- Notices
- Third-party licenses

JDBC 2.x driver documentation

Download the following documentation for the driver:

- JDBC driver installation and configuration guide. Use this guide to install and configure the driver.
- JDBC driver migration guide. Use this guide to migrate from previous versions to the current version.

Connect to Amazon Athena with ODBC

Amazon Athena offers two ODBC drivers, versions 1.x and 2.x. The Athena ODBC 2.x driver is a new alternative that supports Linux, macOS ARM, macOS Intel, and Windows 64-bit systems. The Athena 2.x driver supports all authentication plugins that the 1.x ODBC driver supports, and almost all connection parameters are backward-compatible.

- To download the ODBC 2.x driver, see Amazon Athena ODBC 2.x.
- To download the ODBC 1.x driver, see Athena ODBC 1.x driver.

Topics

- Amazon Athena ODBC 2.x

- Athena ODBC 1.x driver
- Use the Amazon Athena Power BI connector

Amazon Athena ODBC 2.x

You can use an ODBC connection to connect to Amazon Athena from many third-party SQL client tools and applications. You set up the ODBC connection on your client computer.

Considerations and limitations

For information on migrating from the Athena ODBC 1.x driver to the Athena 2.x ODBC driver, see Migrate to the ODBC 2.x driver.

ODBC 2.x driver download

To download the Amazon Athena 2.x ODBC driver, visit the links on this page.

Important

When you use the ODBC 2.x driver, be sure to note the following requirements:

- **Open port 444** – Keep port 444, which Athena uses to stream query results, open to outbound traffic. When you use a PrivateLink endpoint to connect to Athena, ensure that the security group attached to the PrivateLink endpoint is open to inbound traffic on port 444.
- **athena:GetQueryResultsStream policy** – Add the `athena:GetQueryResultsStream` policy action to the IAM principals that use the ODBC driver. This policy action is not exposed directly with the API. It is used only with the ODBC and JDBC drivers as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess.

Linux

Driver version	Download link
ODBC 2.0.4.0 for Linux 64-bit	Linux 64 bit ODBC driver 2.0.4.0

macOS (ARM)

Driver version	Download link
ODBC 2.0.4.0 for macOS 64-bit (ARM)	macOS 64 bit ODBC driver 2.0.4.0 (ARM)

macOS (Intel)

Driver version	Download link
ODBC 2.0.4.0 for macOS 64-bit (Intel)	macOS 64 bit ODBC driver 2.0.4.0 (Intel)

Windows

Driver version	Download link
ODBC 2.0.4.0 for Windows 64-bit	Windows 64 bit ODBC driver 2.0.4.0

Licenses

- AWS license
- Third party license

Topics

- Get started with the ODBC 2.x driver
- Athena ODBC 2.x connection parameters
- Migrate to the ODBC 2.x driver
- Troubleshoot the ODBC 2.x driver
- Amazon Athena ODBC 2.x release notes

Get started with the ODBC 2.x driver

Use the information in this section to get started with the Amazon Athena ODBC 2.x driver. The driver is supported on the Windows, Linux, and macOS operating systems.

Topics

- Windows
- Linux
- macOS

Windows

If you want to use a Windows client computer to access Amazon Athena, the Amazon Athena ODBC driver is required.

Windows system requirements

Install the Amazon Athena ODBC driver on client computers that will access Amazon Athena databases directly instead of using a web browser.

The Windows system you use must meet the following requirements:

- You have administrator rights
- One of the following operating systems:
  - Windows 11, 10, or 8.1
  - Windows Server 2019, 2016, or 2012
  - Supported processor architecture : x86_64 (64-bit)
- At least 100 MB of available disk space
- Microsoft Visual C++ Redistributable for Visual Studio for 64-bit Windows is installed.

Installing the Amazon Athena ODBC driver

To download and install the Amazon Athena ODBC driver for Windows

1. Download the `AmazonAthenaODBC-2.x.x.x.msi` installation file.
2. Launch the installation file, and then choose **Next**.

3. To accept the terms of the license agreement, select the check box, and then choose **Next**.
4. To change the installation location, choose **Browse**, browse to the desired folder, and then choose **OK**.
5. To accept the installation location, choose **Next**.
6. Choose **Install**.
7. When the installation completes, choose **Finish**.

Ways to set driver configuration options

To control the behavior of the Amazon Athena ODBC driver in Windows, you can set driver configuration options in the following ways:

- In the **ODBC Data Source Administrator** program when you configure a data source name (DSN).
- By adding or changing Windows registry keys in the following location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\YOUR_DSN_NAME
```

- By setting driver options in the connection string when you connect programmatically.

Configuring a data source name on Windows

After you download and install the ODBC driver, you must add a data source name (DSN) entry to the client computer or Amazon EC2 instance. SQL client tools use this data source to connect to and query Amazon Athena.

To create a system DSN entry

1. From the Windows **Start** menu, right-click **ODBC Data Sources (64 bit)**, and then choose **More**, **Run as administrator**.
2. In the **ODBC Data Source Administrator**, choose the **Drivers** tab.
3. In the **Name** column, verify that **Amazon Athena ODBC (x64)** is present.
4. Do one of the following:
    - To configure the driver for all users on the computer, choose the **System DSN** tab. Because applications that use a different account to load data might not be able to detect user DSNs from another account, we recommend the system DSN configuration option.

> **Note**
>
> Using the **System DSN** option requires administrative privileges.

- To configure the driver for your user account only, choose the **User DSN** tab.

5. Choose **Add**. The **Create New Data Source** dialog box opens.
6. Choose **Amazon Athena ODBC (x64)**, and then choose **Finish**.
7. In the **Amazon Athena ODBC Configuration** dialog box, enter the following information. For detailed information about these options, see Main ODBC 2.x connection parameters.
   - For **Data Source Name**, enter a name that you want to use to identify the data source.
   - For **Description**, enter a description to help you identify the data source.
   - For **Region**, enter the name of the AWS Region that you will use Athena in (for example, **`us-west-1`**).
   - For **Catalog**, enter the name of the Amazon Athena catalog. The default is **AwsDataCatalog**, which is used by AWS Glue.
   - For **Database**, enter the name of the Amazon Athena database. The default is **default**.
   - For **Workgroup**, enter the name of the Amazon Athena workgroup. The default is **primary**.
   - For **S3 Output Location**, enter the location in Amazon S3 where the query results will be stored (for example, **`s3://amzn-s3-demo-bucket/`**).
   - (Optional) For **Encryption Options**, choose an encryption option. The default is `NOT_SET`.
   - (Optional) For **KMS Key**, choose an encryption KMS key if required.
8. To specify configuration options for IAM authentication, choose **Authentication Options.**
9. Enter the following information:
   - For **Authentication Type**, choose **IAM Credentials**. This is the default. For more information about available authentication types, see Authentication options.
   - For **Username**, enter a user name.
   - For **Password**, enter a password.
   - For **Session Token**, enter a session token if you want to use temporary AWS credentials. For information about temporary credentials, see Using temporary credentials with AWS resources in the *IAM User Guide*.
10. Choose **OK**.

11. At the bottom of the **Amazon Athena ODBC Configuration** dialog box, choose **Test**. If the client computer connects successfully to Amazon Athena, the **Connection test** box reports **Connection successful**. If not, the box reports **Connection failed** with corresponding error information.

12. Choose **OK** to close the connection test. The data source that you created now appears in the list of data source names.

Using a DSN-less connection on Windows

You can use a DSN-less connection to connect to a database without a Data Source Name (DSN). The following example shows a connection string for the Amazon Athena ODBC (x64) ODBC driver that connects to Amazon Athena.

```
DRIVER={Amazon Athena ODBC (x64)};Catalog=AwsDataCatalog;AwsRegion=us-
west-1;Schema=test_schema;S3OutputLocation=
s3://amzn-s3-demo-bucket/;AuthenticationType=IAM Credentials;UID=YOUR_UID;PWD=YOUR_PWD;
```

Linux

If you want use a Linux client computer to access Amazon Athena, the Amazon Athena ODBC driver is required.

Linux system requirements

Each Linux client computer where you install the driver must meet the following requirements.

- You have root access.
- Use one of the following Linux distributions:
  - Red Hat Enterprise Linux (RHEL) 7 or 8
  - CentOS 7 or 8.
- Have 100 MB of disk space available.
- Use version 2.3.1 or later of unixODBC.
- Use version 2.26 or later of the GNU C Library (glibc).

Installing the ODBC data connector on Linux

Use the following procedure to install the Amazon Athena ODBC driver on a Linux operating system.

To install the Amazon Athena ODBC driver on Linux

1. Enter one of the following commands:

   ```
   sudo rpm -Uvh AmazonAthenaODBC-2.X.Y.Z.rpm
   ```

   or

   ```
   sudo yum --nogpgcheck localinstall AmazonAthenaODBC-2.X.Y.Z.rpm
   ```

2. After the installation finishes, enter one of the following commands to verify that the driver is installed:

   - ```
     yum list | grep amazon-athena-odbc-driver
     ```

     Output:

     ```
     amazon-athena-odbc-driver.x86_64 2.0.2.1-1.amzn2int installed
     ```

   - ```
     rpm -qa | grep amazon
     ```

     Output:

     ```
     amazon-athena-odbc-driver-2.0.2.1-1.amzn2int.x86_64
     ```

Configuring a data source name on Linux

After the driver is installed, you can find example `.odbc.ini` and `.odbcinst.ini` files in the following location:

- `/opt/athena/odbc/ini/`.

Use the `.ini` files in this location as examples for configuring the Amazon Athena ODBC driver and data source name (DSN).

> **Note**
>
> By default, ODBC driver managers use the hidden configuration files `.odbc.ini` and `.odbcinst.ini`, which are located in the home directory.

To specify the path to the `.odbc.ini` and `.odbcinst.ini` files using unixODBC, perform the following steps.

To specify ODBC `.ini` file locations using unixODBC

1. Set `ODBCINI` to the full path and file name of the `odbc.ini` file, as in the following example.

   ```
   export ODBCINI=/opt/athena/odbc/ini/odbc.ini
   ```

2. Set `ODBCSYSINI` to the full path of the directory that contains the `odbcinst.ini` file, as in the following example.

   ```
   export ODBCSYSINI=/opt/athena/odbc/ini
   ```

3. Enter the following command to verify that you are using the unixODBC driver manager and the correct `odbc*.ini` files:

   ```
   username % odbcinst -j
   ```

   Sample output

   ```
   unixODBC 2.3.1
   DRIVERS............: /opt/athena/odbc/ini/odbcinst.ini
   SYSTEM DATA SOURCES: /opt/athena/odbc/ini/odbc.ini
   FILE DATA SOURCES..: /opt/athena/odbc/ini/ODBCDataSources
   USER DATA SOURCES..: /opt/athena/odbc/ini/odbc.ini
   SQLULEN Size.......: 8
   SQLLEN Size........: 8
   SQLSETPOSIROW Size.: 8
   ```

4. If you want to use a data source name (DSN) to connect to your data store, configure the `odbc.ini` file to define data source names (DSNs). Set the properties in the `odbc.ini` file to create a DSN that specifies the connection information for your data store, as in the following example.

```
[ODBC Data Sources]
athena_odbc_test=Amazon Athena ODBC (x64)

[ATHENA_WIDE_SETTINGS]  # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0              # To enable ODBC driver logs, set this to 1.
UseAwsLogger=0          # To enable AWS-SDK logs, set this to 1.
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.

[athena_odbc_test]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
AwsRegion=us-west-1
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://amzn-s3-demo-bucket/
```

5. Configure the `odbcinst.ini` file, as in the following example.

```
[ODBC Drivers]
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
Setup=/opt/athena/odbc/lib/libathena-odbc.so
```

6. After you install and configure the Amazon Athena ODBC driver, use the unixODBC `isql` command-line tool to verify the connection, as in the following example.

```
username % isql -v "athena_odbc_test"
+---------------------------------------+
| Connected!                            |
|                                       |
```

```
| sql-statement                          |
| help [tablename]                       |
| quit                                   |
|                                        |
+----------------------------------------+
SQL>
```

Verify the ODBC driver signature

Important

We recommend verifying the Athena ODBC driver RPM signature before installing it on your machine.

Follow these steps to verify the signature of the Athena ODBC driver RPM package:

1. **Prepare the templates**

   Prepare the commands with appropriate public key, RPM signature, and the corresponding access link to the RPM scripts hosted in Amazon S3 buckets. You must download the following to your device.

   - Athena ODBC driver
   - Public Key
   - Athena ODBC RPM signature

2. Download the Athena ODBC driver, public key, and Athena ODBC RPM signature to your device.
3. Run the following command to verify ODBC driver signature:

   ```
   openssl dgst -sha256 -verify public_key.pem -signature signature.bin
    AmazonAthenaODBC-2.0.4.0.rpm
   ```

   If verification passes, you will see a message similar to `Verified OK`. This means you can now proceed to install the Athena ODBC driver.

   If it fails with a message `Verification Failure`, it means that the signature on RPM has been tampered. Ensure that all the three files mentioned in step 1 are present, the paths are correctly specified ,and the files haven't been modified since download and then retry the verification process.

macOS

If you want to use a macOS client computer to access Amazon Athena, the Amazon Athena ODBC driver is required.

macOS system requirements

Each macOS computer where you install the driver must meet the following requirements.

- Use macOS version 14 or later.
- Have 100 MB of disk space available.
- Use version 3.52.16 or later of iODBC.

Installing the ODBC data connector on macOS

Use the following procedure to download and install the Amazon Athena ODBC driver for macOS operating systems.

To download and install the Amazon Athena ODBC driver for macOS

1. Download the `.pkg` package file.
2. Double-click the `.pkg` file.
3. Follow the steps in the wizard to install the driver.
4. On the **License Agreement** page, press **Continue**, and then choose **Agree**.
5. Choose **Install**.
6. When the installation completes, choose **Finish**.
7. Enter the following command to verify that the driver is installed:

```
> pkgutil --pkgs | grep athenaodbc
```

   Depending on your system, the output can look like one of the following.

```
com.amazon.athenaodbc-x86_64.Config
com.amazon.athenaodbc-x86_64.Driver
```

or

```
com.amazon.athenaodbc-arm64.Config
com.amazon.athenaodbc-arm64.Driver
```

Configuring a data source name on macOS

After the driver is installed, you can find example `.odbc.ini` and `.odbcinst.ini` files in the following locations:

- Intel processor computers: `/opt/athena/odbc/x86_64/ini/`
- ARM processor computers: `/opt/athena/odbc/arm64/ini/`

Use the `.ini` files in this location as examples for configuring the Amazon Athena ODBC driver and data source name (DSN).

> **Note**
>
> By default, ODBC driver managers use the hidden configuration files `.odbc.ini` and `.odbcinst.ini`, which are located in the home directory.

To specify the path to the `.odbc.ini` and `.odbcinst.ini` files using the iODBC driver manager, perform the following steps.

To specify ODBC `.ini` file locations using iODBC driver manager

1. Set `ODBCINI` to the full path and file name of the `odbc.ini` file.

   - For macOS computers that have Intel processors, use the following syntax.

     ```
     export ODBCINI=/opt/athena/odbc/x86_64/ini/odbc.ini
     ```

   - For macOS computers that have ARM processors, use the following syntax.

     ```
     export ODBCINI=/opt/athena/odbc/arm64/ini/odbc.ini
     ```

2. Set `ODBCSYSINI` to the full path and file name of the `odbcinst.ini` file.

- For macOS computers that have Intel processors, use the following syntax.

```
export ODBCSYSINI=/opt/athena/odbc/x86_64/ini/odbcinst.ini
```

- For macOS computers that have ARM processors, use the following syntax.

```
export ODBCSYSINI=/opt/athena/odbc/arm64/ini/odbcinst.ini
```

3. If you want to use a data source name (DSN) to connect to your data store, configure the odbc.ini file to define data source names (DSNs). Set the properties in the odbc.ini file to create a DSN that specifies the connection information for your data store, as in the following example.

```
[ODBC Data Sources]
athena_odbc_test=Amazon Athena ODBC (x64)

[ATHENA_WIDE_SETTINGS] # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0             # set to 1 to enable ODBC driver logs
UseAwsLogger=0         # set to 1 to enable AWS-SDK logs
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.

[athena_odbc_test]
Description=Amazon Athena ODBC (x64)
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
AwsRegion=us-west-1
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://amzn-s3-demo-bucket/
```

4. Configure the odbcinst.ini file, as in the following example.

```
[ODBC Drivers]
```

```
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
Setup=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
# Setup=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
```

5. After you install and configure the Amazon Athena ODBC driver, use the `iodbctest` command-line tool to verify the connection, as in the following example.

```
username@ % iodbctest
iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.1623.0502

Enter ODBC connect string (? shows list): ?

DSN                                 | Driver
------------------------------------------------------------------------------
athena_odbc_test                    | Amazon Athena ODBC (x64)

Enter ODBC connect string (? shows list): DSN=athena_odbc_test;
Driver: 2.0.2.1 (Amazon Athena ODBC Driver)

SQL>
```

Athena ODBC 2.x connection parameters

The **Amazon Athena ODBC Configuration** dialog box options include **Authentication Options**, **Advanced Options**, **Logging Options**, **Endpoint Overrides** and **Proxy Options**. For detailed information about each, visit the corresponding links.

- Main ODBC 2.x connection parameters
- Authentication options
- Advanced options
- Logging options
- Endpoint overrides

- Proxy options

Main ODBC 2.x connection parameters

The following sections describe each of the main connection parameters.

Data source name

Specifies the name of your data source.

Connection string name	Parameter type	Default value	Connection string example
DSN	Optional for DSN-less connection types	none	`DSN=AmazonAthenaOdbcUsWest1;`

Description

Contains description of your data source.

Connection string name	Parameter type	Default value	Connection string example
Description	Optional	none	`Description=Connection to Amazon Athena us-west-1;`

Catalog

Specifies the data catalog name. For more information about catalogs, see DataCatalog in the Amazon Athena API Reference.

Connection string name	Parameter type	Default value	Connection string example
Catalog	Optional	`AwsDataCatalog`	`Catalog=AwsDataCatalog;`

Region

Specifies the AWS Region. For information about AWS Regions, see [Regions and Availability Zones](#).

Connection string name	Parameter type	Default value	Connection string example
AwsRegion	Mandatory	`none`	`AwsRegion =us-west-1;`

Database

Specifies the database name. For more information about databases, see [Database](#) in the *Amazon Athena API Reference*.

Connection string name	Parameter type	Default value	Connection string example
Schema	Optional	`default`	`Schema=default;`

Workgroup

Specifies the workgroup name. For more information about workgroups, see [WorkGroup](#) in the *Amazon Athena API Reference*.

Connection string name	Parameter type	Default value	Connection string example
Workgroup	Optional	`primary`	`Workgroup =primary;`

Output location

Specifies the location in Amazon S3 where query results are stored. For more information about output location, see ResultConfiguration in the *Amazon Athena API Reference*.

Connection string name	Parameter type	Default value	Connection string example
S3OutputLocation	Mandatory	none	`S3OutputLocation=s3://amzn-s3-demo-bucket/;`

Encryption options

Dialog parameter name: Encryption options

Specifies encryption option. For more information about encryption options, see EncryptionConfiguration in the *Amazon Athena API Reference*.

Connection string name	Parameter type	Default value	Possible values	Connection string example
S3OutputEncOption	Optional	none	`NOT_SET, SSE_S3, SSE_KMS, CSE_KMS`	`S3OutputEncOption=SSE_S3;`

KMS key

Specifies a KMS key for encryption. For more information about encryption configuration for KMS Keys, see EncryptionConfiguration in the *Amazon Athena API Reference*.

Connection string name	Parameter type	Default value	Connection string example
S3OutputEncKMSKey	Optional	none	`S3OutputEncKMSKey=your_key;`

Connection test

ODBC Data Source Administrator provides a **Test** option that you can use to test your ODBC 2.x connection to Amazon Athena. For steps, see Configuring a data source name on Windows. When you test a connection, the ODBC driver calls the GetWorkGroup Athena API action. The call uses the authentication type and corresponding credentials provider that you specified to retrieve the credentials. There is no charge for the connection test when you use the ODBC 2.x driver. The test does not generate query results in your Amazon S3 bucket.

Authentication options

You can connect to Amazon Athena using the following authentication types. For all types, the connection string name is `AuthenticationType`, the parameter type is `Required`, and the default value is `IAM Credentials`. For information about the parameters for each authentication type, visit the corresponding link. For common authentication parameters, see Common authentication parameters.

Authentication type	Connection string example
IAM credentials	`AuthenticationType=IAM Credentials;`
IAM profile	`AuthenticationType=IAM Profile;`
AD FS	`AuthenticationType=ADFS;`
Azure AD	`AuthenticationType=AzureAD;`
Browser Azure AD	`AuthenticationType=BrowserAzureAD;`
Browser SAML	`AuthenticationType=BrowserSAML;`

Authentication type	Connection string example
Browser SSO OIDC	AuthenticationType=BrowserSSOOIDC;
Default credentials	AuthenticationType=Default Credentials;
External credentials	AuthenticationType=External Credentials;
Instance profile	AuthenticationType=Instance Profile;
JWT	AuthenticationType=JWT;
Okta	AuthenticationType=Okta;
Ping	AuthenticationType=Ping;

IAM credentials

You can use your IAM credentials to connect to Amazon Athena with the ODBC driver using the connection string parameters described in this section.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=IAM Credentials;`

User ID

Your AWS Access Key ID. For more information about access keys, see AWS security credentialsin the *IAM User Guide*.

Connection string name	Parameter type	Default value	Connection string example
UID	Required	none	`UID=AKIAIOSFODNN7EXAMPLE;`

Password

Your AWS secret key id. For more information about access keys, see AWS security credentialsin the *IAM User Guide*.

Connection string name	Parameter type	Default value	Connection string example
PWD	Required	none	`PWD=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE;`

Session token

If you use temporary AWS credentials, you must specify a session token. For information about temporary credentials, see Temporary security credentials in IAM in the *IAM User Guide*.

Connection string name	Parameter type	Default value	Connection string example
SessionToken	Optional	none	`SessionToken=AQoDYXdzEJr...<remainder of session token>;`

IAM profile

You can configure a named profile to connect to Amazon Athena using the ODBC driver. To use the credentials available in your hosting Amazon EC2 instance profile, set the `credential_source` parameter to `Ec2InstanceMetadata`. If you want to use a custom credentials provider in a named profile, specify a value for the `plugin_name` parameter in your profile configuration.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=IAM Profile;`

AWS profile

The profile name to use for your ODBC connection. For more information about profiles, see Using named profiles in the *AWS Command Line Interface User Guide*.

Connection string name	Parameter type	Default value	Connection string example
AWSProfile	Required	none	`AWSProfile=default;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. The preferred role parameter is used when the custom credentials provider is specified by the `plugin_name` parameter in your profile configuration. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information about session duration, see AssumeRole in the *AWS Security Token Service API Reference*. The session duration parameter is used when the custom credentials provider is specified by the `plugin_name` parameter in your profile configuration.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

Plugin name

Specifies the name of a custom credentials provider used in a named profile. This parameter can take the same values as those in the **Authentication Type** field of the ODBC Data Source Administrator, but is used only by `AWSProfile` configuration.

Connection string name	Parameter type	Default value	Connection string example
plugin_name	Optional	none	`plugin_name=AzureAD;`

AD FS

AD FS is a SAML based authentication plugin that works with the Active Directory Federation Service (AD FS) identity provider. The plugin supports Integrated Windows authentication and form-based authentication. If you use Integrated Windows Authentication, you can omit the user name and password. For information about configuring AD FS and Athena, see [Configure federated access to Amazon Athena for Microsoft AD FS users using an ODBC client](#).

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=ADFS;`

User ID

Your user name for connecting to the AD FS server. For Integrated Windows Authentication, you can omit the user name. If your AD FS setup requires a user name, you must provide it in the connection parameter.

Connection string name	Parameter type	Default value	Connection string example
UID	Optional for windows integrated authentication	`none`	`UID=domain\username;`

Password

Your password for connecting to the AD FS server. Like the user name field, you can omit the user name if you use Integrated Windows Authentication. If your AD FS setup requires a password, you must provide it in the connection parameter.

Connection string name	Parameter type	Default value	Connection string example
PWD	Optional for windows integrated authentication	none	PWD=password_3EXAMPLE;

Preferred role

The Amazon Resource Name (ARN) of the role to assume. If your SAML assertion has multiple roles, you can specify this parameter to choose the role to be assumed. This role should present in the SAML assertion. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	preferred_role=arn:aws:IAM::123456789012:id/user1;

Session duration

The duration, in seconds, of the role session. For more information about session duration, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	900	duration=900;

IdP host

The name of the AD FS service host.

Connection string name	Parameter type	Default value	Connection string example
idp_host	Require	none	`idp_host=<server-name>.<company.com>;`

IdP port

The port to use to connect to the AD FS host.

Connection string name	Parameter type	Default value	Connection string example
idp_port	Required	none	`idp_port=443;`

LoginToRP

The trusted relying party. Use this parameter to override the AD FS relying party endpoint URL.

Connection string name	Parameter type	Default value	Connection string example
LoginToRP	Optional	`urn:amazon:webservices`	`LoginToRP=trustedparty;`

Azure AD

Azure AD is a SAML-based authentication plugin that works with Azure AD identity provider. This plugin does not support multifactor authentication (MFA). If you require MFA support, consider using the `BrowserAzureAD` plugin instead.

Authentication Type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType =AzureAD;`

User ID

Your user name for connecting to Azure AD.

Connection string name	Parameter type	Default value	Connection string example
UID	Required	none	`UID=jane.doe@example.com;`

Password

Your password for connecting to Azure AD.

Connection string name	Parameter type	Default value	Connection string example
PWD	Required	none	`PWD=password_3EXAMPLE;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For information about ARN roles, see [AssumeRole](#) in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:iam::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information, see [AssumeRole](#) in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

Tenant ID

Specifies your application tenant ID.

Connection string name	Parameter type	Default value	Connection string example
idp_tenant	Required	none	`idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;`

Client ID

Specifies your application client ID.

Connection string name	Parameter type	Default value	Connection string example
client_id	Required	none	`client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;`

Client secret

Specifies your client secret.

Connection string name	Parameter type	Default value	Connection string example
client_secret	Required	none	`client_secret=zG12q~.xzG1xxxZ1wX1.~ZzXXX1XxkHZizeT1zzZ;`

Browser Azure AD

Browser Azure AD is a SAML based authentication plugin that works with Azure AD identity provider and supports multi-factor authentication. Unlike the standard Azure AD plugin, this plugin does not require a user name, password, or client secret in the connection parameters.

Authentication Type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=BrowserAzureAD;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. If your SAML assertion has multiple roles, you can specify this parameter to choose the role to be assumed. The role specified should be present in the SAML assertion. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information about session duration, see [AssumeRole](#) in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

Tenant ID

Specifies your application tenant ID.

Connection string name	Parameter type	Default value	Connection string example
idp_tenant	Required	none	`idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;`

Client ID

Specifies your application client ID.

Connection string name	Parameter type	Default value	Connection string example
client_id	Required	none	`client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;`

Timeout

The duration, in seconds, before the plugin stops waiting for the SAML response from Azure AD.

Connection string name	Parameter type	Default value	Connection string example
timeout	Optional	`120`	`timeout=90;`

Enable Azure file cache

Enables a temporary credentials cache. This connection parameter enables temporary credentials to be cached and reused between multiple processes. Use this option to reduce the number of opened browser windows when you use BI tools such as Microsoft Power BI.

Connection string name	Parameter type	Default value	Connection string example
browser_azure_cache	Optional	`1`	`browser_azure_cache=0;`

Browser SAML

Browser SAML is a generic authentication plugin that can work with SAML based identity providers and support multi-factor authentication. For detailed configuration information, see [Configure single sign-on using ODBC, SAML 2.0, and the Okta Identity Provider](#).

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=BrowserSAML;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. If your SAML assertion has multiple roles, you can specify this parameter to choose the role to be assumed. This role should be present in the SAML assertion. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

Login URL

The single sign-on URL that is displayed for your application.

Connection string name	Parameter type	Default value	Connection string example
login_url	Required	none	`login_url=https://trial-1234567.okta.com/app/trial-1234567_okta`

Connection string name	Parameter type	Default value	Connection string example
			browsersaml_1/zzz4izzzAzDFBzZz1234/sso/saml;

Listen port

The port number that is used to listen for the SAML response. This value should match the IAM Identity Center URL that you configured the IdP with (for example, `http://localhost:7890/athena`).

Connection string name	Parameter type	Default value	Connection string example
listen_port	Optional	7890	listen_port=7890;

Timeout

The duration, in seconds, before the plugin stops waiting for the SAML response from the identity provider.

Connection string name	Parameter type	Default value	Connection string example
timeout	Optional	120	timeout=90;

Browser SSO OIDC

Browser SSO OIDC is an authentication plugin that works with AWS IAM Identity Center. For information on enabling and using IAM Identity Center, see Step 1: Enable IAM Identity Center in the *AWS IAM Identity Center User Guide*.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType =BrowserSSOOIDC;`

IAM Identity Center Start URL

The URL for the AWS access portal. The IAM Identity Center [StartDeviceAuthorization](https://docs.aws.amazon.com/singlesignon/latest/OIDCAPIReference/API_StartDeviceAuthorization.html) API action uses this value for the `startUrl` parameter.

To copy the AWS access portal URL

1. Sign in to the AWS Management Console and open the AWS IAM Identity Center console at https://console.aws.amazon.com/singlesignon/.
2. In the navigation pane, choose **Settings**.
3. On the **Settings** page, under **Identity source**, choose the clipboard icon for **AWS access portal URL**.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_ start_url	Required	none	`sso_oidc_start_url=https:// app_id.awsapps.com/start;`

IAM Identity Center Region

The AWS Region where your SSO is configured. The `SSOOIDCClient` and `SSOClient` AWS SDK clients use this value for the `region` parameter.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_region	Required	none	`sso_oidc_region=us-east-1;`

Scopes

The list of scopes that are defined by the client. Upon authorization, this list restricts permissions when an access token is granted. The IAM Identity Center RegisterClient API action uses this value for the `scopes` parameter.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_scopes	Optional	none	`sso_oidc_scopes=scope1,scope2,scope3;`

Account ID

The identifier for the AWS account that is assigned to the user. The IAM Identity Center GetRoleCredentials API uses this value for the `accountId` parameter.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_account_id	Required	none	`sso_oidc_account_id=123456789123;`

Role name

The friendly name of the role that is assigned to the user. The name that you specify for this permission set appears in the AWS access portal as an available role. The IAM Identity Center [GetRoleCredentials](#) API action uses this value for the `roleName` parameter.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_role_name	Required	none	`sso_oidc_role_name=AthenaReadAccess;`

Timeout

The number of seconds the polling SSO API should check for the access token.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_timeout	Optional	`120`	`sso_oidc_timeout=60;`

Enable file cache

Enables a temporary credentials cache. This connection parameter enables temporary credentials to be cached and reused between multiple processes. Use this option to reduce the number of opened browser windows when you use BI tools such as Microsoft Power BI.

Connection string name	Parameter type	Default value	Connection string example
sso_oidc_cache	Optional	1	`sso_oidc_cache=0;`

Default credentials

You can use the default credentials that you configure on your client system to connect to Amazon Athena. For information about using default credentials, see Using the Default Credential Provider Chain in the *AWS SDK for Java Developer Guide*.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=Default Credentials;`

External credentials

External credentials is a generic authentication plugin that you can use to connect to any external SAML based identity provider. To use the plugin, you pass an executable file that returns a SAML response.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=External Credentials;`

Executable path

The path to the executable that has the logic of your custom SAML-based credential provider. The output of the executable must be the parsed SAML response from the identity provider.

Connection string name	Parameter type	Default value	Connection string example
ExecutablePath	Required	none	ExecutablePath=C:\Users *user_name* *external_ credential.exe*

Argument list

The list of arguments that you want to pass to the executable.

Connection string name	Parameter type	Default value	Connection string example
ArgumentList	Optional	none	`ArgumentList=` *`arg1 arg2 arg3`*

Instance profile

This authentication type is used on EC2 instances and is delivered through the Amazon EC2 metadata service.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType =Instance Profile;`

JWT

The JWT (JSON Web Token) plugin provides an interface that uses JSON Web Tokens to assume an Amazon IAM role. The configuration depends on the identity provider. For information about configuring federation for Google Cloud and AWS, see Configure workload identity federation with AWS or Azure in the Google Cloud documentation.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=JWT;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information about session duration, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

JSON web token

The JSON web token that is used to retrieve IAM temporary credentials using the AssumeRoleWithWebIdentity AWS STS API action. For information about generating JSON web tokens for Google Cloud Platform (GCP) users, see Using JWT OAuth tokens in the Google Cloud documentation.

Connection string name	Parameter type	Default value	Connection string example
web_identity_token	Required	none	web_identity_token=eyJhbGc...<remainder of token>;

Role session name

A name for the session. A common technique is to use the name or identifier of the user of your application as the role session name. This conveniently associates the temporary security credentials that your application uses with the corresponding user.

Connection string name	Parameter type	Default value	Connection string example
role_session_name	Required	none	role_session_name=familiarname;

Okta

Okta is a SAML-based authentication plugin that works with the Okta identity provider. For information about configuring federation for Okta and Amazon Athena, see Configure SSO for ODBC using the Okta plugin and Okta Identity Provider.

Authentication Type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=Okta;`

User ID

Your Okta user name.

Connection string name	Parameter type	Default value	Connection string example
UID	Required	none	`UID=jane.doe@org.com;`

Password

Your Okta user password.

Connection string name	Parameter type	Default value	Connection string example
PWD	Required	none	`PWD=oktauserpasswordexample;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information, see [AssumeRole](https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRole.html) in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	`900`	`duration=900;`

IdP host

The URL for your Okta organization. You can extract the `idp_host` parameter from the **Embed Link** URL in your Okta application. For steps, see Retrieve ODBC configuration information from Okta. The first segment after `https://`, up to and including `okta.com` is your IdP host (for example, `http://trial-1234567.okta.com`).

Connection string name	Parameter type	Default value	Connection string example
idp_host	Required	None	`idp_host=dev-99999999.okta.com;`

IdP port

The port number to use to connect to your IdP host.

Connection string name	Parameter type	Default value	Connection string example
idp_port	Required	None	`idp_port=443;`

Okta app ID

The two-part identifier for your application. You can extract the `app_id` parameter from the **Embed Link** URL in your Okta application. For steps, see [Retrieve ODBC configuration information from Okta](). The application ID is the last two segments of the URL, including the forward slash in the middle. The segments are two 20-character strings with a mix of numbers and upper and lowercase letters (for example, `Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4`).

Connection string name	Parameter type	Default value	Connection string example
app_id	Required	None	`app_id=0oa25kx8ze9A3example/alnexamplea0piaWa0g7;`

Okta app name

The name of the Okta application.

Connection string name	Parameter type	Default value	Connection string example
app_name	Required	None	`app_name=amazon_aws_redshift;`

Okta wait time

Specifies the duration in seconds to wait for the multifactor authentication (MFA) code.

Connection string name	Parameter type	Default value	Connection string example
okta_mfa_wait_time	Optional	`10`	`okta_mfa_wait_time=20;`

Okta MFA type

The MFA factor type. Supported types are Google Authenticator, SMS (Okta), Okta Verify with Push, and Okta Verify with TOTP. Individual organization security policies determine whether or not MFA is required for user login.

Connection string name	Parameter type	Default value	Possible values	Connection string example
okta_mfa_type	`Optional`	None	`googleauthenticator, smsauthentication, oktaverifywithpush, oktaverifywithtotp`	`okta_mfa_type=oktaverifywithpush;`

Okta phone number

The phone number to use with AWS SMS authentication. This parameter is required only for multifactor enrollment. If your mobile number is already enrolled, or if AWS SMS authentication is not used by the security policy, you can ignore this field.

Connection string name	Parameter type	Default value	Connection string example
okta_mfa_phone_number	Required for MFA enrollment, optional otherwise	None	`okta_mfa_phone_number=19991234567;`

Enable Okta file cache

Enables a temporary credentials cache. This connection parameter enables temporary credentials to be cached and reused between the multiple processes opened by BI applications. Use this option to avoid the Okta API throttling limit.

Connection string name	Parameter type	Default value	Connection string example
okta_cache	Optional	`0`	`okta_cache=1;`

Ping

Ping is a SAML based plugin that works with the PingFederate identity provider.

Authentication type

Connection string name	Parameter type	Default value	Connection string example
AuthenticationType	Required	`IAM Credentials`	`AuthenticationType=Ping;`

User ID

The user name for the PingFederate server.

Connection string name	Parameter type	Default value	Connection string example
UID	Required	none	`UID=pingusername@domain.com;`

Password

The password for the PingFederate server.

Connection string name	Parameter type	Default value	Connection string example
PWD	Required	none	`PWD=pingpassword;`

Preferred role

The Amazon Resource Name (ARN) of the role to assume. If your SAML assertion has multiple roles, you can specify this parameter to choose the role to be assumed. This role should be present in the SAML assertion. For more information about ARN roles, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
preferred_role	Optional	none	`preferred_role=arn:aws:iam::123456789012:id/user1;`

Session duration

The duration, in seconds, of the role session. For more information about session duration, see AssumeRole in the *AWS Security Token Service API Reference*.

Connection string name	Parameter type	Default value	Connection string example
duration	Optional	900	duration=900;

IdP host

The address for your Ping server. To find your address, visit the following URL and view the **SSO Application Endpoint** field.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

Connection string name	Parameter type	Default value	Connection string example
idp_host	Required	none	idp_host=ec2-1-83-65-12.compute-1.amazonaws.com;

IdP port

The port number to use to connect to your IdP host.

Connection string name	Parameter type	Default value	Connection string example
idp_port	Required	None	idp_port=443;

Partner SPID

The service provider address. To find the service provider address, visit the following URL and view the **SSO Application Endpoint** field.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

Connection string name	Parameter type	Default value	Connection string example
partner_spid	Required	None	`partner_spid=https://us-east-1.signin.aws.amazon.com/platform/saml/<...>;`

Ping URI param

Passes a URI argument for an authentication request to Ping. Use this parameter to bypass the Lake Formation single role limitation. Configure Ping to recognize the passed parameter, and verify that the role passed exists in the list of roles assigned to the user. Then, send a single role in the SAML assertion.

Connection string name	Parameter type	Default value	Connection string example
ping_uri_param	Optional	None	`ping_uri_param=role=my_iam_role;`

Common authentication parameters

The parameters in this section are common to the authentication types as noted.

Use Proxy for IdP

Enables communication between the driver and the IdP through the proxy. This option is available for the following authentication plugins:

- AD FS
- Azure AD
- Browser Azure AD
- Browser SSO OIDC
- JWT

- Okta
- Ping

Connection string name	Parameter type	Default value	Connection string example
UseProxyForIdP	Optional	0	UseProxyForIdP=1;

Use Lake Formation

Uses the AssumeDecoratedRoleWithSAML Lake Formation API action to retrieve temporary IAM credentials instead of the AssumeRoleWithSAML AWS STS API action. This option is available for the Azure AD, Browser Azure AD, Browser SAML, Okta, Ping, and AD FS authentication plugins.

Connection string name	Parameter type	Default value	Connection string example
LakeformationEnabled	Optional	0	LakeformationEnabled=1;

SSL insecure (IdP)

Disables SSL when communicating with the IdP. This option is available for the Azure AD, Browser Azure AD, Okta, Ping, and AD FS authentication plugins.

Connection string name	Parameter type	Default value	Connection string example
SSL_Insecure	Optional	0	SSL_Insecure=1;

Endpoint overrides

Athena endpoint override

The `endpointOverride ClientConfiguration` class uses this value override the default HTTP endpoint for the Amazon Athena client. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
EndpointOverride	Optional	none	`EndpointOverride=athena.us-west-2.amazonaws.com;`

Athena streaming endpoint override

The `ClientConfiguration.endpointOverride` method uses this value to override the default HTTP endpoint for the Amazon Athena streaming client. For more information, AWS Client configuration in the *AWS SDK for C++ Developer Guide*. The Athena Streaming service is available through port 444.

Connection string name	Parameter type	Default value	Connection string example
Streaming EndpointOverride	Optional	none	`StreamingEndpointOverride=athena.us-west-1.amazonaws.com:444;`

AWS STS endpoint override

The `ClientConfiguration.endpointOverride` method uses this value to override the default HTTP endpoint for the AWS STS client. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
StsEndpointOverride	Optional	none	`StsEndpointOverride=sts.us-west-1.amazonaws.com;`

Lake Formation endpoint override

The `ClientConfiguration.endpointOverride` method uses this value to override the default HTTP endpoint for the Lake Formation client. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
LakeFormationEndpointOverride	Optional	none	`LakeFormationEndpointOverride=lakeformation.us-west-1.amazonaws.com;`

SSO endpoint override

The `ClientConfiguration.endpointOverride` method uses this value to override the default HTTP endpoint for the SSO client. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
SSOEndpointOverride	Optional	none	`SSOEndpointOverride=portal.sso.us-east-2.amazonaws.com;`

SSO OIDC endpoint override

The `ClientConfiguration.endpointOverride` method uses this value to override the default HTTP endpoint for the SSO OIDC client. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
SSOOIDCEndpointOverride	Optional	none	`SSOOIDCEndpointOverride=oidc.us-east-2.amazonaws.com`

Advanced options

Fetch size

The maximum number of results (rows) to return in this request. For parameter information, see GetQuery MaxResults. For the streaming API, the maximum value is 10000000.

Connection string name	Parameter type	Default value	Connection string example
RowsToFetchPerBlock	Optional	`1000` for non-streaming `20000` for streaming	`RowsToFetchPerBlock=20000;`

Result fetcher

The default result fetcher downloads query results directly from Amazon S3 without going through the Athena API operations. When it detects situations where direct S3 download is not possible, it automatically falls back to using the `GetQueryResultsStream` API operation. For example, this happens when query results are encrypted with the `CSE_KMS` option.

Using the `auto` fetcher is recommended in most situations. However, if your IAM policies, or S3 bucket policies use the `s3:CalledVia` condition to limit access to S3 objects to requests from Athena, the auto fetcher first attempts to download the results from S3 and then falls back to using the `GetQueryResultsStream`. In this situation, you might want to set the `ResultFetcher` to `GetQueryResultsStream` to avoid an extra API call.

Note

The driver still recognizes the Enable streaming API (`UseResultsetStreaming=1;`) and Enable S3 fetcher (`EnableS3Fetcher=1;`) parameters. However, we encourage you to use `ResultFetcher` parameter for better experience.

Connection string name	Parameter type	Default value	Possible values	Connection string example
ResultFetcher	Optional	`auto`	`auto`, `S3`, `GetQueryResults` , `GetQueryResultsStream`	`ResultFetcher=auto`

Enable result reuse

Specifies if previous query results can be reused when the query is run. For parameter information, see ResultReuseByAgeConfiguration.

Connection string name	Parameter type	Default value	Connection string example
EnableResultReuse	Optional	`0`	`EnableResultReuse=1;`

Result reuse maximum age

Specifies, in minutes, the maximum age of a previous query result that Athena should consider for reuse. For parameter information, see ResultReuseByAgeConfiguration.

Connection string name	Parameter type	Default value	Connection string example
ReusedResultMaxAgeInMinutes	Optional	`60`	`ReusedResultMaxAgeInMinutes=90;`

Use multiple S3 threads

Fetches data from Amazon S3 using multiple threads. When this option is enabled, the result file stored in the Amazon S3 bucket is fetched in parallel using multiple threads.

Enable this option only if you have good network bandwidth. For example, in our measurements on an EC2 c5.2xlarge instance, a single-threaded S3 client reached 1 Gbps, while multiple-threaded S3 clients reached 4 Gbps of network throughput.

Connection string name	Parameter type	Default value	Connection string example
UseMultipleS3Threads	Optional	`0`	`UseMultipleS3Threads=1;`

Use single catalog and schema

By default, the ODBC driver queries Athena to get the list of available catalogs and schemas. This option forces the driver to use the catalog and schema specified by the ODBC Data Source Administrator configuration dialog box or connection parameters.

Connection string name	Parameter type	Default value	Connection string example
UseSingleCatalogAndSchema	Optional	`0`	`UseSingleCatalogAndSchema=1;`

Use query to list tables

For LAMBDA catalog types, enables the ODBC driver to submit a SHOW TABLES query to get a list of available tables. This setting is the default. If this parameter is set to 0, the ODBC driver uses the Athena ListTableMetadata API to get a list of available tables. Note that, for LAMBDA catalog types, using ListTableMetadata leads to performance regression.

Connection string name	Parameter type	Default value	Connection string example
UseQueryToListTables	Optional	1	`UseQueryToListTables=1;`

Use WCHAR for string types

By default, the ODBC driver uses SQL_CHAR and SQL_VARCHAR for Athena the string data types char, varchar, string, array, map<>, struct<>, and row. Setting this parameter to 1 forces the driver to use SQL_WCHAR and SQL_WVARCHAR for string data types. Wide character and wide variable character types are used to ensure that characters from different languages can be stored and retrieved correctly.

Connection string name	Parameter type	Default value	Connection string example
UseWCharForStringTypes	Optional	0	`UseWCharForStringTypes=1;`

Query external catalogs

Specifies if the driver needs to query external catalogs from Athena. For more information, see Migrate to the ODBC 2.x driver.

Connection string name	Parameter type	Default value	Connection string example
QueryExternalCatalogs	Optional	`0`	`QueryExternalCatalogs=1;`

Verify SSL

Controls whether to verify SSL certificates when you use the AWS SDK. This value is passed to `ClientConfiguration.verifySSL` parameter. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
VerifySSL	Optional	`1`	`VerifySSL=0;`

S3 result block size

Specifies, in bytes, the size of the block to download for a single Amazon S3 GetObject API request. The default value is 67108864 (64 MB). The minimum and maximum values allowed are 10485760 (10 MB) and 2146435072 (about 2 GB).

Connection string name	Parameter type	Default value	Connection string example
S3ResultBlockSize	Optional	`67108864`	`S3ResultBlockSize=268435456;`

String column length

Specifies the column length for columns with the `string` data type. Because Athena uses the Apache Hive string data type, which does not have defined precision, the default length reported by Athena is 2147483647 (`INT_MAX`). Because BI tools usually pre-allocate memory for columns, this can lead to high memory consumption. To avoid this, the Athena ODBC driver limits the reported precision for columns of the `string` data type and exposes the `StringColumnLength` connection parameter so that the default value can be changed.

Connection string name	Parameter type	Default value	Connection string example
StringColumnLength	Optional	255	`StringColumnLength =65535;`

Complex type column length

Specifies the column length for columns with complex data types like `map`, `struct`, and `array`. Like StringColumnLength, Athena reports 0 precision for columns with complex data types. The Athena ODBC driver sets the default precision for columns with complex data types and exposes the `ComplexTypeColumnLength` connection parameter so that the default value can be changed.

Connection string name	Parameter type	Default value	Connection string example
ComplexTypeColumnLength	Optional	65535	`ComplexTypeColumnLength=123456;`

Trusted CA certificate

Instructs the HTTP client where to find your SSL certificate trust store. This value is passed to the `ClientConfiguration.caFile` parameter. For more information, see AWS Client configuration in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
TrustedCerts	Optional	`%INSTALL_PATH%/bin`	`TrustedCerts=C:\\Program Files\\Amazon Athena ODBC Driver\\bin\\cacert.pem;`

Min poll period

Specifies the minimum value in milliseconds to wait before polling Athena for query execution status.

Connection string name	Parameter type	Default value	Connection string example
MinQueryExecutionPollingInterval	Optional	100	`MinQueryExecutionPollingInterval=200;`

Max poll period

Specifies the maximum value in milliseconds to wait before polling Athena for the query execution status.

Connection string name	Parameter type	Default value	Connection string example
MaxQueryExecutionPollingInterval	Optional	60000	`MaxQueryExecutionPollingInterval=1000;`

Poll multiplier

Specifies the factor for increasing the poll period. By default, polling begins with the value of min poll period and doubles with each poll until it reaches the value of max poll period.

Connection string name	Parameter type	Default value	Connection string example
QueryExecutionPollingIntervalMultiplier	Optional	2	`QueryExecutionPollingIntervalMultiplier=2;`

Max poll duration

Specifies the maximum value in milliseconds that a driver can poll Athena for query execution status.

Connection string name	Parameter type	Default value	Connection string example
MaxPollDuration	Optional	`1800000`	`MaxPollDuration=1800000;`

Connection timeout

The amount of time (in milliseconds) that the HTTP connection waits to establish a connection. This value is set for `ClientConfiguration.connectTimeoutMs` Athena client. If not specified, the curl default value is used. For information about connection parameters, see [Client Configuration] in the *AWS SDK for Java Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
ConnectionTimeout	Optional	`0`	`ConnectionTimeout=2000;`

Request timeout

Specifies the socket read timeout for HTTP clients. This value is set for the `ClientConfiguration.requestTimeoutMs` parameter of the Athena client. For parameter information, see [Client Configuration] in the *AWS SDK for Java Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
RequestTimeout	Optional	`10000`	`RequestTimeout=30000;`

Proxy options

Proxy host

If you require users to go through a proxy, use this parameter to set the proxy host. This parameter corresponds to the `ClientConfiguration.proxyHost` parameter in the AWS SDK. For more information, see [AWS Client configuration](#) in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
ProxyHost	Optional	none	`ProxyHost =127.0.0.1;`

Proxy port

Use this parameter to set the proxy port. This parameter corresponds to the `ClientConfiguration.proxyPort` parameter in the AWS SDK. For more information, see [AWS Client configuration](#) in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
ProxyPort	Optional	none	`ProxyPort=8888;`

Proxy user name

Use this parameter to set the proxy user name. This parameter corresponds to the `ClientConfiguration.proxyUserName` parameter in the AWS SDK. For more information, see [AWS Client configuration](#) in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
ProxyUID	Optional	none	`ProxyUID= username;`

Proxy password

Use this parameter to set the proxy password. This parameter corresponds to the `ClientConfiguration.proxyPassword` parameter in the AWS SDK. For more information, see [AWS Client configuration](#) in the *AWS SDK for C++ Developer Guide*.

Connection string name	Parameter type	Default value	Connection string example
ProxyPWD	Optional	none	`ProxyPWD=password;`

Non proxy host

Use this optional parameter to specify a host that the driver connects to without using a proxy. This parameter corresponds to the `ClientConfiguration.nonProxyHosts` parameter in the AWS SDK. For more information, see [AWS Client configuration](#) in the *AWS SDK for C++ Developer Guide*.

The `NonProxyHost` connection parameter is passed to the `CURLOPT_NOPROXY` curl option. For information about the `CURLOPT_NOPROXY` format, see [CURLOPT_NOPROXY](#) in the curl documentation.

Connection string name	Parameter type	Default value	Connection string example
NonProxyHost	Optional	none	`NonProxyHost=.amazonaws.com,localhost,.example.net,.example.com;`

Use proxy

Enables user traffic through the specified proxy.

Connection string name	Parameter type	Default value	Connection string example
UseProxy	Optional	none	`UseProxy=1;`

Logging options

Administrator rights are required to modify the settings described here. To make the changes, you can use the ODBC Data Source Administrator **Logging Options** dialog box or modify the Windows registry directly.

Log level

This option enables ODBC driver logs. In Windows, you can use the registry or a dialog box to enable or disable logging. The option is located in the following registry path:

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

Connection string name	Parameter type	Default value	Connection string example
LogLevel	Optional	`0`	`LogLevel=1;`

Log path

Specifies path to the file where the ODBC driver logs are stored. You can use the registry or a dialog box to set this value. The option is located in the following registry path:

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

Connection string name	Parameter type	Default value	Connection string example
LogPath	Optional	none	`LogPath=C:\Users\ username\projects\internal\trunk\;`

Use AWS Logger

Specifies if AWS SDK logging is enabled. Specify 1 to enable, 0 to disable.

Connection string name	Parameter type	Default value	Connection string example
UseAwsLogger	Optional	`0`	`UseAwsLogger=1;`

Migrate to the ODBC 2.x driver

Because most Athena ODBC 2.x connection parameters are backwardly compatible with the ODBC 1.x driver, you can reuse most of your existing connection string with the Athena ODBC 2.x driver. However, the following connection parameters require modifications.

Log level

While the current ODBC driver provides a range of available logging options, starting from `LOG_OFF (0)` to `LOG_TRACE (6)`, the Amazon Athena ODBC driver has only two values: 0 (disabled) and 1 (enabled).

For more information about logging the ODBC 2.x driver, see [Logging options](#).

	ODBC 1.x driver	ODBC 2.x driver
Connection string name	`LogLevel`	`LogLevel`
Parameter type	Optional	Optional
Default value	`0`	`0`
Possible values	`0-6`	`0,1`
Connection string example	`LogLevel=6;`	`LogLevel=1;`

MetadataRetrievalMethod

The current ODBC driver provides several options for retrieving the metadata from Athena. The Amazon Athena ODBC driver deprecates the `MetadataRetrievalMethod` and always uses the Amazon Athena API to extract metadata.

Athena introduces the flag `QueryExternalCatalogs` for querying external catalogs. To query external catalogs with the current ODBC driver, set `MetadataRetrievalMethod` to `ProxyAPI`. To query external catalogs with the Athena ODBC driver, set `QueryExternalCatalogs` to 1.

	ODBC 1.x driver	ODBC 2.x driver
Connection string name	`MetadataRetrievalMethod`	`QueryExternalCatalogs`
Parameter type	Optional	Optional
Default value	`Auto`	`0`
Possible values	`Auto`, `AWS Glue`, `ProxyAPI`, `Query`	`0`,`1`
Connection string example	`MetadataRetrievalMethod=ProxyAPI;`	`QueryExternalCatalogs=1;`

Connection test

When you test an ODBC 1.x driver connection, the driver runs a `SELECT 1` query that generates two files in your Amazon S3 bucket: one for the result set, and one for the metadata. The test connection is charged according to the [Amazon Athena Pricing] policy.

When you test an ODBC 2.x driver connection, the driver calls the [GetWorkGroup] Athena API action. The call uses the authentication type and corresponding credentials provider that you specified to retrieve the credentials. There is no charge for the connection test when you use the ODBC 2.x driver, and the test does not generate query results in your Amazon S3 bucket.

Troubleshoot the ODBC 2.x driver

If you encounter issues with the Amazon Athena ODBC driver, you can contact Support (in the AWS Management Console, choose **Support**, **Support Center**).

Be sure to include the following information, and provide any additional details that will help the support team understand your use case.

- **Description** – (Required) A description that includes detailed information about your use case and the difference between the expected and observed behavior. Include any information that can help support engineers navigate the issue easily. If the issue is intermittent, specify the dates, timestamps, or interval points at which the issue occurred.

- **Version information –** (Required) Information about the driver version, the operating system, and the applications that you used. For example, "ODBC driver version 1.2.3, Windows 10 (x64), Power BI."
- **Log files –** (Required) The minimum number of ODBC driver log files that are required to understand the issue. For information about logging options for the ODBC 2.x driver, see Logging options.
- **Connection string –** (Required) Your ODBC connection string or a screen shot of the dialog box that shows the connection parameters that you used. For information about connection parameters, see Athena ODBC 2.x connection parameters.
- **Issue steps –** (Optional) If possible, include steps or a standalone program that can help reproduce the issue.
- **Query error information –** (Optional) If you have errors that involve DML or DDL queries, include the following information:
  - A full or simplified version of the failed DML or DDL query.
  - The account ID and AWS Region used, and the query execution ID.
- **SAML errors –** (Optional) If you have an issue related to authentication with SAML assertion, include the following information:
  - The identity provider and authentication plugin that was used.
  - An example with the SAML token.

Amazon Athena ODBC 2.x release notes

These release notes provide details of enhancements, features, known issues, and workflow changes in the Amazon Athena ODBC 2.x driver.

2.0.4.0

Released 2025-06-17

The Amazon Athena ODBC v2.0.4.0 driver contains the following improvements and fixes.

Improvements

- **Result Fetcher** – The driver now automatically selects the method to download query results. This removes the need to manually configure the fetcher in most situations. For more information, see ???.
- Curl Library has been updated to 8.12.1.

Fixes

- Fixed proxy configuration for IAM profile when connecting to STS. The fix allows IAM Profile to be used for successful authentication.
- Read all additional configuration options for IAM profile with authentication plugins. This includes `UseProxyForIdP`, `SSL_Insecure`, `LakeformationEnabled`, and `LoginToRP` to resolve misconfigurations for the affected plugins.
- Fixed round function by allowing it to take in an optional 2nd parameter. It successfully processes queries containing the escape syntax.
- Fixed column size for `TIME WITH TIME ZONE` and `TIMESTAMP WITH TIME ZONE` data types. Values with timestamp and timezone data type will not get truncated.

To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see Amazon Athena ODBC 2.x.

2.0.3.0

Released 2024-04-08

The Amazon Athena ODBC v2.0.3.0 driver contains the following improvements and fixes.

Improvements

- Added MFA support for the Okta authentication plugin on Linux and Mac platforms.
- Both the `athena-odbc.dll` library and the `AmazonAthenaODBC-2.x.x.x.msi` installer for Windows are now signed.
- Updated the CA certificate `cacert.pem` file that is installed with the driver.
- Improved the time required to list tables under Lambda catalogs. For `LAMBDA` catalog types, the ODBC driver can now submit a SHOW TABLES query to get a list of available tables. For more information, see Use query to list tables.

- Introduced the `UseWCharForStringTypes` connection parameter to report string data types using `SQL_WCHAR` and `SQL_WVARCHAR`. For more information, see Use WCHAR for string types.

Fixes

- Fixed a registry corruption warning that occurred when the Get-OdbcDsn PowerShell tool was used.
- Updated the parsing logic to handle comments at the start of query strings.
- Date and timestamp data types now allow zero in the year field.

To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see Amazon Athena ODBC 2.x.

2.0.2.2

Released 2024-02-13

The Amazon Athena ODBC v2.0.2.2 driver contains the following improvements and fixes.

Improvements

- Added two connection parameters, `StringColumnLength` and `ComplexTypeColumnLength`, that you can use to change the default column length for string and complex data types. For more information, see String column length and Complex type column length.
- Support has been added for the Linux and macOS (Intel and ARM) operating systems. For more information, see Linux and macOS.
- AWS-SDK-CPP has been updated to the 1.11.245 tag version.
- The curl library has been updated to the 8.6.0 version.

Fixes

- Resolved an issue that cause incorrect values to be reported in result-set metadata for string-like data types in the precision column.

To download the ODBC v2 driver, see ODBC 2.x driver download. For connection information, see Amazon Athena ODBC 2.x.

2.0.2.1

Released 2023-12-07

The Amazon Athena ODBC v2.0.2.1 driver contains the following improvements and fixes.

Improvements

- Improved ODBC driver thread safety for all interfaces.
- When logging is enabled, datetime values are now recorded with millisecond precision.
- During authentication with the Browser SSO OIDC plugin, the terminal now opens to display the device code to the user.

Fixes

- Resolved a memory release issue that occurred when parsing results from the streaming API.
- Requests for the interfaces `SQLTablePrivileges()`, `SQLSpecialColumns()`, `SQLProcedureColumns()`, and `SQLProcedures()` now return empty result sets.

To download the ODBC v2 driver, see ODBC 2.x driver download. For connection information, see Amazon Athena ODBC 2.x.

2.0.2.0

Released 2023-10-17

The Amazon Athena ODBC v2.0.2.0 driver contains the following improvements and fixes.

Improvements

- File cache feature added for the Browser Azure AD, Browser SSO OIDC, and Okta browser-based authentication plugins.

  BI Tools like Power BI and browser-based plugins use multiple browser windows. The new file cache connection parameter enables temporary credentials to be cached and reused between the multiple processes opened by BI applications.
- Applications can now query information about the result set after a statement is prepared.
- Default connection and request timeouts have been increased for use with slower client networks. For more information, see Connection timeout and Request timeout.

- Endpoint overrides have been added for SSO and SSO OIDC. For more information, see Endpoint overrides.
- Added a connection parameter to pass a URI argument for an authentication request to Ping. You can use this parameter to bypass the Lake Formation single role limitation. For more information, see Ping URI param.

Fixes

- Fixed an integer overflow issue that occurred when using the row-based binding mechanism.
- Removed timeout from the list of required connection parameters for the Browser SSO OIDC authentication plugin.
- Added missing interfaces for `SQLStatistics()`, `SQLPrimaryKeys()`, `SQLForeignKeys()`, and `SQLColumnPrivileges()`, and added the ability to return empty result sets upon request.

To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see Amazon Athena ODBC 2.x.

2.0.1.1

Released 2023-08-10

The Amazon Athena ODBC v2.0.1.1 driver contains the following improvements and fixes.

Improvements

- Added URI logging to the Okta authentication plugin.
- Added the preferred role parameter to the external credentials provider plugin.
- Adding handling for the profile prefix in the profile name of AWS configuration file.

Fixes

- Corrected a AWS Region use issue that occurred when working with Lake Formation and AWS STS clients.
- Restored missing partition keys to the list of table columns.
- Added the missing `BrowserSSOOIDC` authentication type to the AWS profile.

To download the new ODBC v2 driver, see ODBC 2.x driver download.

2.0.1.0

Released 2023-06-29

Amazon Athena releases the ODBC v2.0.1.0 driver.

Athena has released a new ODBC driver that improves the experience of connecting to, querying, and visualizing data from compatible SQL development and business intelligence applications. The latest version of the Athena ODBC driver supports the features of the existing driver and is straightforward to upgrade. The new version includes support for authenticating users through AWS IAM Identity Center. It also offers the option to read query results from Amazon S3, which can make query results available to you sooner.

For more information, see Amazon Athena ODBC 2.x.

Athena ODBC 1.x driver

You can use an ODBC connection to connect to Athena from third-party SQL client tools and applications. Use the links on this page to download the Amazon Athena 1.x ODBC driver License Agreement, ODBC drivers, and ODBC documentation. For information about the ODBC connection string, see the ODBC Driver Installation and Configuration Guide PDF file, downloadable from this page. For permissions information, see Control access through JDBC and ODBC connections.

Important

When you use the ODBC 1.x driver, be sure to note the following requirements:

- **Open port 444** – Keep port 444, which Athena uses to stream query results, open to outbound traffic. When you use a PrivateLink endpoint to connect to Athena, ensure that the security group attached to the PrivateLink endpoint is open to inbound traffic on port 444.
- **athena:GetQueryResultsStream policy** – Add the `athena:GetQueryResultsStream` policy action to the IAM principals that use the ODBC driver. This policy action is not exposed directly with the API. It is used only with the ODBC and JDBC drivers as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess.

Windows

Driver version	Download link
ODBC 1.2.3.1000 for Windows 32-bit	Windows 32 bit ODBC driver 1.2.3.1000
ODBC 1.2.3.1000 for Windows 64-bit	Windows 64 bit ODBC driver 1.2.3.1000

Linux

Driver version	Download link
ODBC 1.2.3.1000 for Linux 32-bit	Linux 32 bit ODBC driver 1.2.3.1000
ODBC 1.2.3.1000 for Linux 64-bit	Linux 64 bit ODBC driver 1.2.3.1000

OSX

Driver version	Download link
ODBC 1.2.3.1000 for OSX	OSX ODBC driver 1.2.3.1000

Documentation

Content	Documentation link
Amazon Athena ODBC driver license agreement	License agreement
Documentation for ODBC 1.2.3.1000	ODBC driver installation and configuration guide version 1.2.3.1000
Release Notes for ODBC 1.2.3.1000	ODBC driver release notes version 1.2.3.1000

ODBC driver notes

Connecting Without Using a Proxy

If you want to specify certain hosts that the driver connects to without using a proxy, you can use the optional `NonProxyHost` property in your ODBC connection string.

The `NonProxyHost` property specifies a comma-separated list of hosts that the connector can access without going through the proxy server when a proxy connection is enabled, as in the following example:

```
.amazonaws.com,localhost,.example.net,.example.com
```

The `NonProxyHost` connection parameter is passed to the `CURLOPT_NOPROXY` curl option. For information about the `CURLOPT_NOPROXY` format, see CURLOPT_NOPROXY in the curl documentation.

Configure federated access to Amazon Athena for Microsoft AD FS users using an ODBC client

To set up federated access to Amazon Athena for Microsoft Active Directory Federation Services (AD FS) users using an ODBC client, you first establish trust between AD FS and your AWS account. With this trust in place, your AD users can federate into AWS using their AD credentials and assume permissions of an AWS Identity and Access Management (IAM) role to access AWS resources such as the Athena API.

To create this trust, you add AD FS as a SAML provider to your AWS account and create an IAM role that federated users can assume. On the AD FS side, you add AWS as a relying party and write SAML claim rules to send the right user attributes to AWS for authorization (specifically, Athena and Amazon S3).

Configuring AD FS access to Athena involves the following major steps:

1. Setting up an IAM SAML provider and role

2. Configuring AD FS

3. Creating Active Directory users and groups

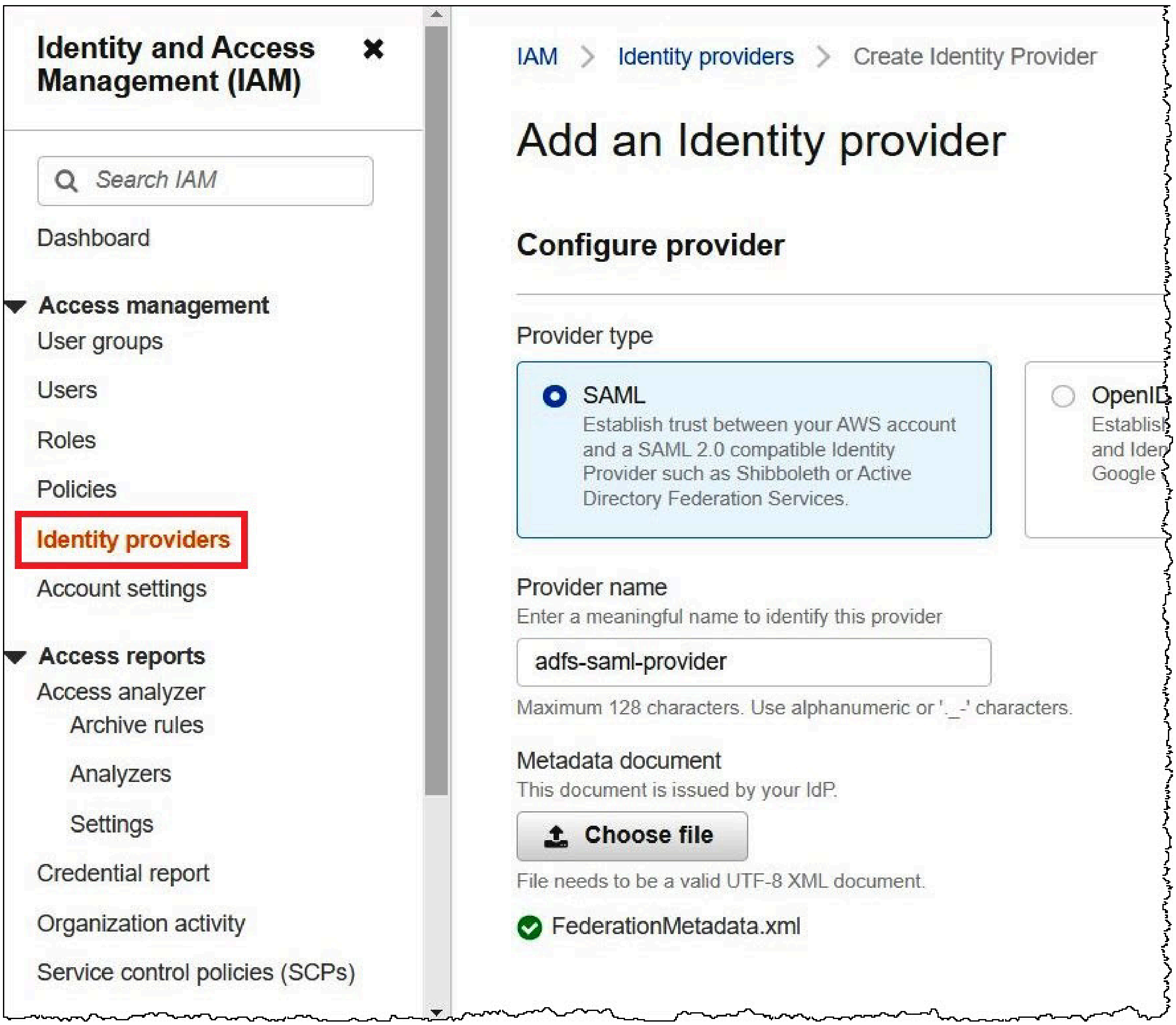
4. Configuring the AD FS ODBC connection to Athena

1. Setting up an IAM SAML provider and role

In this section, you add AD FS as a SAML provider to your AWS account and create an IAM role that your federated users can assume.

To set up a SAML provider

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**.
3. Choose **Add provider**.
4. For **Provider type**, choose **SAML**.



5. For **Provider name**, enter **`adfs-saml-provider`**.
6. In a browser, enter the following address to download the federation XML file for your AD FS server. To perform this step, your browser must have access to the AD FS server.

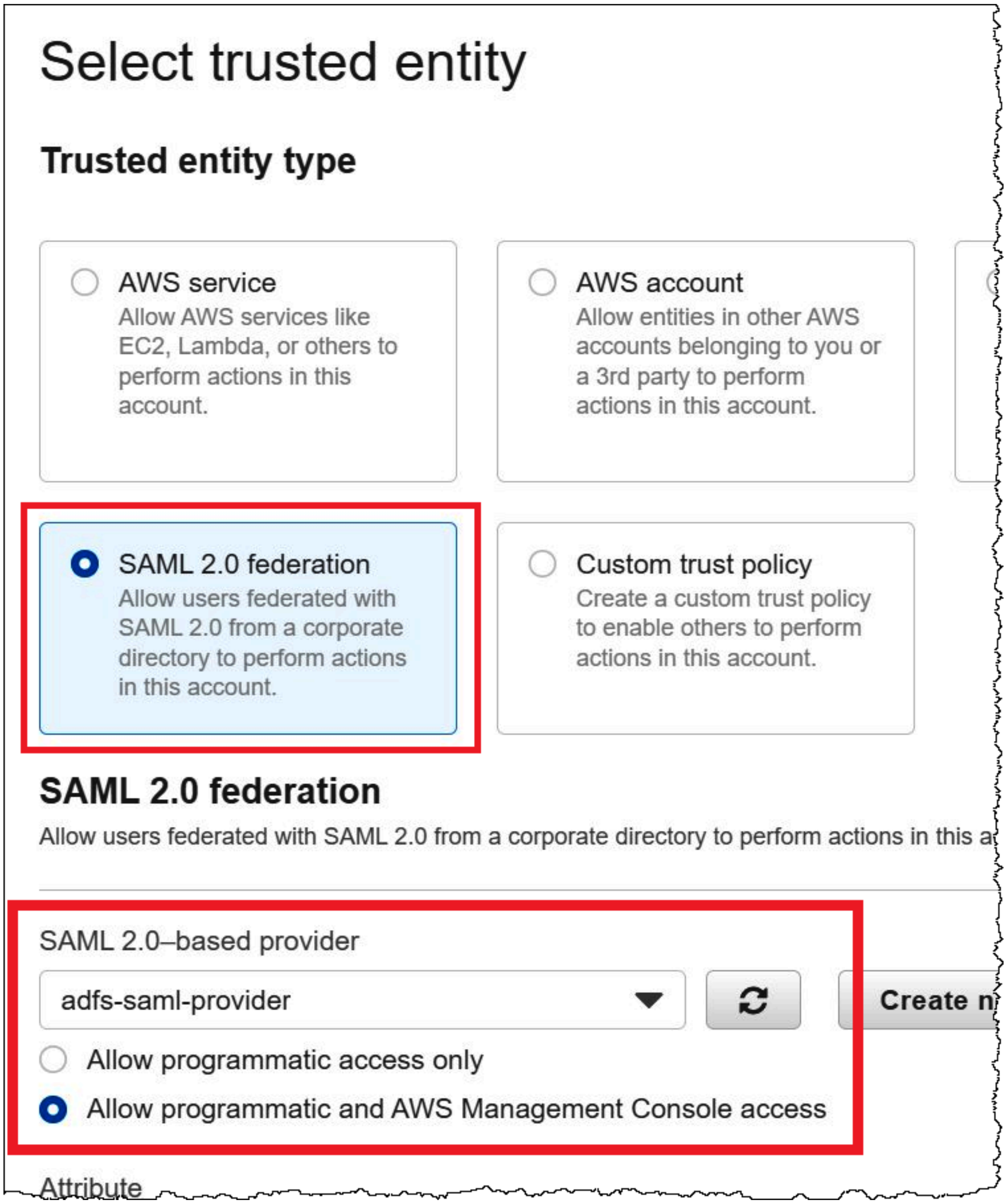
```
https://adfs-server-name/federationmetadata/2007-06/federationmetadata.xml
```

7. In the IAM console, for **Metadata document**, choose **Choose file**, and then upload the federation metadata file to AWS.
8. To finish, choose **Add provider**.

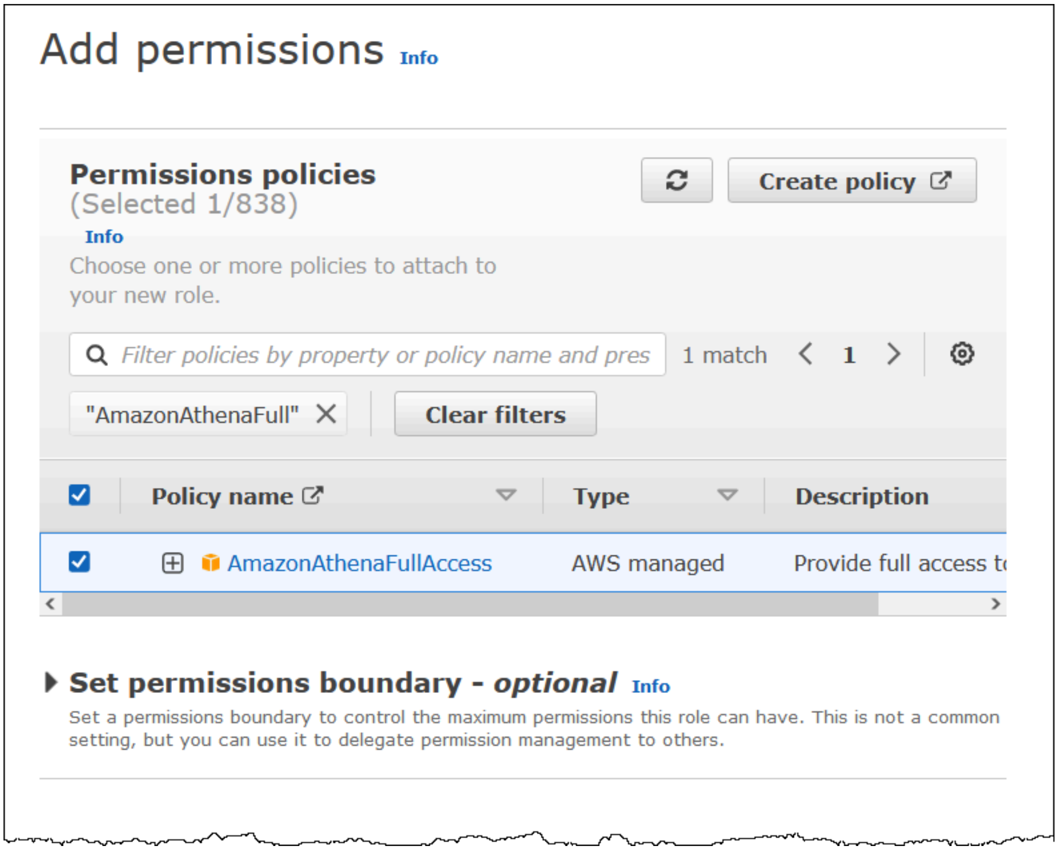
Next, you create the IAM role that your federated users can assume.

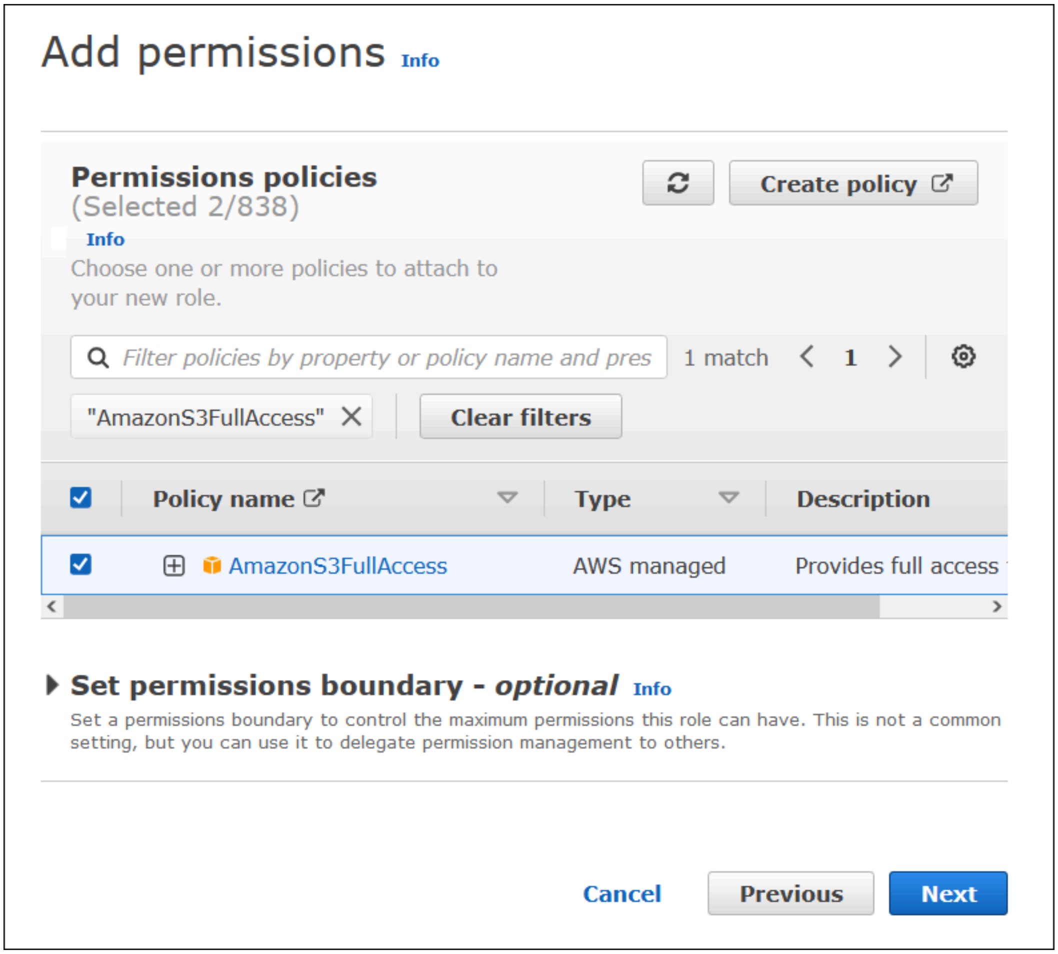
To create an IAM role for federated users

1. In the IAM console navigation pane, choose **Roles**.
2. Choose **Create role**.
3. For **Trusted entity type**, choose **SAML 2.0 federation**.
4. For **SAML 2.0-based provider**, choose the **adfs-saml-provider** provider that you created.
5. Choose **Allow programmatic and AWS Management Console access**, and then choose **Next**.



6. On the **Add permissions** page, filter for the IAM permissions policies that you require for this role, and then select the corresponding check boxes. This tutorial attaches the `AmazonAthenaFullAccess` and `AmazonS3FullAccess` policies.


Add permissions Info
Permissions policies
(Selected 1/838)
Info
Choose one or more policies to attach to your new role.
Create policy
Filter policies by property or policy name and pres
1 match
1
"AmazonAthenaFull"
Clear filters
Policy name
Type
Description
AmazonAthenaFullAccess
AWS managed
Provide full access t
Set permissions boundary - optional Info
Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.



7. Choose **Next**.

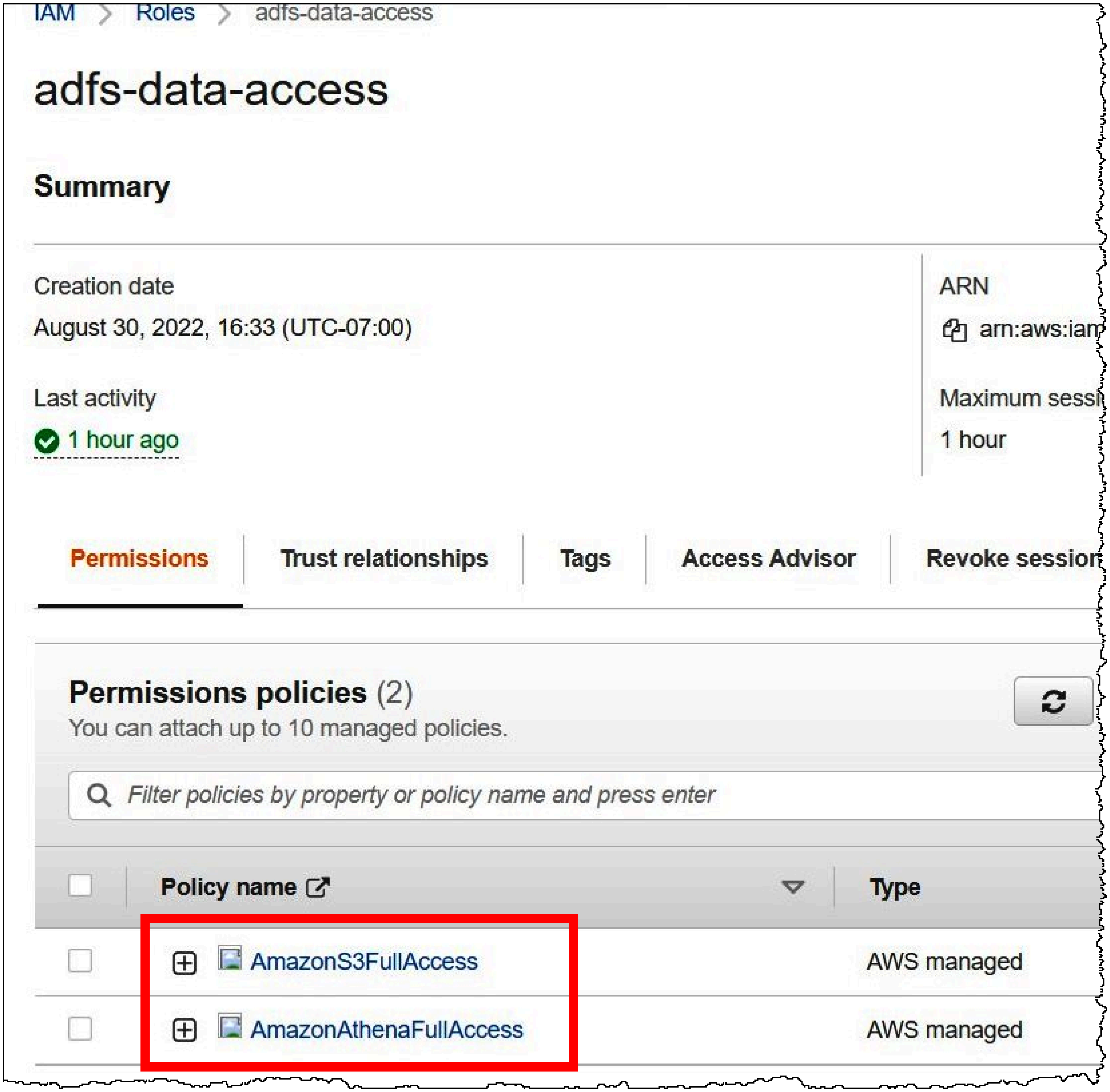
8. On the **Name, review, and create** page, for **Role name**, enter a name for the role. This tutorial uses the name **adfs-data-access**.

   In **Step 1: Select trusted entities**, the **Principal** field should be automatically populated with `"Federated:" "arn:aws:iam::account_id:saml-provider/adfs-saml-provider"`. The `Condition` field should contain `"SAML:aud"` and `"https://signin.aws.amazon.com/saml"`.

Step 2: Add permissions shows the policies that you have attached to the role.

9. Choose **Create role**. A banner message confirms creation of the role.

10. On the **Roles** page, choose the name of the role that you just created. The summary page for the role shows the policies that have been attached.



2. Configuring AD FS

Now you are ready to add AWS as a relying party and write SAML claim rules so that you can send the right user attributes to AWS for authorization.

SAML-based federation has two participant parties: the IdP (Active Directory) and the relying party (AWS), which is the service or application that uses authentication from the IdP.

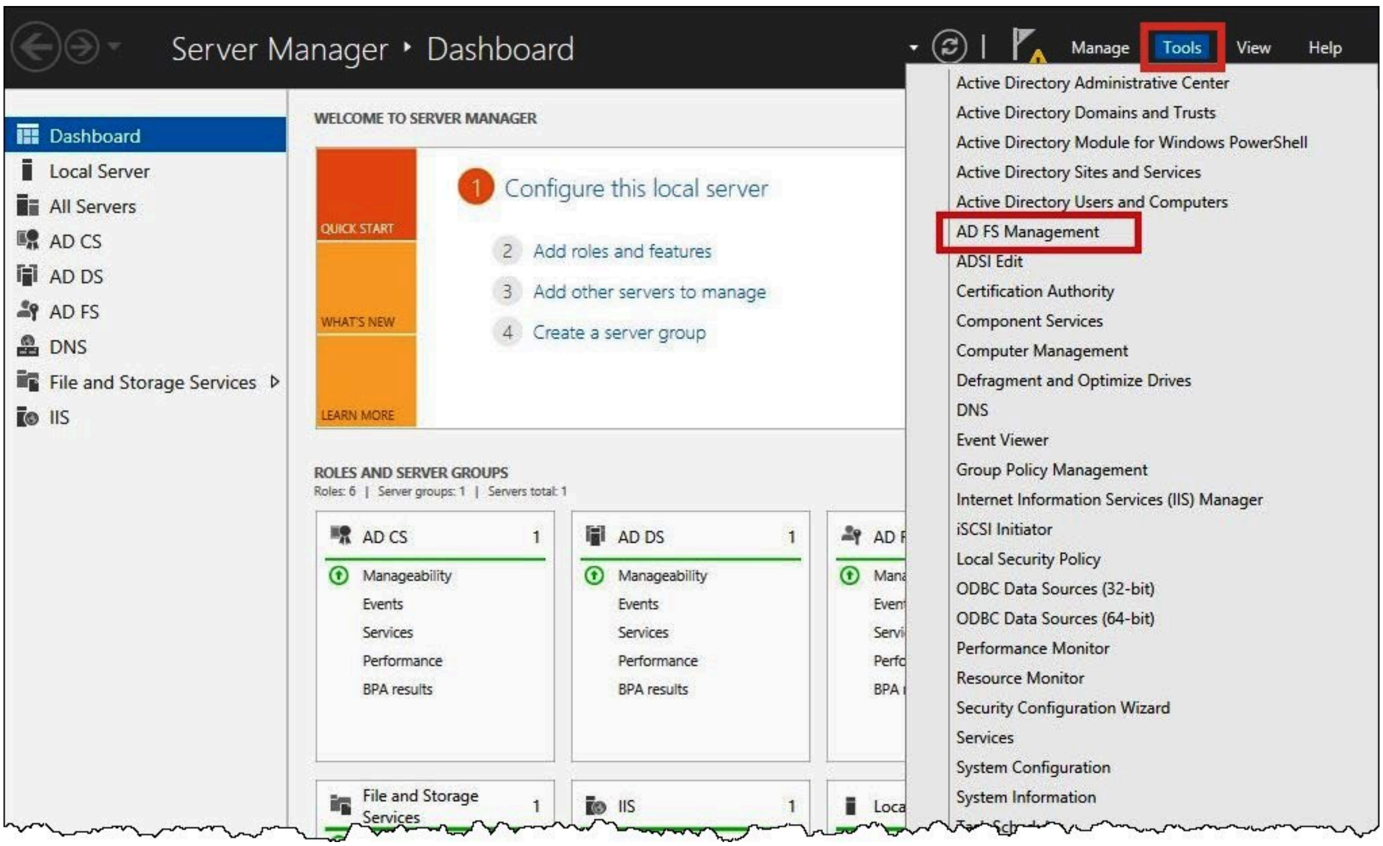
To configure AD FS, you first add a relying party trust, then you configure SAML claim rules for the relying party. AD FS uses claim rules to form a SAML assertion that is sent to a relying party. The SAML assertion states that the information about the AD user is true, and that it has authenticated the user.

Adding a relying party trust

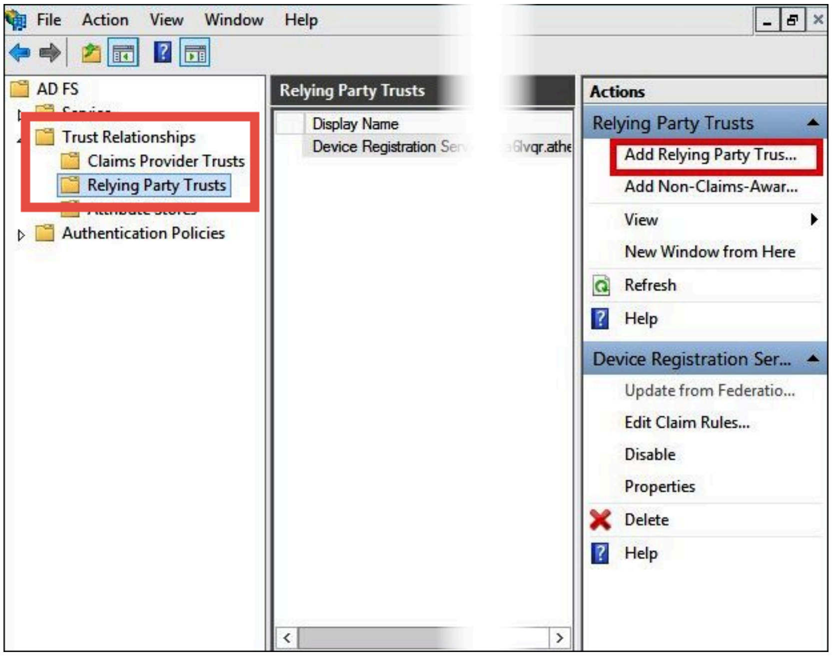
To add a relying party trust in AD FS, you use the AD FS server manager.

To add a relying party trust in AD FS

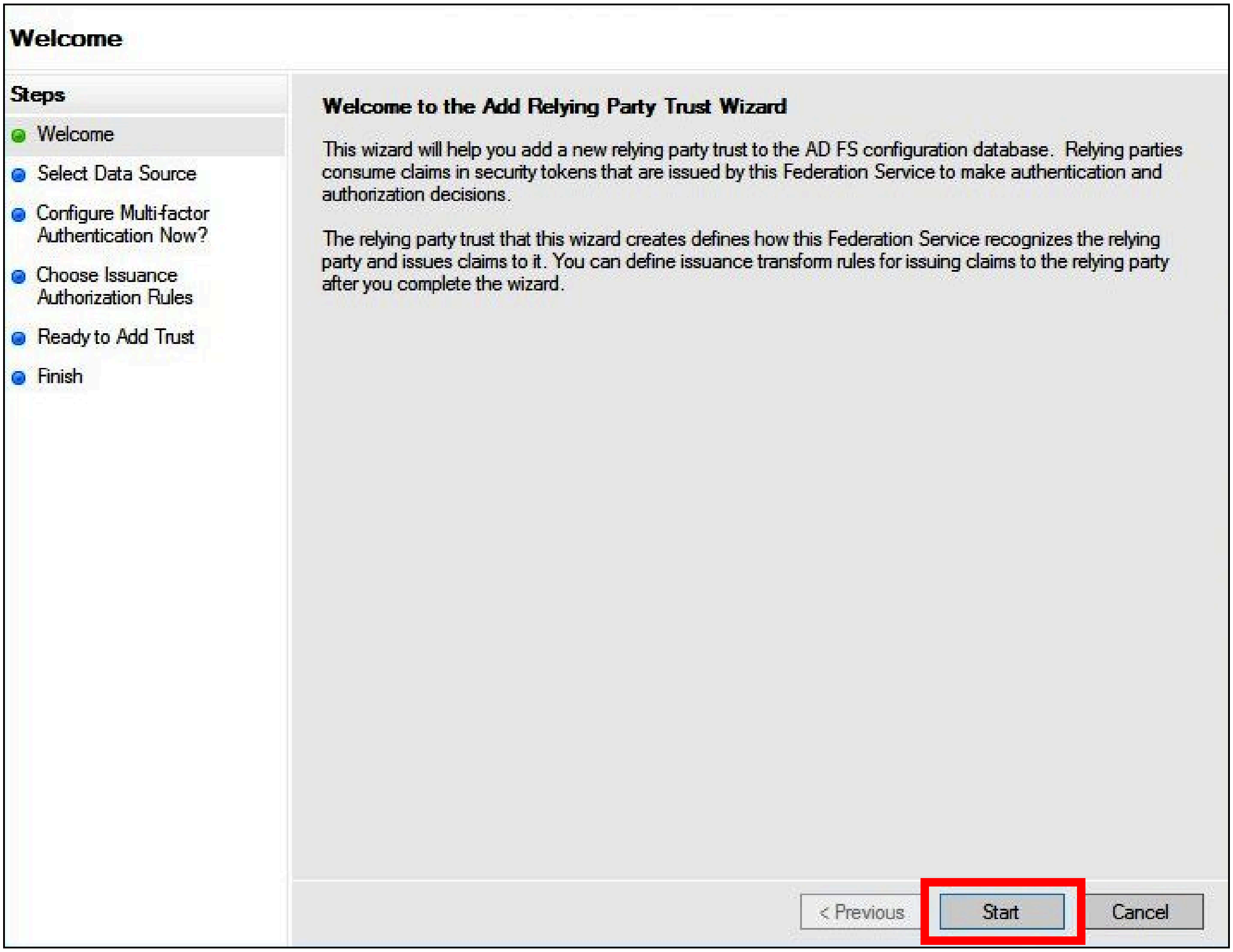
1. Sign in to the AD FS server.
2. On the **Start** menu, open **Server Manager**.
3. Choose **Tools**, and then choose **AD FS Management**.



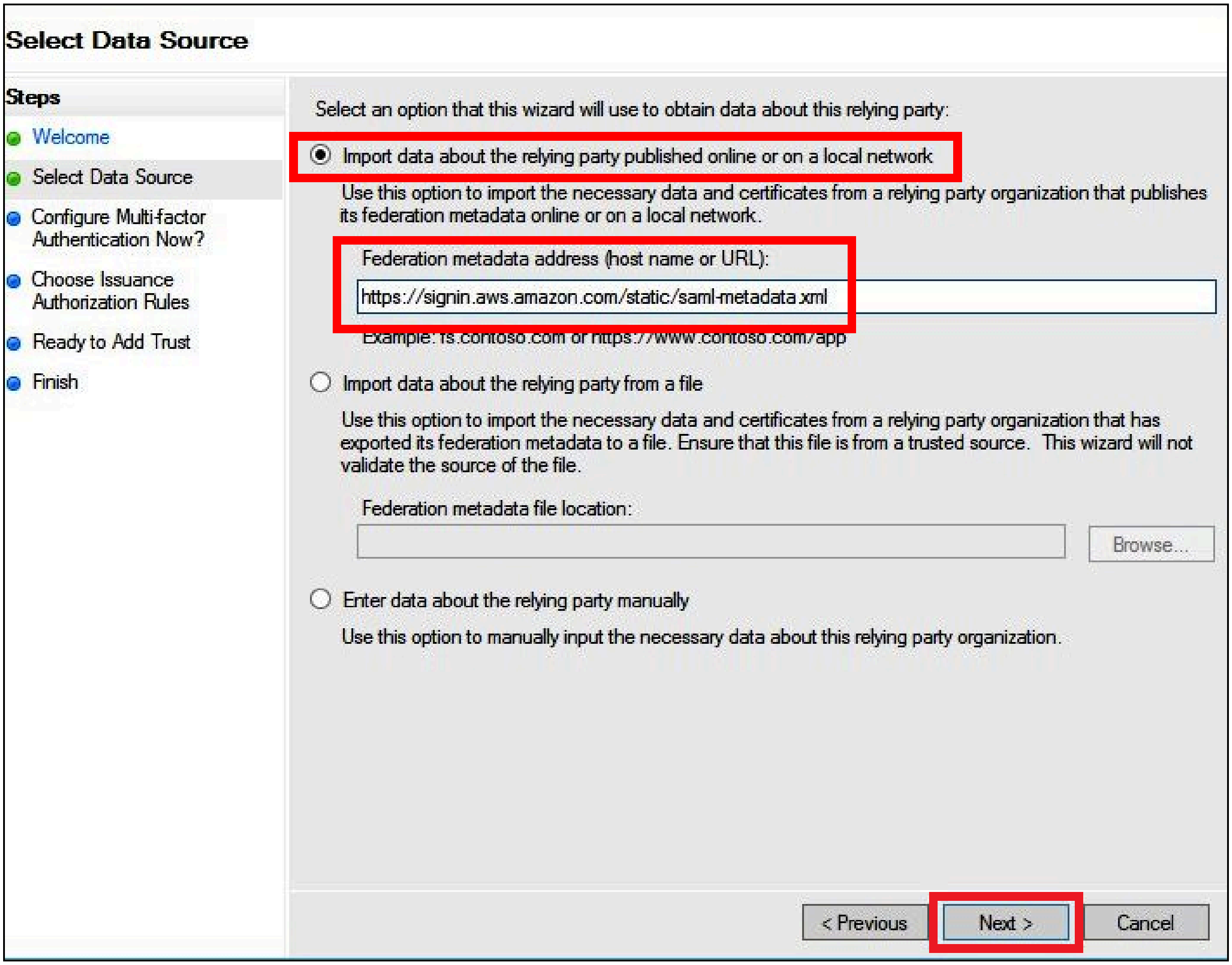
4. In the navigation pane, under **Trust Relationships**, choose **Relying Party Trusts**.
5. Under **Actions**, choose **Add Relying Party Trust**.



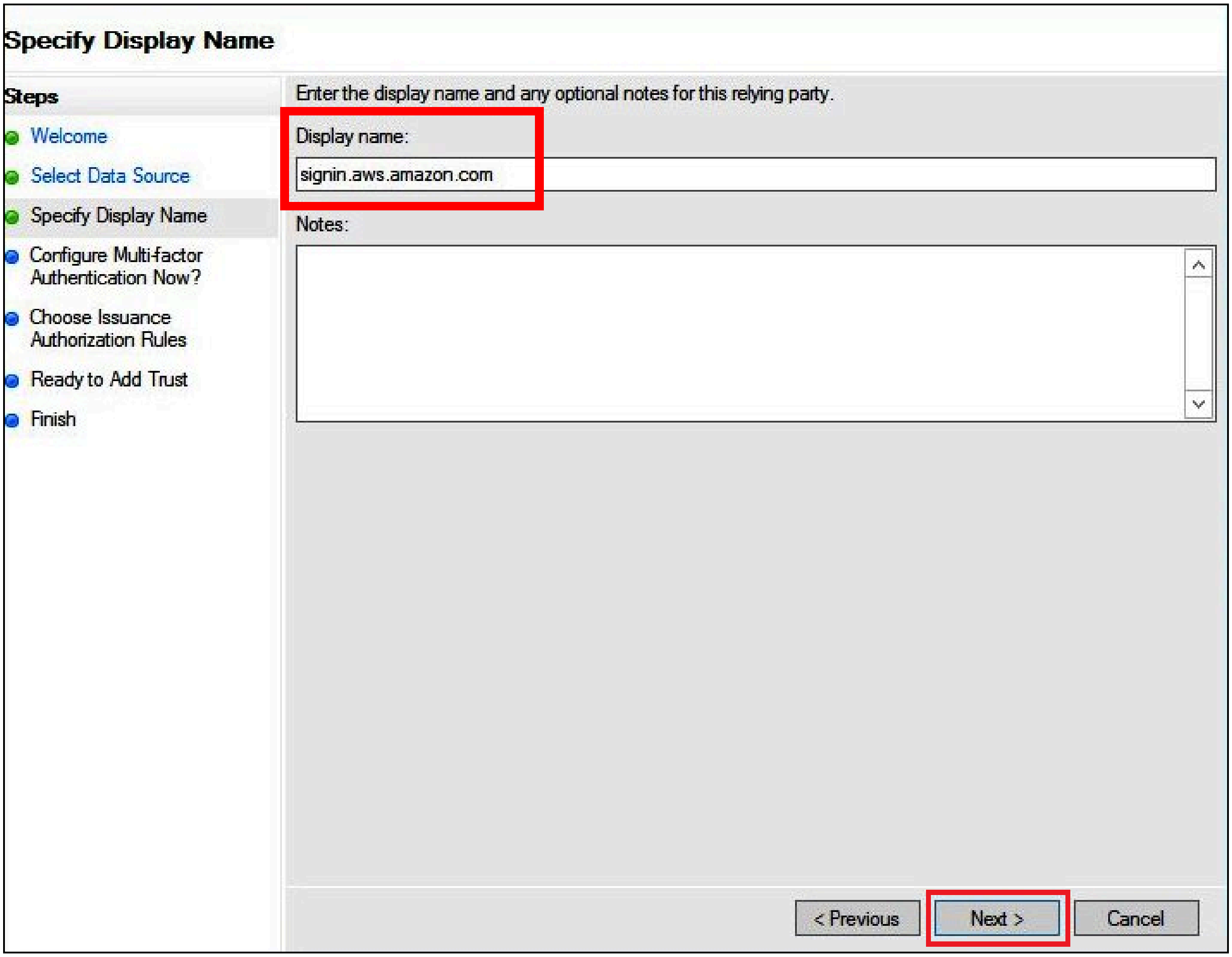
6. On the **Add Relying Party Trust Wizard** page, choose **Start**.



7. On the **Select Data Source** screen, select the option **Import data about the relying party published online or on a local network**.
8. For **Federation metadata address (host name or URL)**, enter the URL **`https://signin.aws.amazon.com/static/saml-metadata.xml`**
9. Choose **Next.**

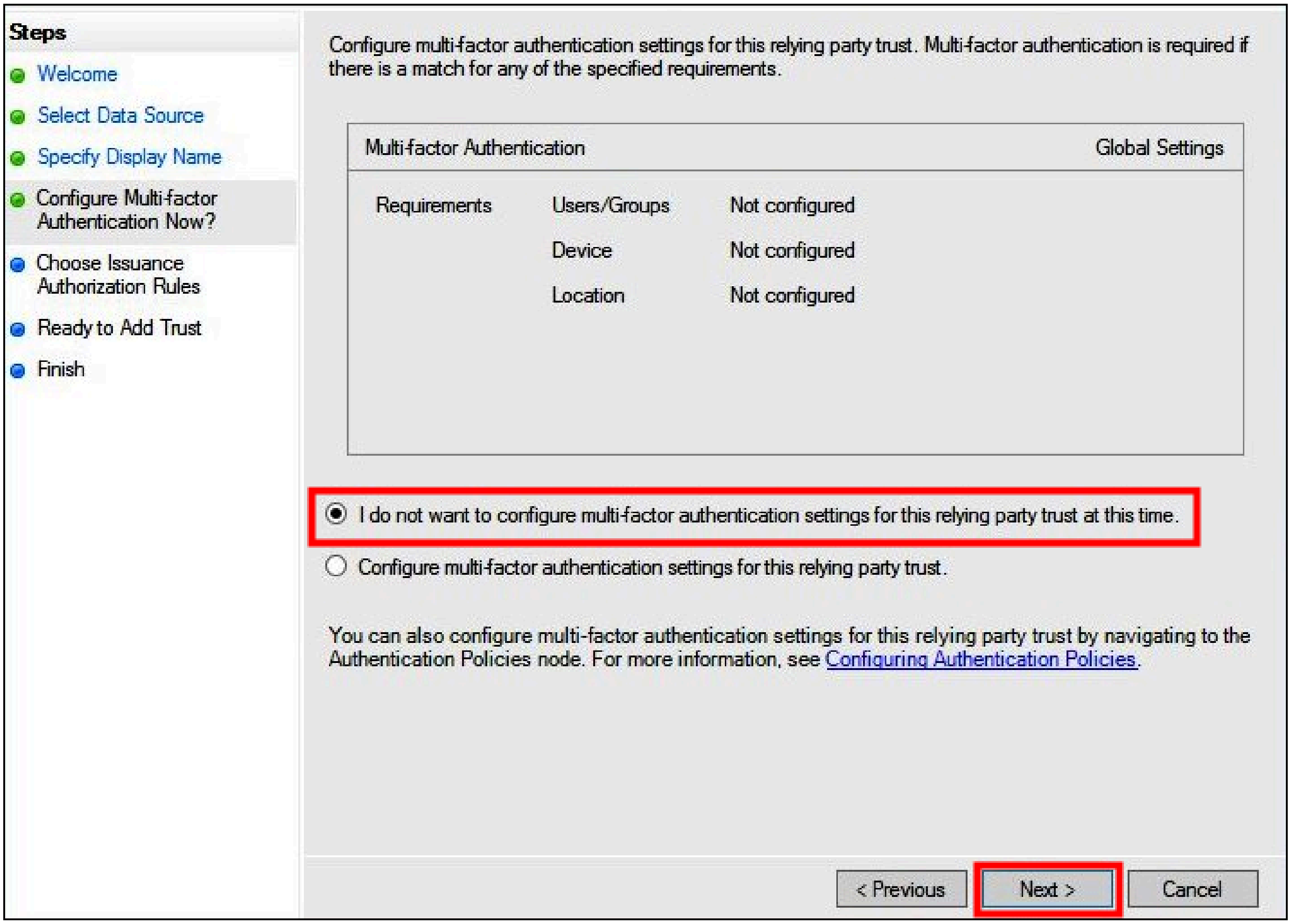


10. On the **Specify Display Name** page, for **Display name**, enter a display name for your relying party, and then choose **Next**.



11. On the **Configure Multi-factor Authentication Now** page, this tutorial selects **I do not want to configure multi-factor authentication for this relying party trust at this time**.

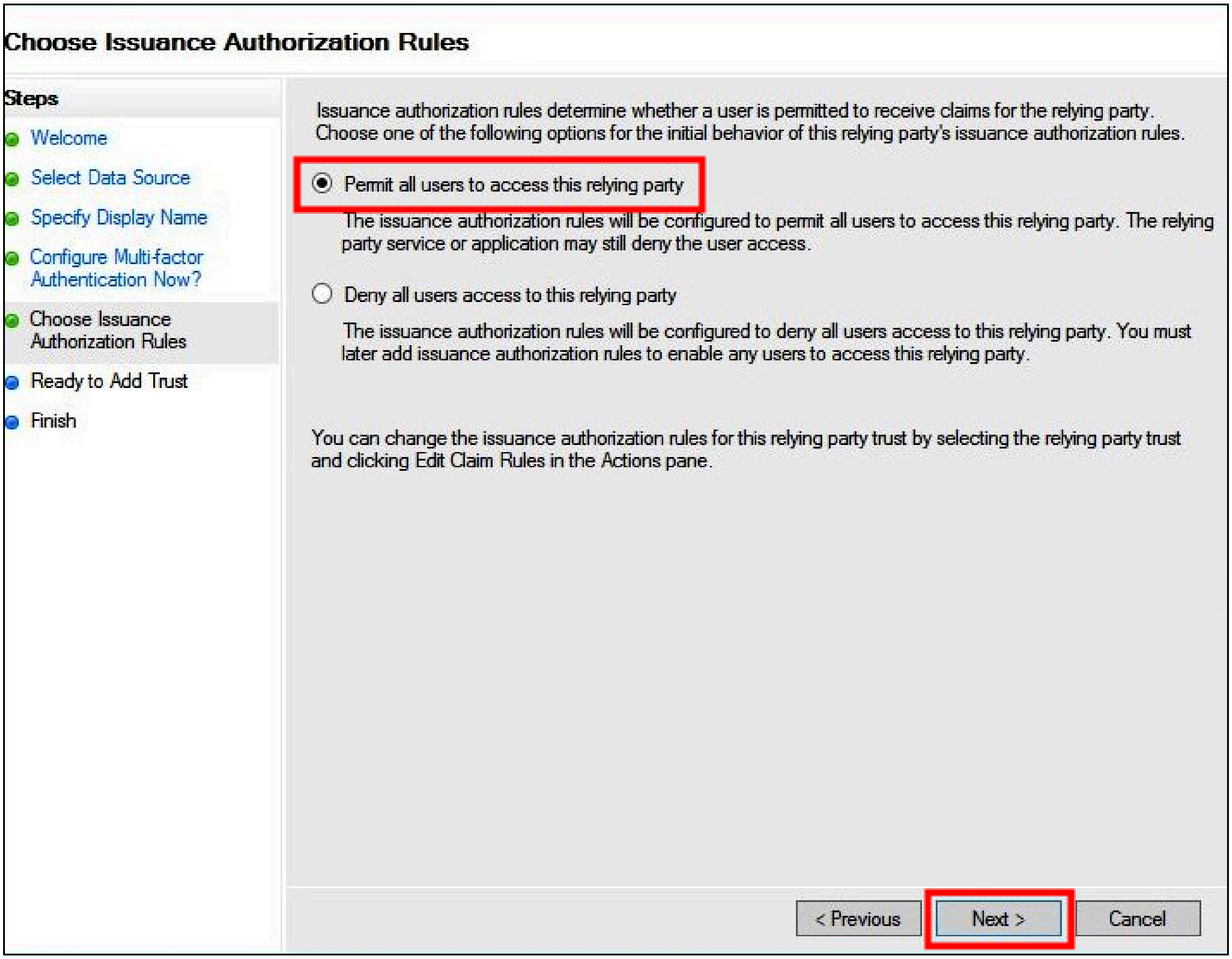
    For increased security, we recommend that you configure multi-factor authentication to help protect your AWS resources. Because it uses a sample dataset, this tutorial doesn't enable multi-factor authentication.



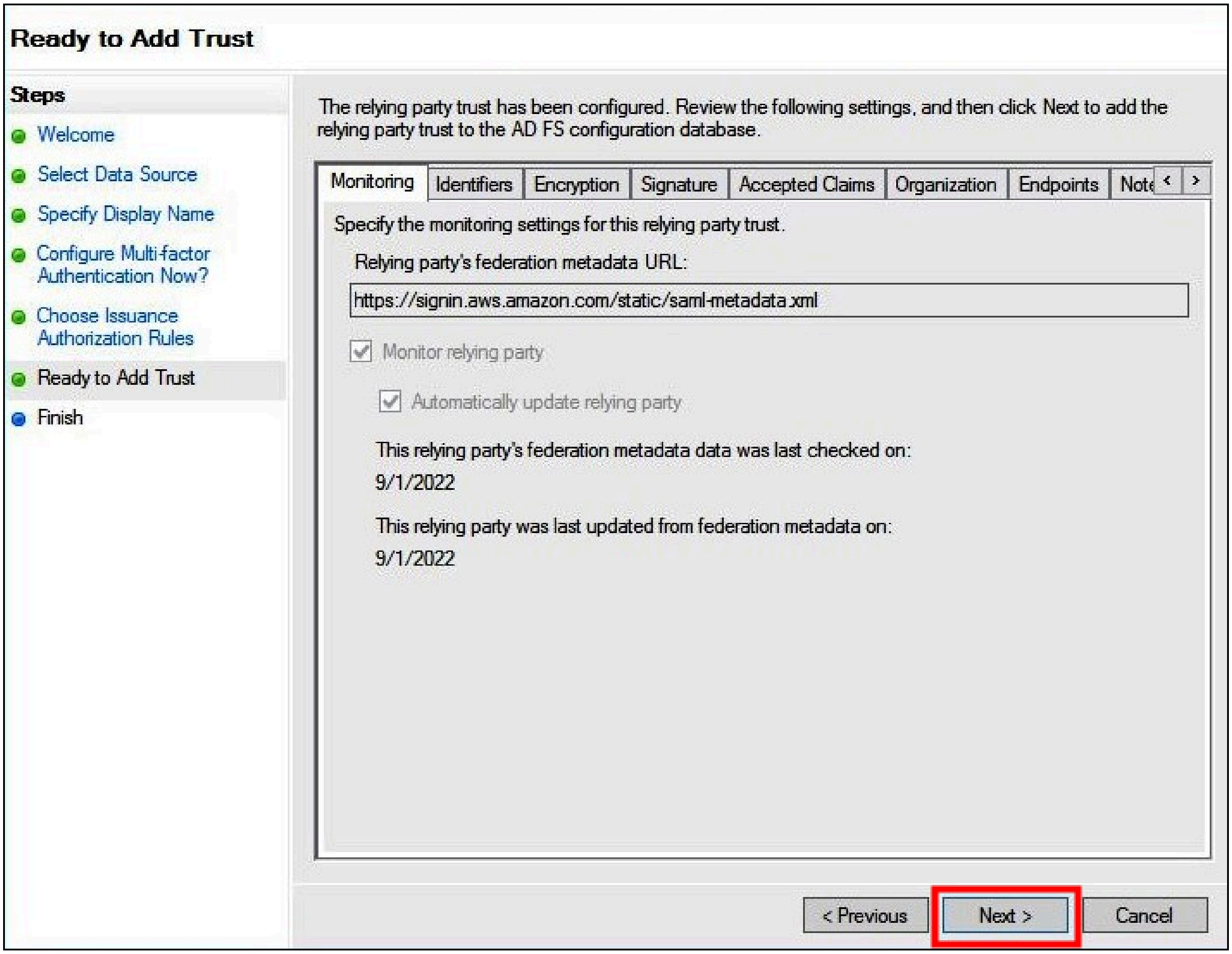
12. Choose **Next**.

13. On the **Choose Issuance Authorization Rules** page, select **Permit all users to access this relying party**.

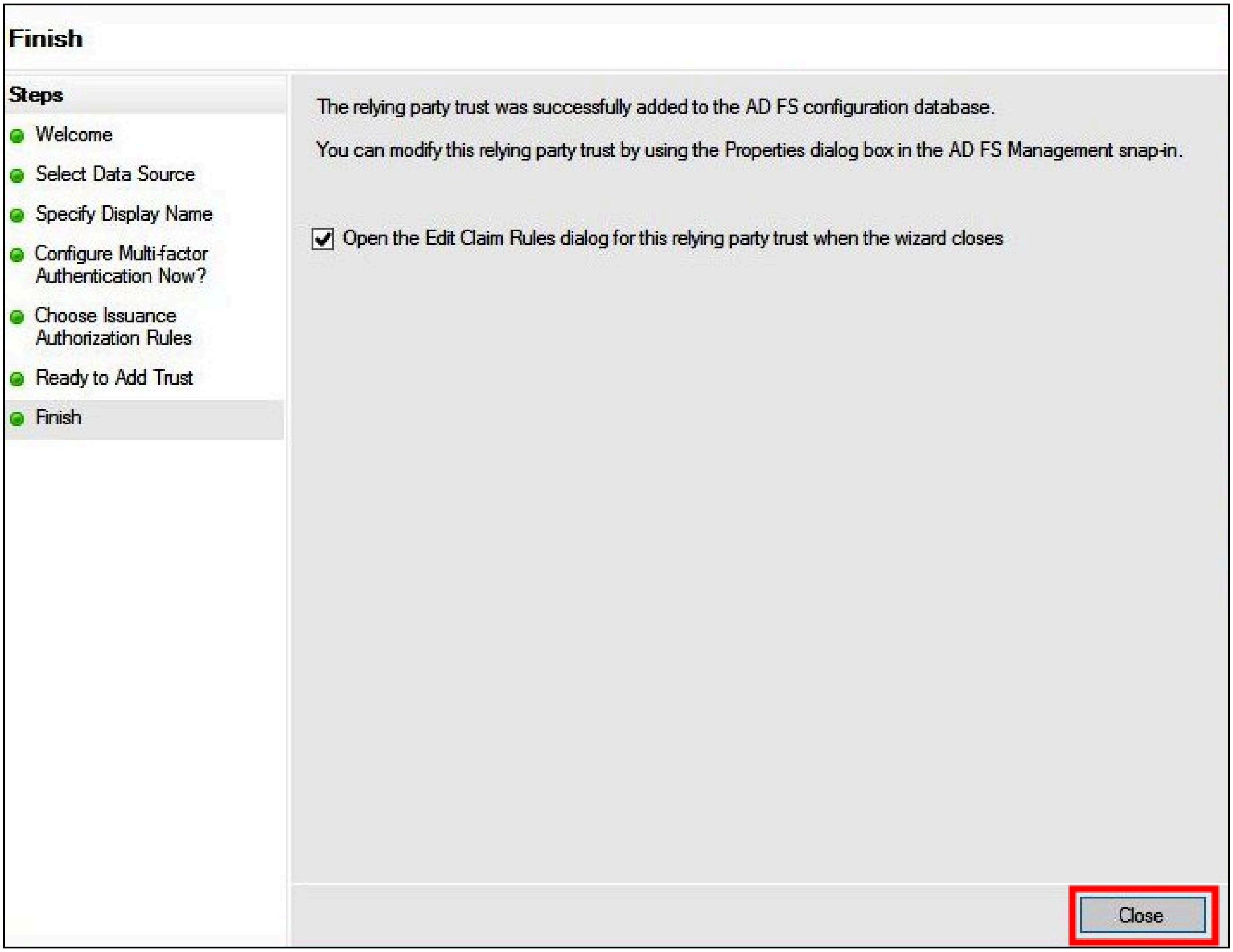
    This option allows all users in Active Directory to use AD FS with AWS as a relying party. You should consider your security requirements and adjust this configuration accordingly.



14. Choose **Next**.
15. On the **Ready to Add Trust** page, choose **Next** to add the relying party trust to the AD FS configuration database.



16. On the **Finish** page, choose **Close**.



Configuring SAML claim rules for the relying party

In this task, you create two sets of claim rules.

The first set, rules 1–4, contains AD FS claim rules that are required to assume an IAM role based on AD group membership. These are the same rules that you create if you want to establish federated access to the [AWS Management Console](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_enable-console-saml.html).

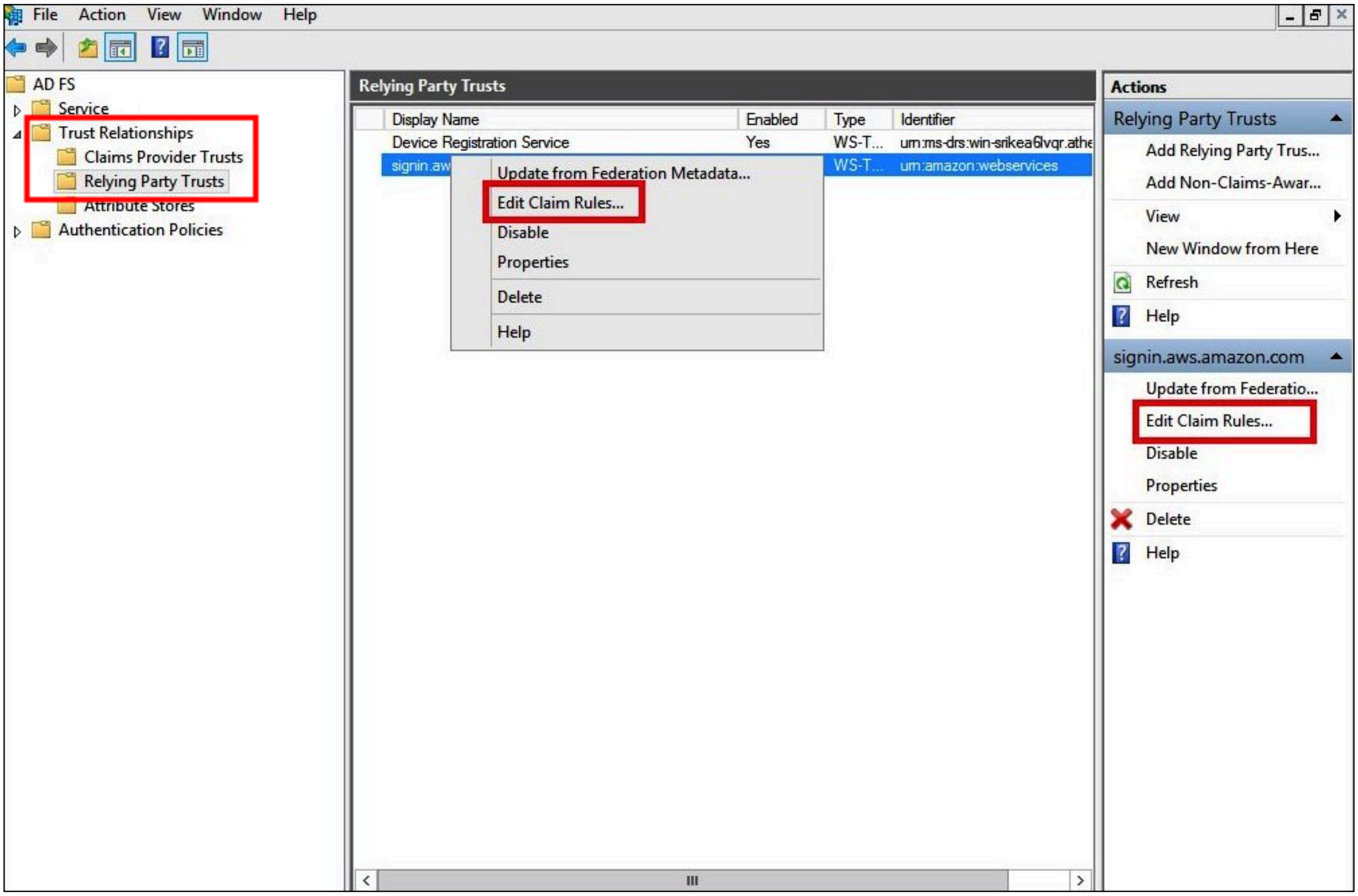
The second set, rules 5–6, are claim rules required for Athena access control.

To create AD FS claim rules

1. In the AD FS Management console navigation pane, choose **Trust Relationships**, **Relying Party Trusts**.

2. Find the relying party that you created in the previous section.

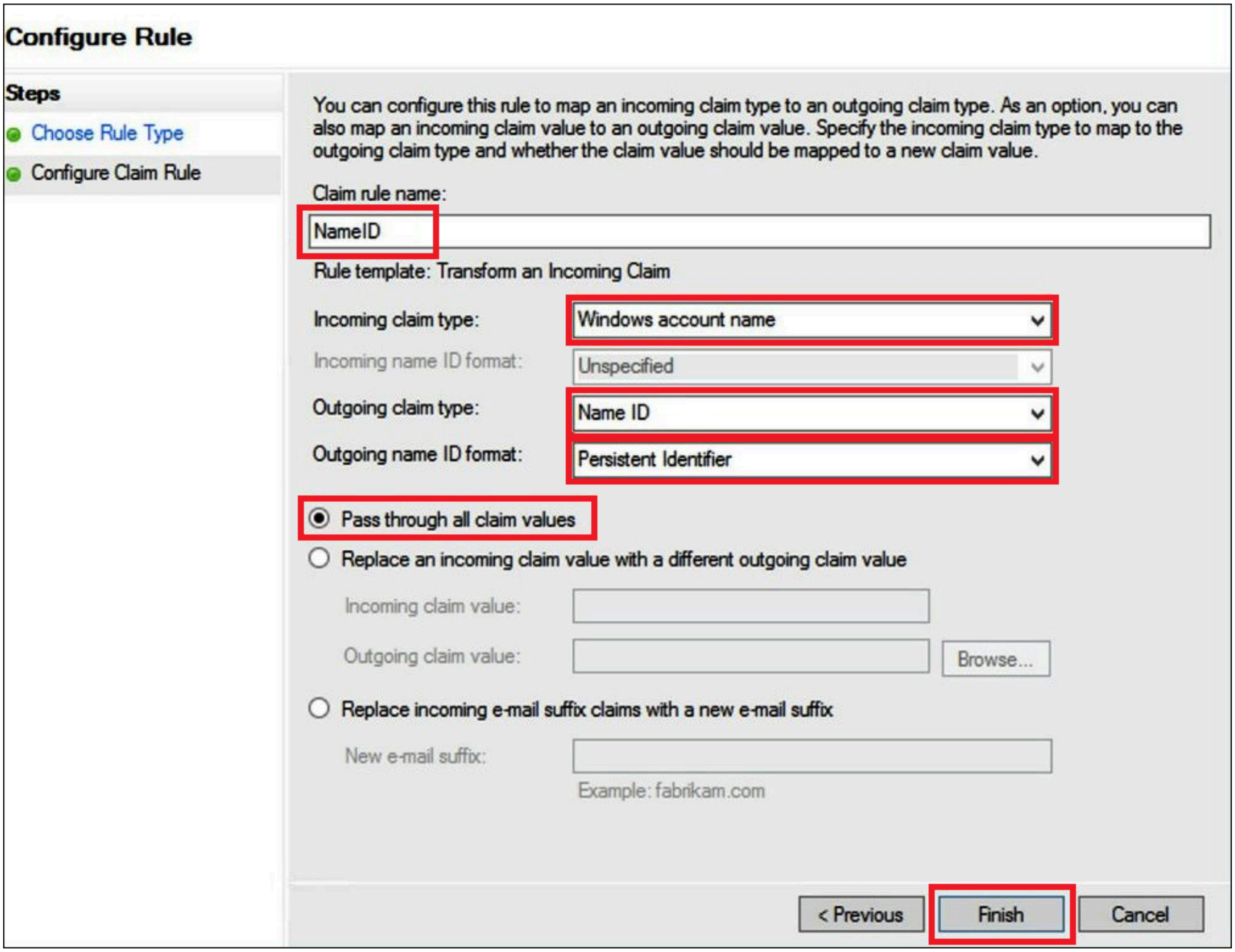
3. Right-click the relying party and choose **Edit Claim Rules**, or choose **Edit Claim Rules** from the **Actions** menu.



4. Choose **Add Rule.**

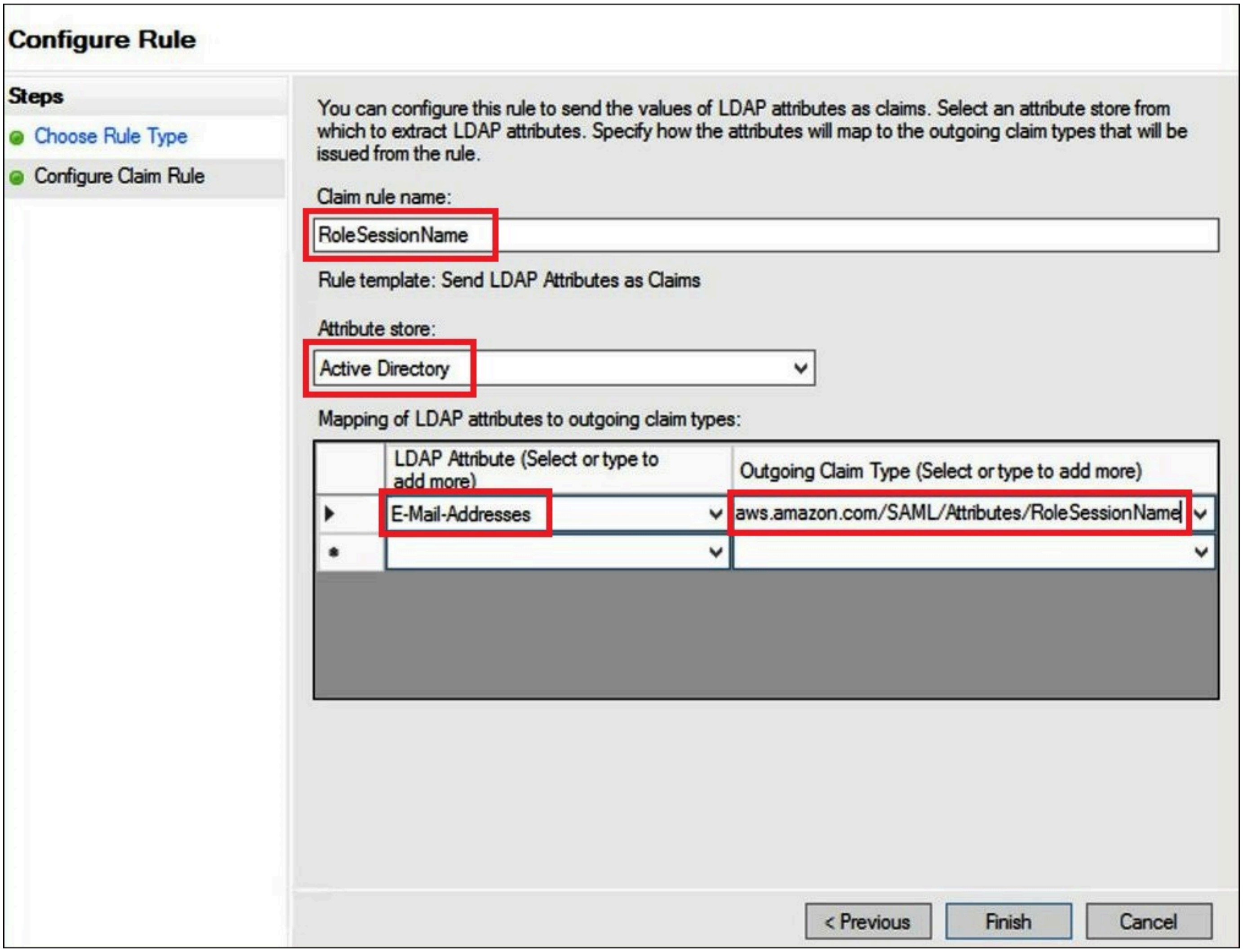
5. On the **Configure Rule** page of the Add Transform Claim Rule Wizard, enter the following information to create claim rule 1, and then choose **Finish**.

   - For **Claim Rule name**, enter **`NameID`**.
   - For **Rule template**, use **Transform an Incoming Claim**.
   - For **Incoming claim type**, choose **Windows account name**.
   - For **Outgoing claim type**, choose **Name ID**.
   - For **Outgoing name ID format**, choose **Persistent Identifier**.
   - Select **Pass through all claim values**.



6. Choose **Add Rule**, and then enter the following information to create claim rule 2, and then choose **Finish**.

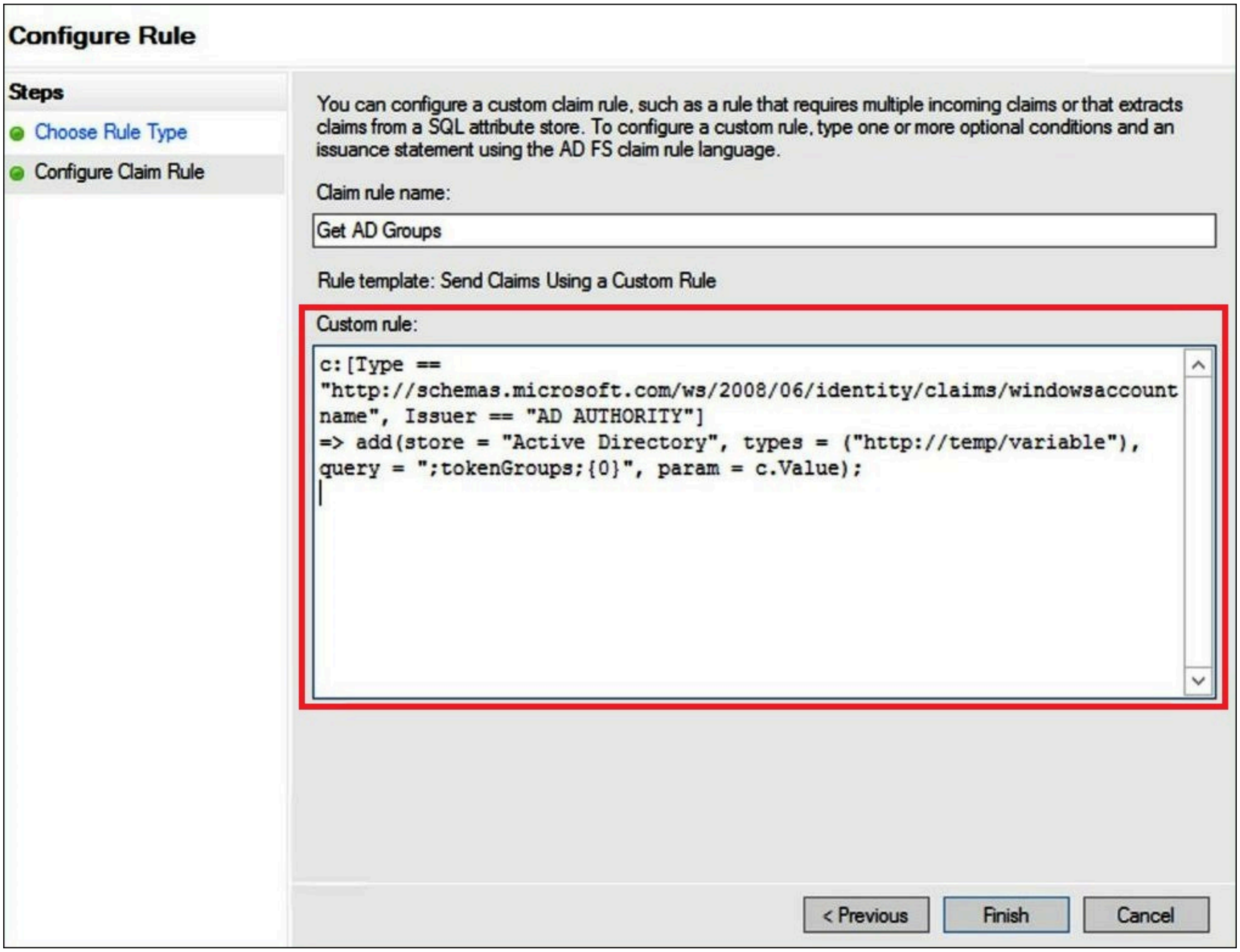
   - For **Claim rule name**, enter **`RoleSessionName`**.
   - For **Rule template**, use **Send LDAP Attribute as Claims**.
   - For **Attribute store**, choose **Active Directory**.
   - For **Mapping of LDAP attributes to outgoing claim types**, add the attribute **`E-Mail-Addresses`**. For the **Outgoing Claim Type**, enter **`https://aws.amazon.com/SAML/Attributes/RoleSessionName`**.



7. Choose **Add Rule**, and then enter the following information to create claim rule 3, and then choose **Finish**.

   - For **Claim rule name**, enter **`Get AD Groups`**.
   - For **Rule template**, use **Send Claims Using a Custom Rule**.
   - For **Custom rule**, enter the following code:

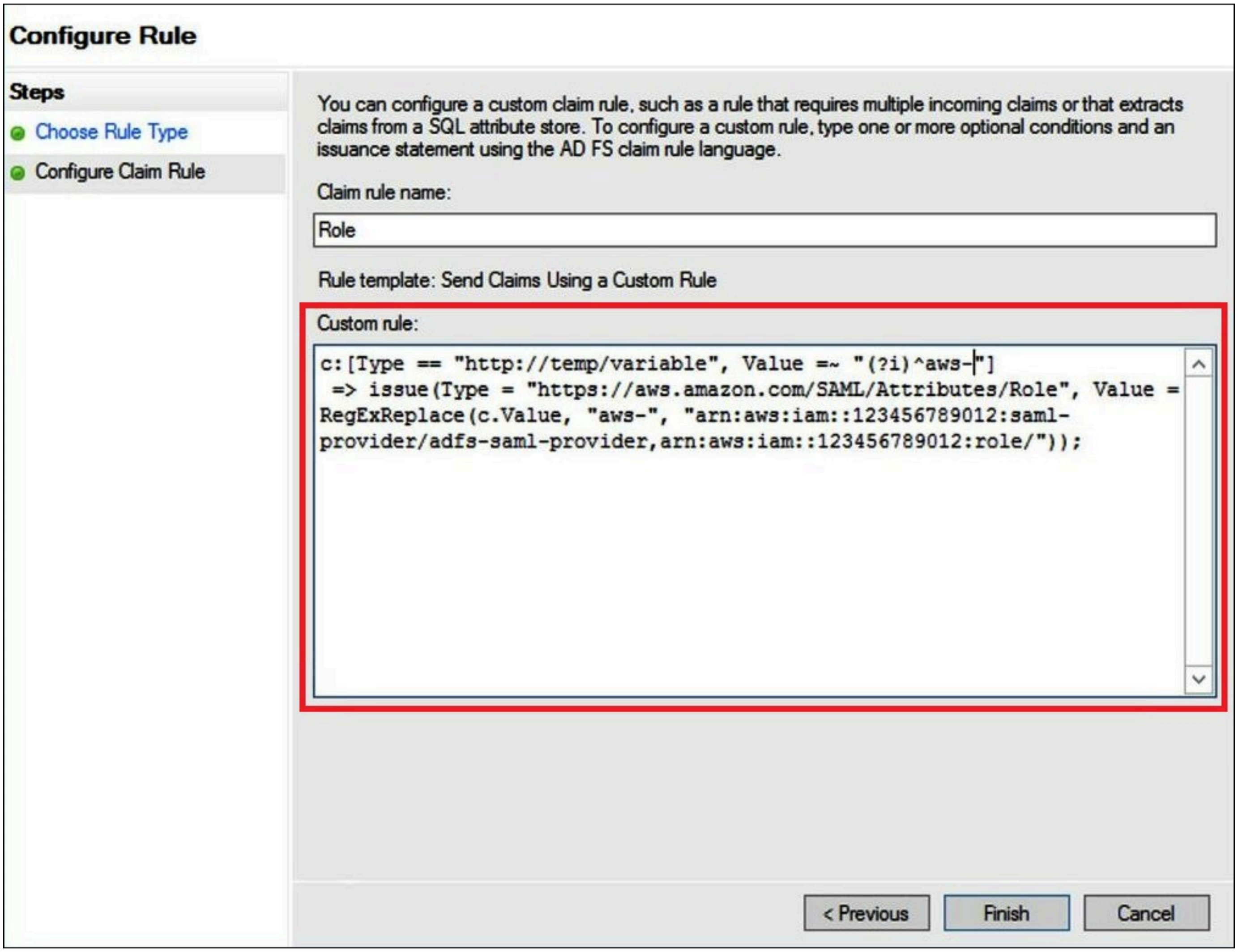
```
c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/
windowsaccountname",
 Issuer == "AD AUTHORITY"]=> add(store = "Active Directory", types = ("http://
temp/variable"),
 query = ";tokenGroups;{0}", param = c.Value);
```



8. Choose **Add Rule**. Enter the following information to create claim rule 4, and then choose **Finish**.

   - For **Claim rule name**, enter **`Role`**.
   - For **Rule template**, use **Send Claims Using a Custom Rule**.
   - For **Custom rule**, enter the following code with your account number and name of the SAML provider that you created earlier:

```
c:[Type == "http://temp/variable", Value =~ "(?i)^aws-"]=> issue(Type = "https://
aws.amazon.com/SAML/Attributes/Role",
Value = RegExReplace(c.Value, "aws-", "arn:aws:iam::AWS_ACCOUNT_NUMBER:saml-
provider/adfs-saml-provider,arn:aws:iam:: AWS_ACCOUNT_NUMBER:role/"));
```

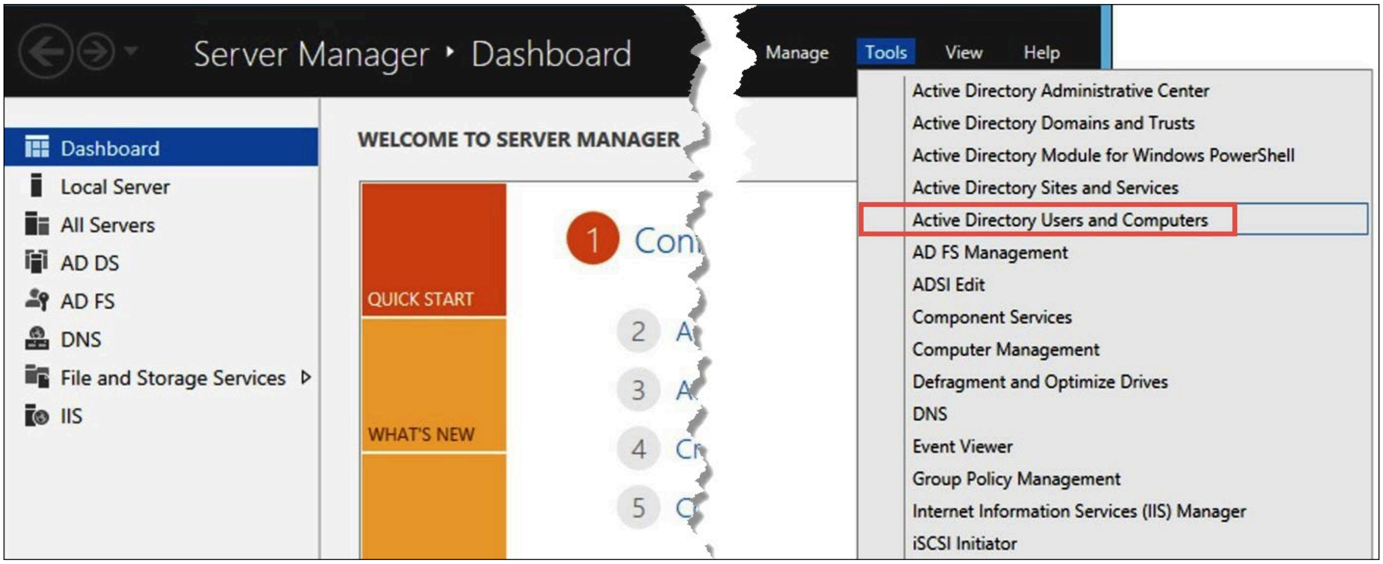


3. Creating Active Directory users and groups

Now you are ready to create AD users that will access Athena, and AD groups to place them in so that you can control levels of access by group. After you create AD groups that categorize patterns of data access, you add your users to those groups.

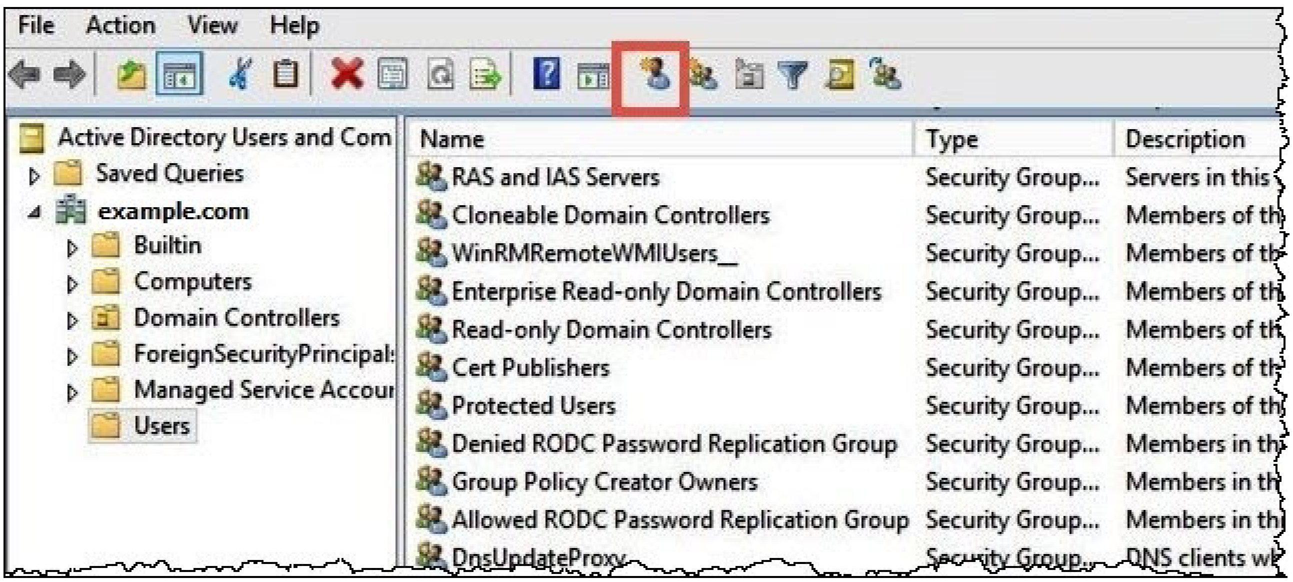
To create AD users for access to Athena

1. On the Server Manager dashboard, choose **Tools**, and then choose **Active Directory Users and Computers**.

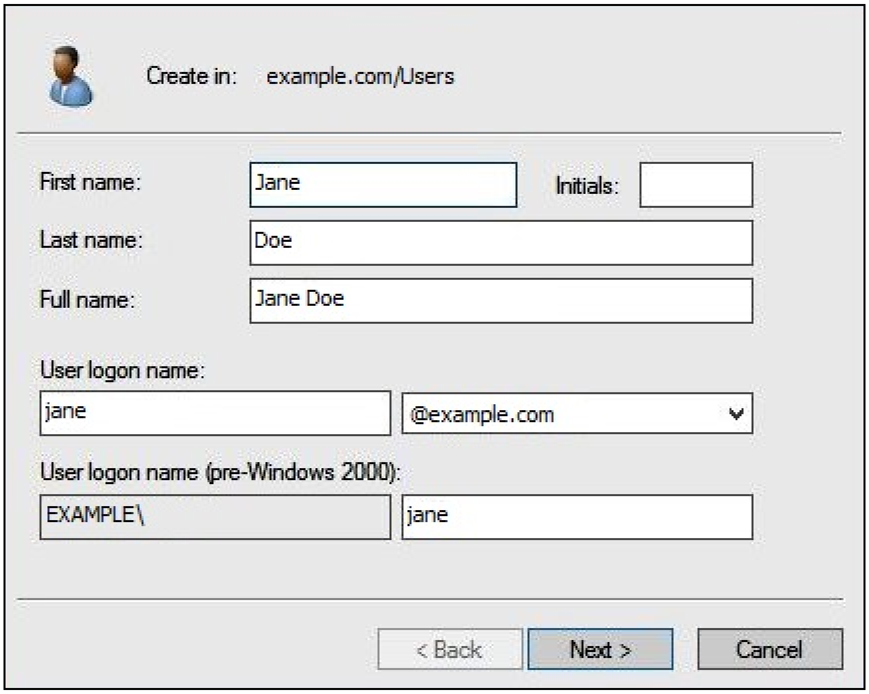


2. In the navigation pane, choose **Users**.

3. On the **Active Directory Users and Computers** tool bar, choose the **Create user** option.



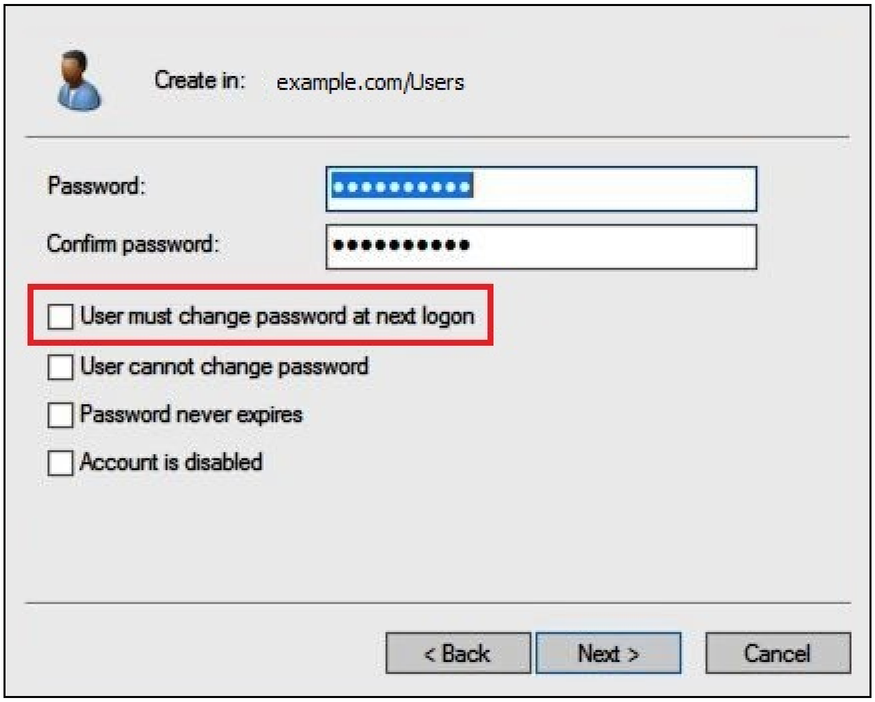
4. In the **New Object – User** dialog box, for **First name**, **Last name**, and **Full name**, enter a name. This tutorial uses **`Jane Doe`**.



5. Choose **Next**.

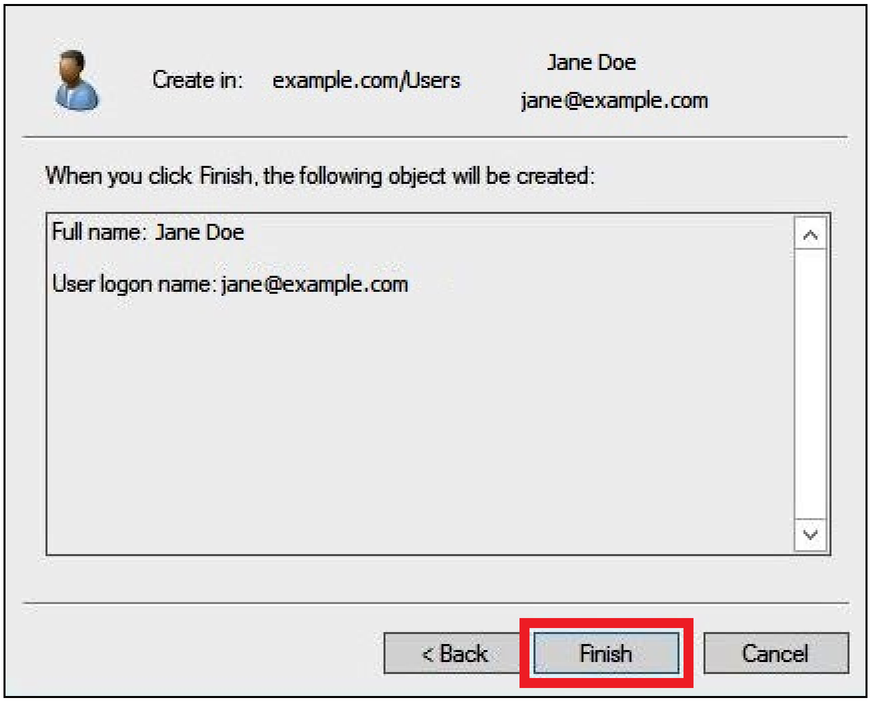
6. For **Password**, enter a password, and then retype to confirm.

   For simplicity, this tutorial deselects **User must change password at next sign on**. In real-world scenarios, you should require newly created users to change their password.

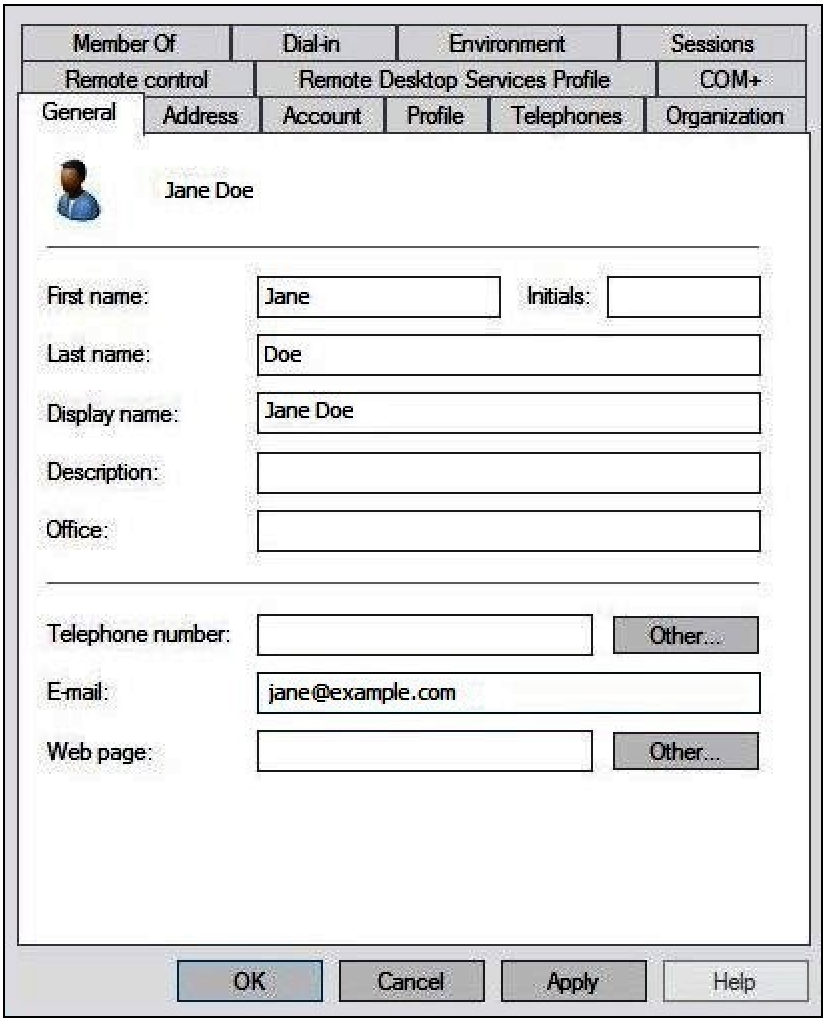


7. Choose **Next**.

8. Choose **Finish.**



9. In **Active Directory Users and Computers**, choose the user name.
10. In the **Properties** dialog box for the user, for **E-mail**, enter an email address. This tutorial uses **`jane@example.com`**.



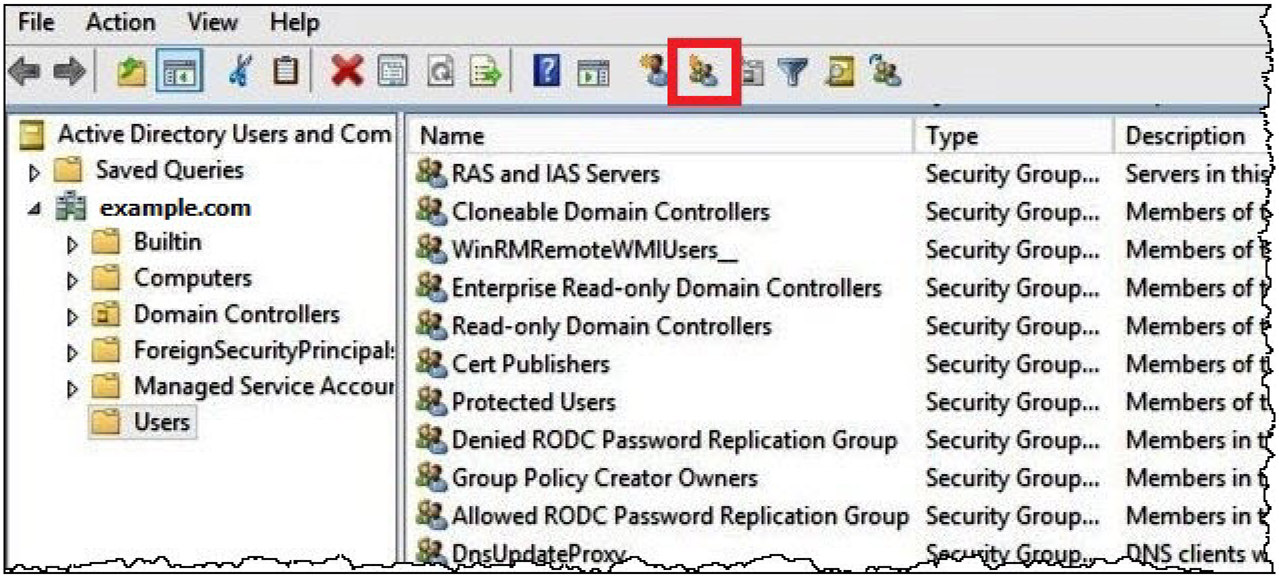
11. Choose **OK**.

Create AD groups to represent data access patterns

You can create AD groups whose members assume the `adfs-data-access` IAM role when they log in to AWS. The following example creates an AD group called aws-adfs-data-access.

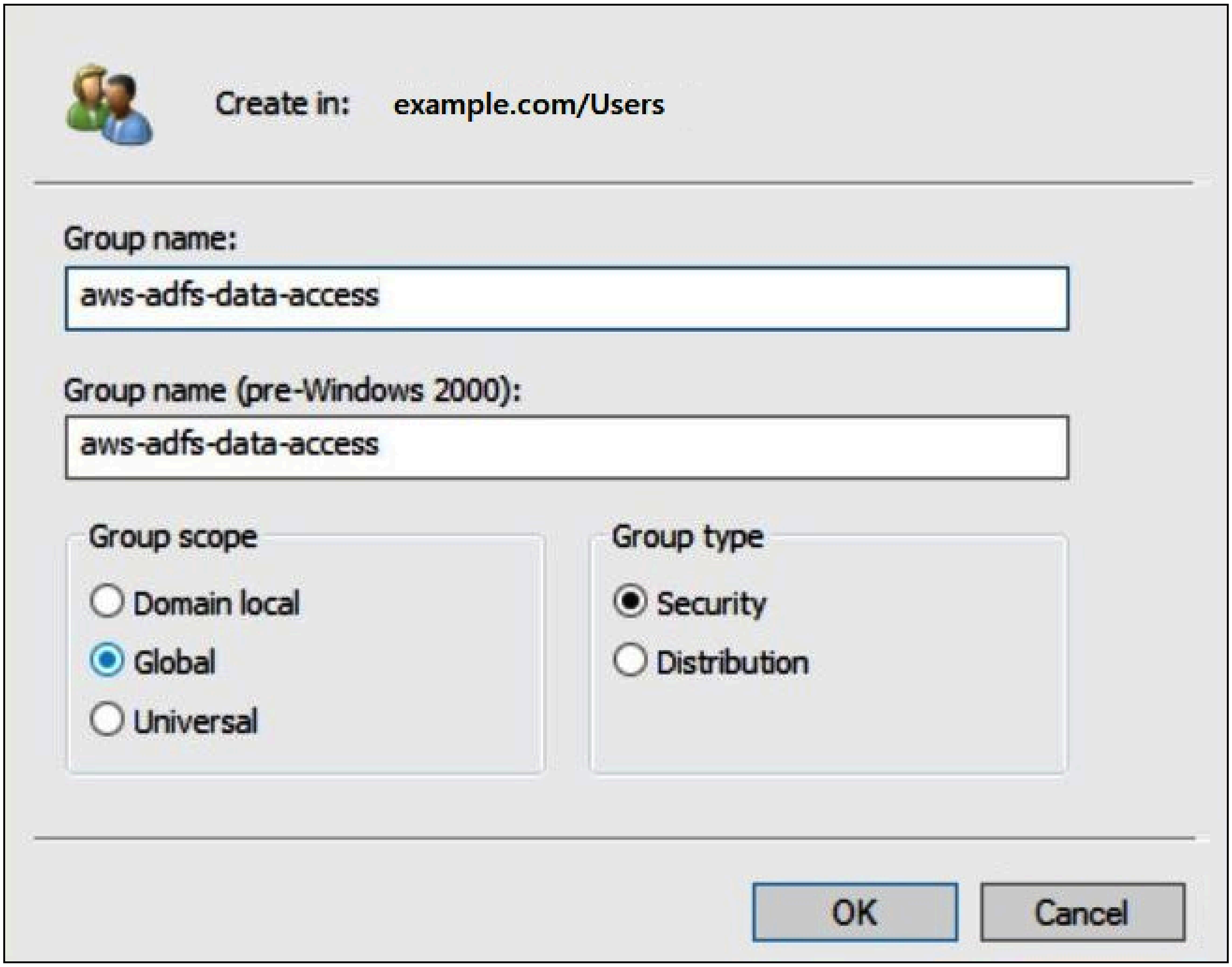
To create an AD group

1. On the Server Manager Dashboard, from the **Tools** menu, choose **Active Directory Users and Computers.**
2. On the tool bar, choose the **Create new group** option.



3. In the **New Object - Group** dialog box, enter the following information:

   - For **Group name**, enter **`aws-adfs-data-access`**.
   - For **Group scope**, select **Global**.
   - For **Group type**, select **Security**.



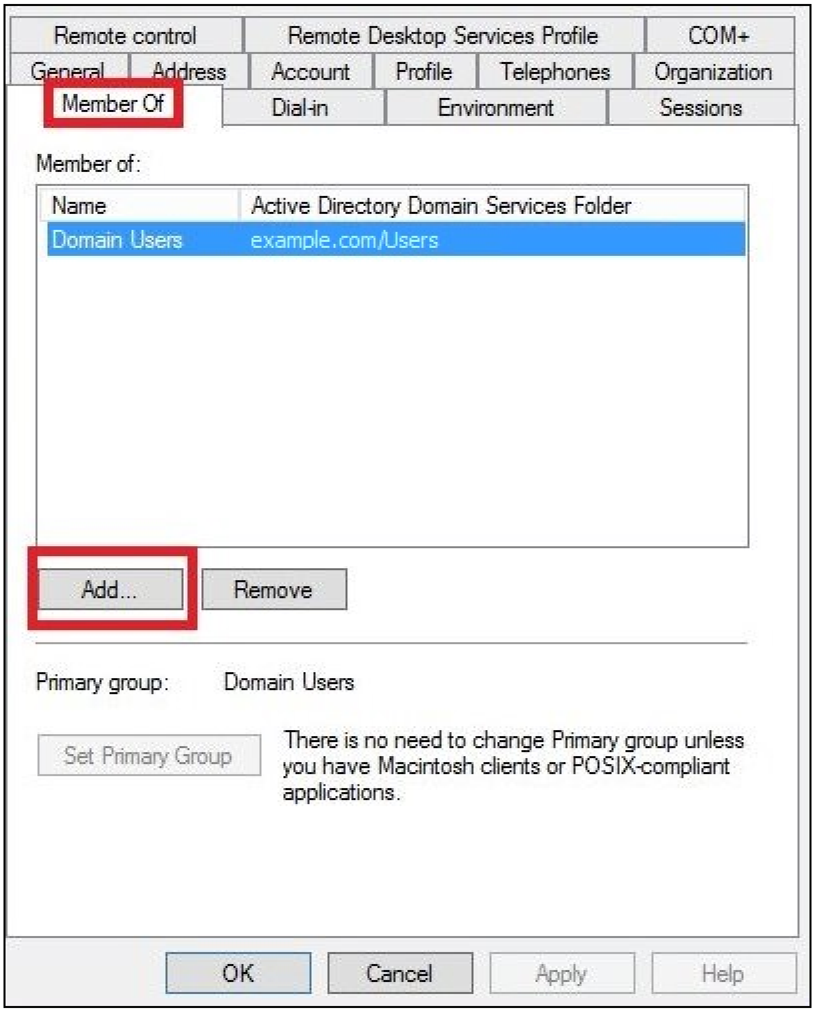
4. Choose **OK**.

Add AD users to appropriate groups

Now that you have created both an AD user and an AD group, you can add the user to the group.

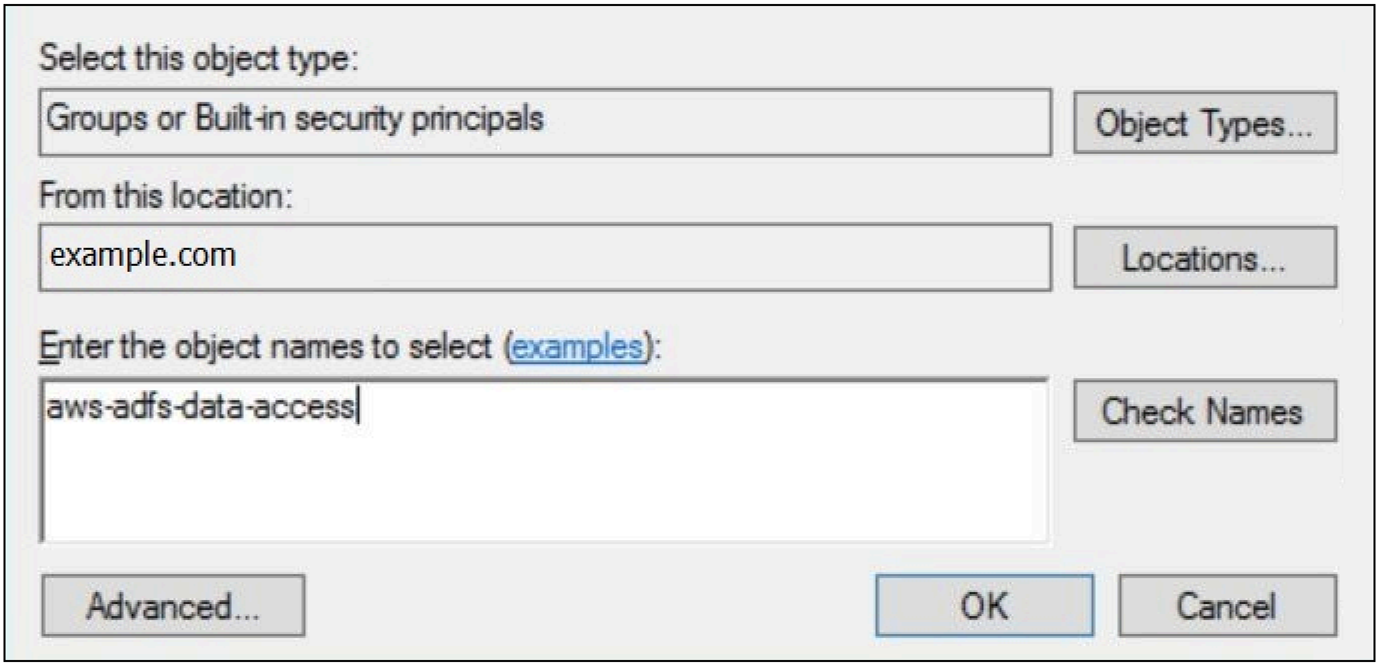
To add an AD user to an AD group

1. On the Server Manager Dashboard, on the **Tools** menu, choose **Active Directory Users and Computers**.
2. For **First name** and **Last name**, choose a user (for example, **Jane Doe**).
3. In the **Properties** dialog box for the user, on the **Member Of** tab, choose **Add**.



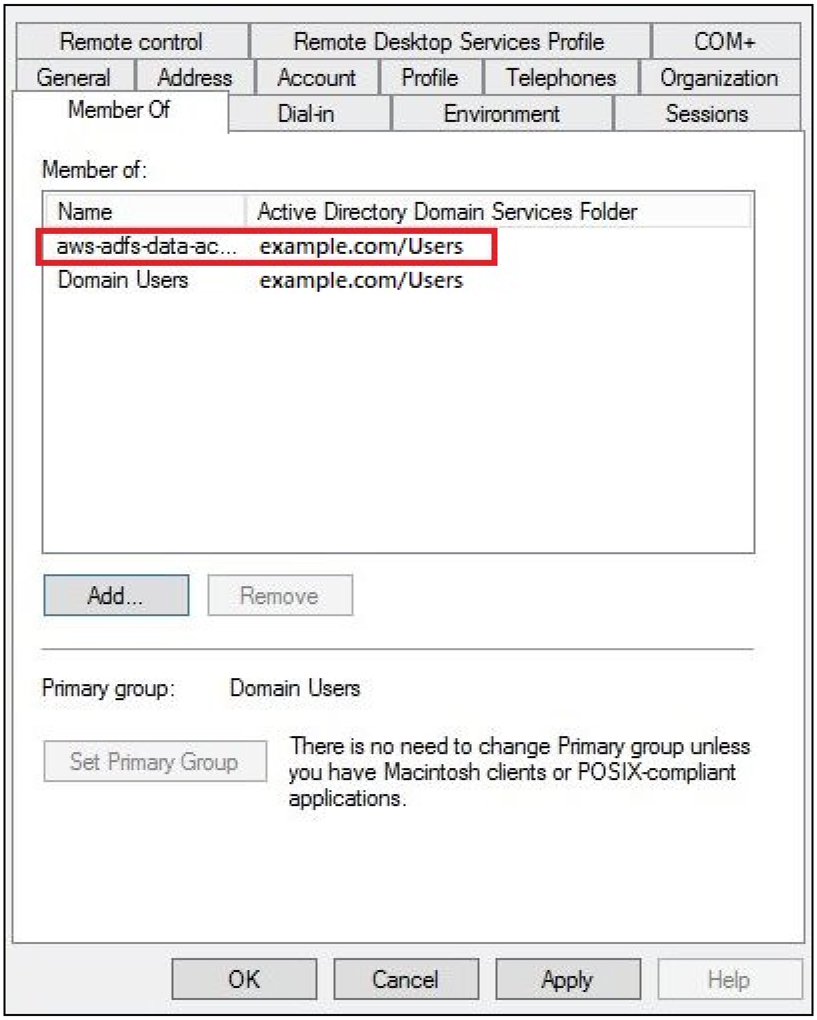
4. Add one or more AD FS groups according to your requirements. This tutorial adds the **aws-adfs-data-access** group.

5. In the **Select Groups** dialog box, for **Enter the object names to select**, enter the name of the AD FS group that you created (for example, **`aws-adfs-data-access`**), and then choose **Check Names**.



6. Choose **OK**.

   In the **Properties** dialog box for the user, the name of the AD group appears in the **Member of** list.



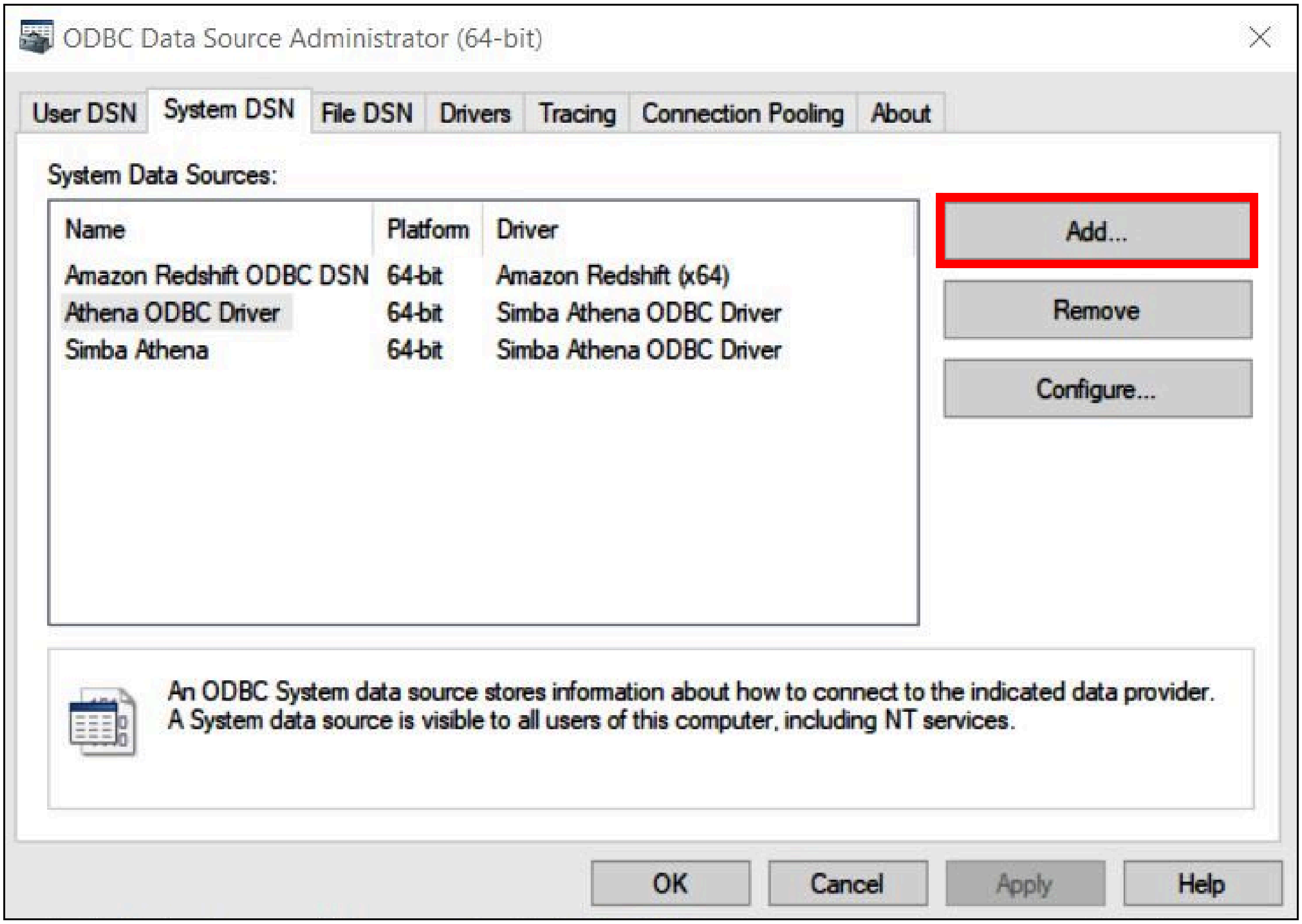
7. Choose **Apply**, then choose **OK**.

4. Configuring the AD FS ODBC connection to Athena

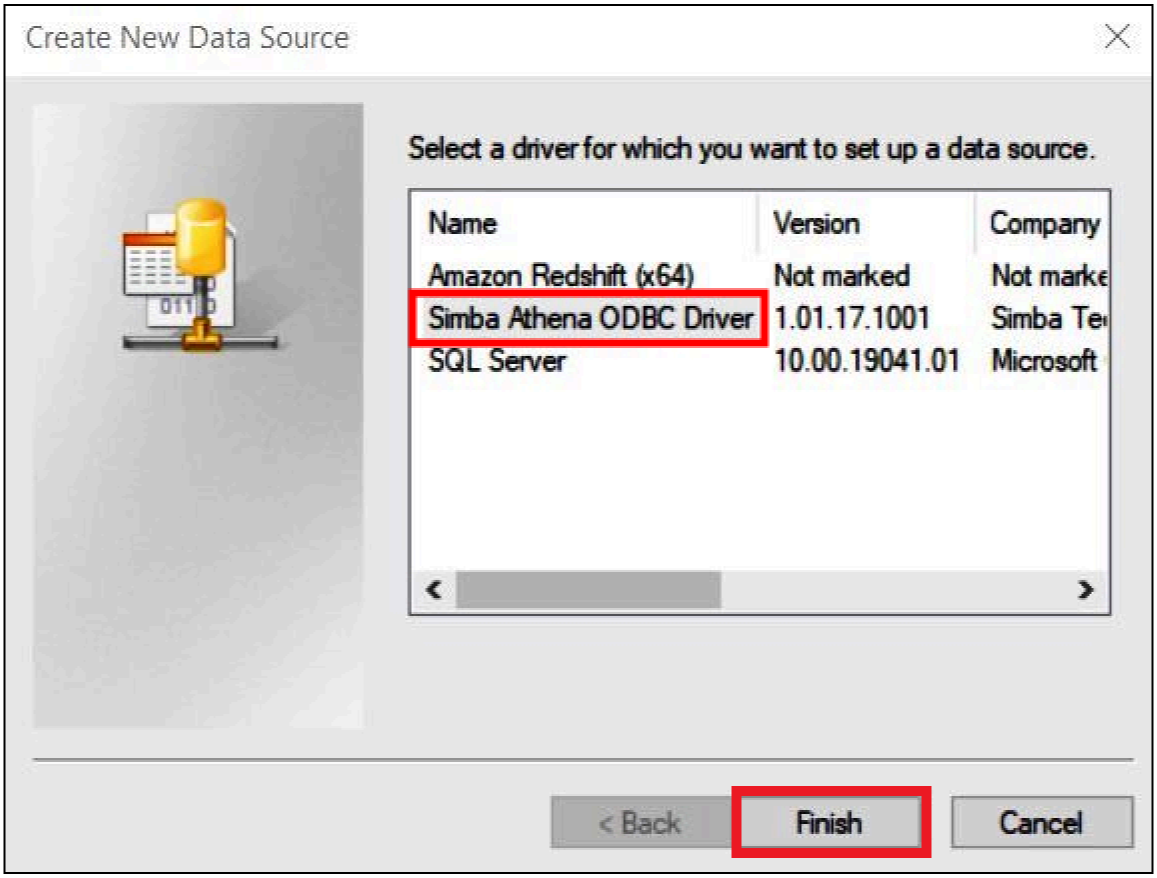
After you have created your AD users and groups, you are ready to use the ODBC Data Sources program in Windows to configure your Athena ODBC connection for AD FS.

To configure the AD FS ODBC connection to Athena

1. Install the ODBC driver for Athena. For download links, see [Connect to Amazon Athena with ODBC](#).
2. In Windows, choose **Start**, **ODBC Data Sources**.
3. In the **ODBC Data Source Administrator** program, choose **Add**.

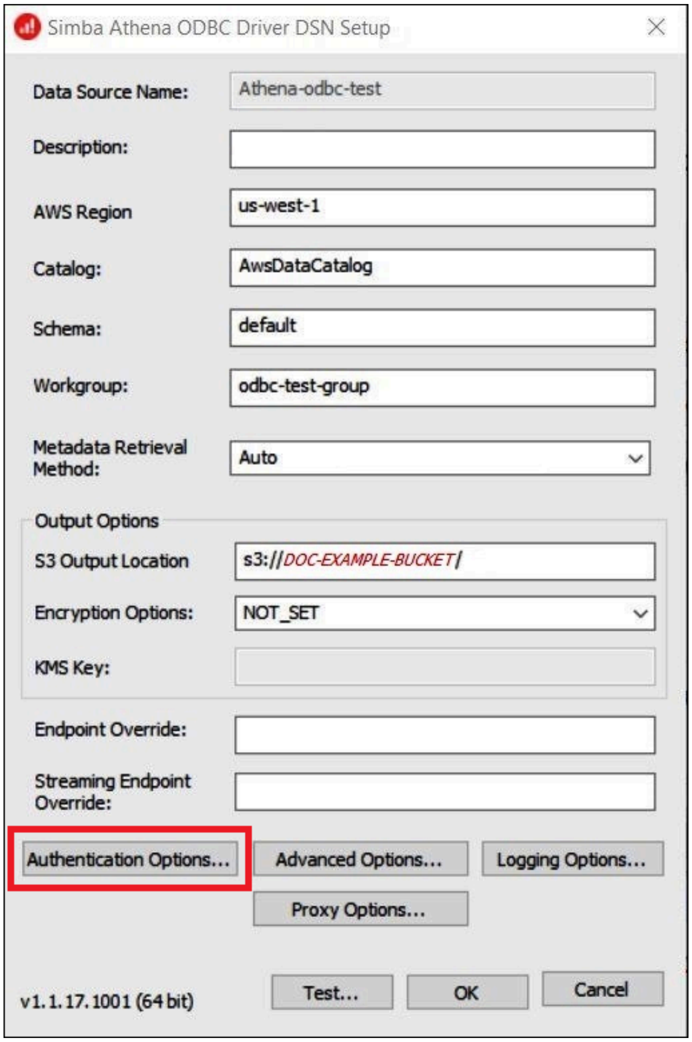


4. In the **Create New Data Source** dialog box, choose **Simba Athena ODBC Driver**, and then choose **Finish**.



5. In the **Simba Athena ODBC Driver DSN Setup** dialog box, enter the following values:

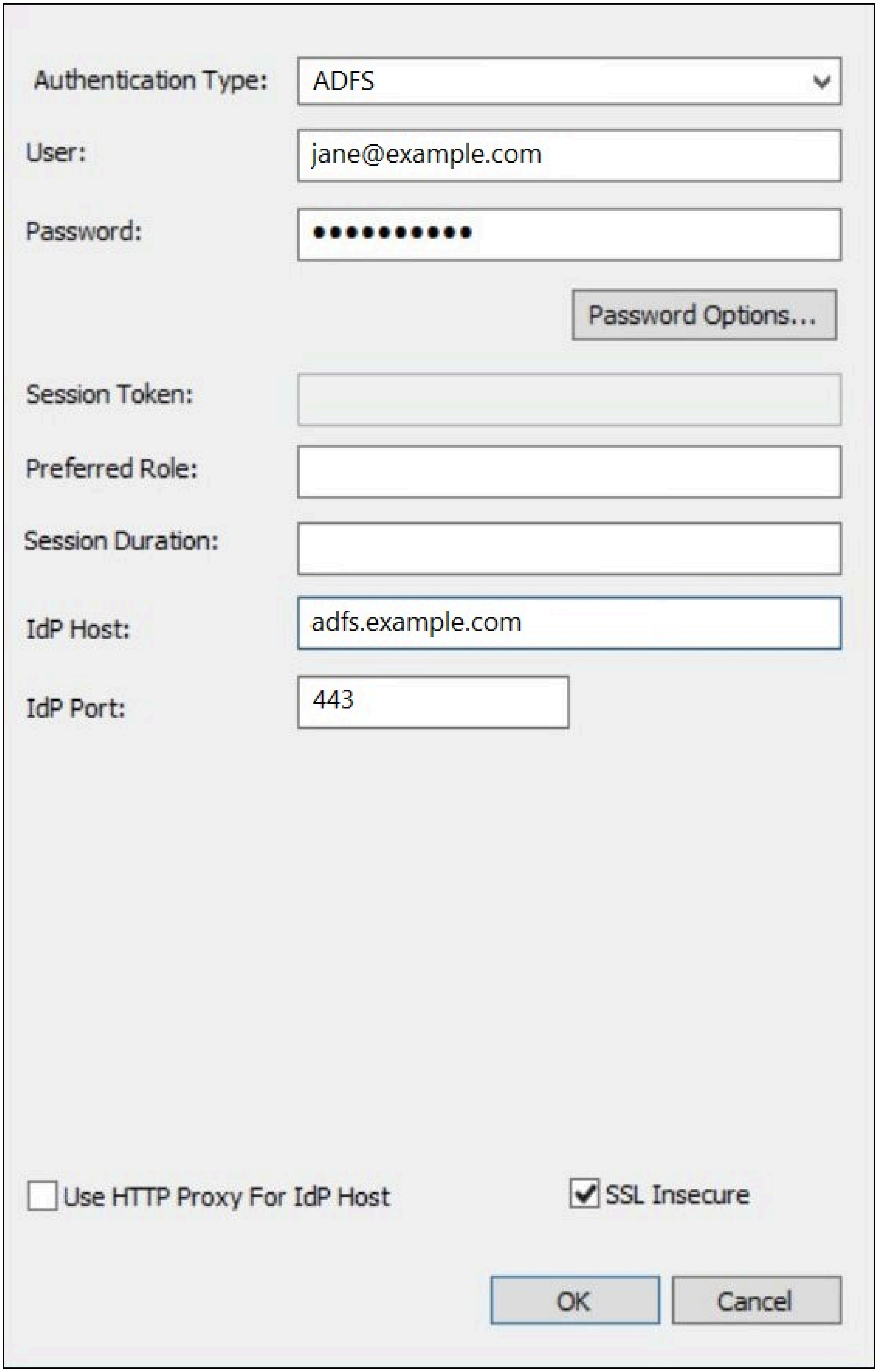
   - For **Data Source Name,** enter a name for your data source (for example, **`Athena-odbc-test`**).
   - For **Description**, enter a description for your data source.
   - For **AWS Region**, enter the AWS Region that you are using (for example, **`us-west-1`**).
   - For **S3 Output Location**, enter the Amazon S3 path where you want your output to be stored.



6. Choose **Authentication Options**.

7. In the **Authentication Options** dialog box, specify the following values:

   - For **Authentication Type**, choose **ADFS**.
   - For **User,** enter the user's email address (for example, **`jane@example.com`**).
   - For **Password**, enter the user's ADFS password.
   - For **IdP Host**, enter the AD FS server name (for example, **`adfs.example.com`**).
   - For **IdP Port**, use the default value **443**.
   - Select the **SSL Insecure** option.


Authentication Type:
ADFS
User:
jane@example.com
Password:
Password Options...
Session Token:
Preferred Role:
Session Duration:
IdP Host:
adfs.example.com
IdP Port:
443
Use HTTP Proxy For IdP Host
SSL Insecure
OK
Cancel

8. Choose **OK** to close **Authentication Options**.
9. Choose **Test** to test the connection, or **OK** to finish.

Configure SSO for ODBC using the Okta plugin and Okta Identity Provider

This page describes how to configure the Amazon Athena ODBC driver and Okta plugin to add single sign-on (SSO) capability using the Okta identity provider.

Prerequisites

Completing the steps in this tutorial requires the following:

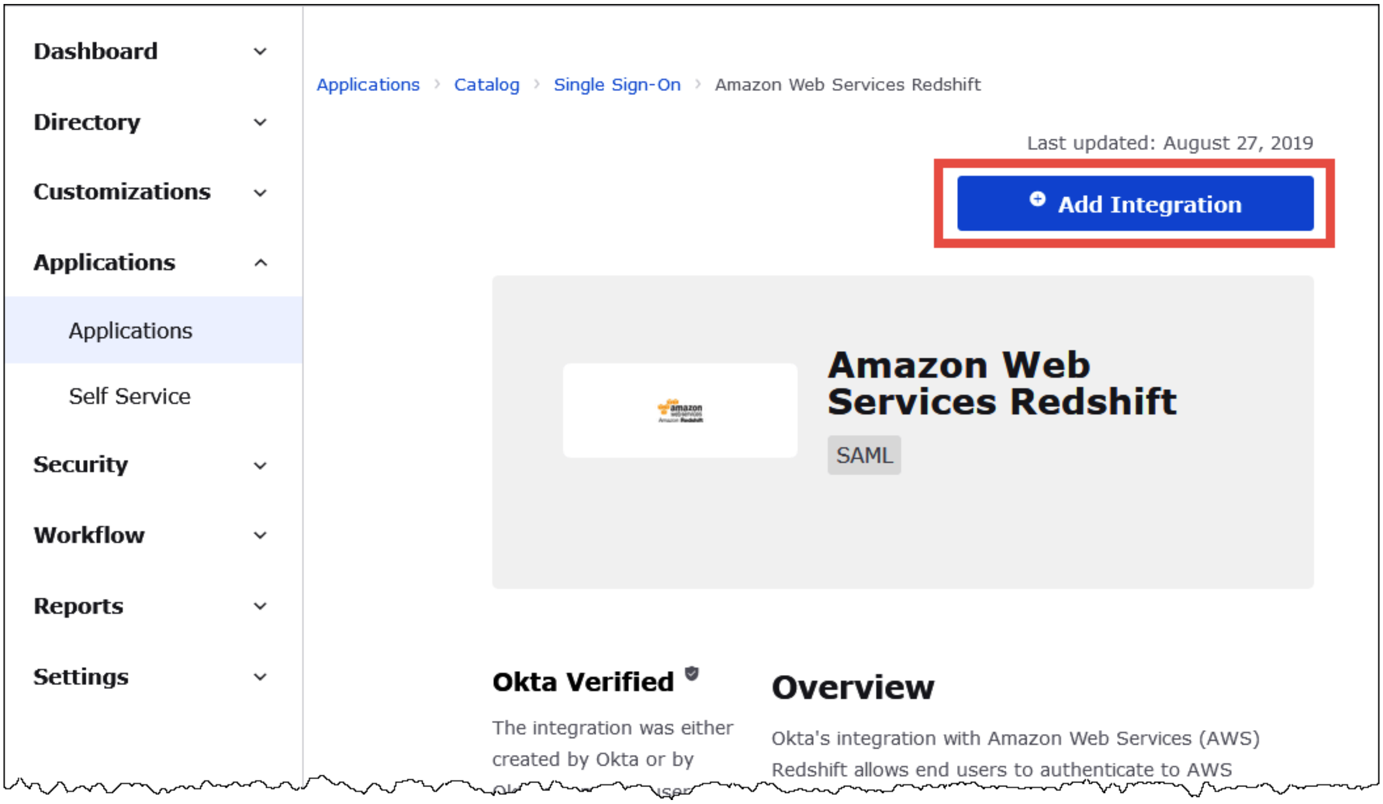
- Amazon Athena ODBC driver. For download links, see Connect to Amazon Athena with ODBC.
- An IAM Role that you want to use with SAML. For more information, see Creating a role for SAML 2.0 federation in the *IAM User Guide*.
- An Okta account. For information, visit Okta.com.

Creating an app integration in Okta

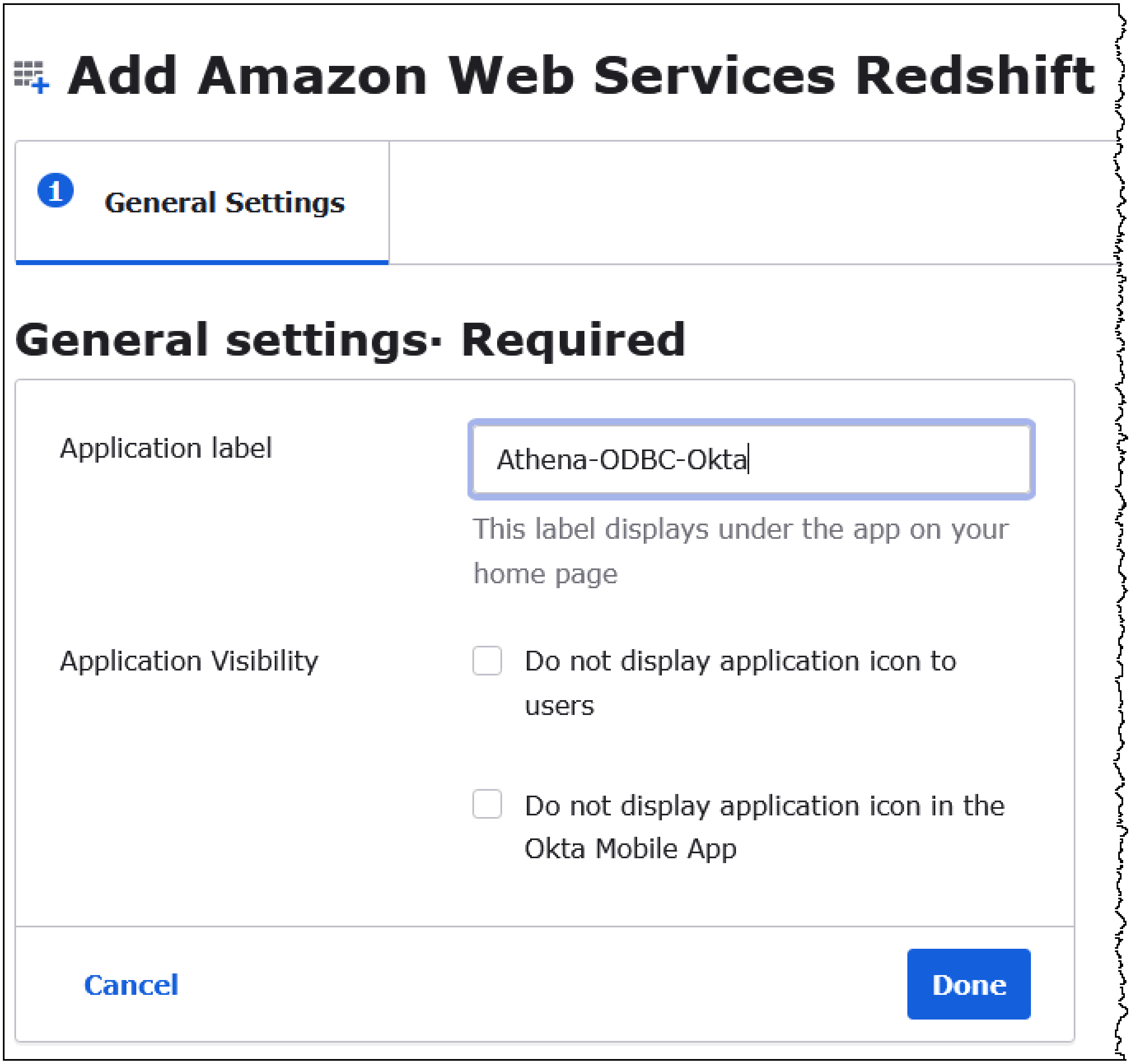
First, use the Okta dashboard to create and configure a SAML 2.0 app for single sign-on to Athena. You can use an existing Redshift application in Okta to configure access to Athena.

To create an app integration in Okta

1. Log in to the admin page for your account on Okta.com.
2. In the navigation panel, choose **Applications**, **Applications.**
3. On the **Applications** page, choose **Browse App Catalog.**
4. On the **Browse App Integration Catalog** page, in the **Use Case** section, choose **All Integrations**.
5. In the search box, enter **Amazon Web Services Redshift**, and then choose **Amazon Web Services Redshift SAML**.
6. Choose **Add Integration**.

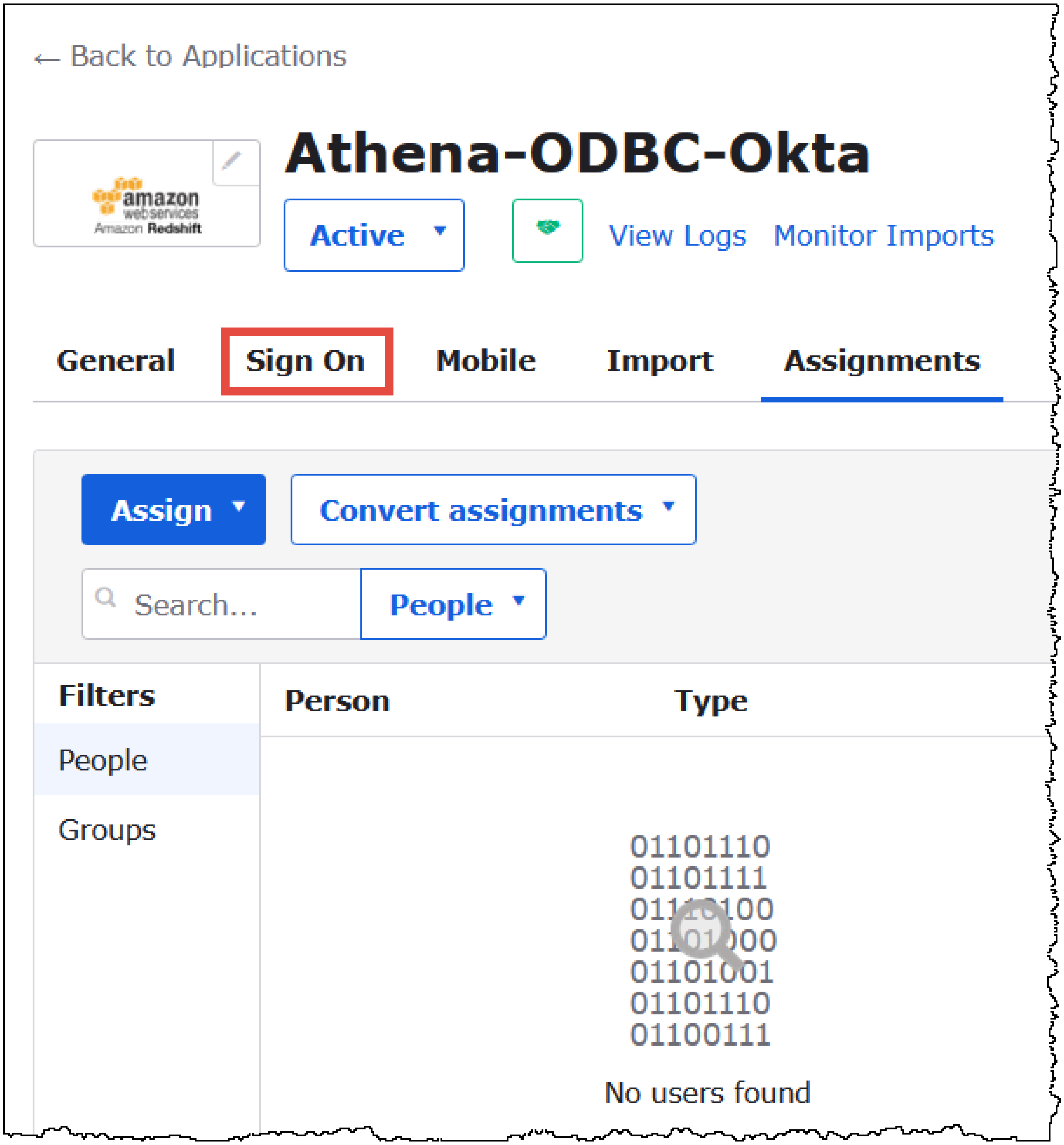


7. In the **General Settings Required** section, for **Application label**, enter a name for the application. This tutorial uses the name **Athena-ODBC-Okta.**

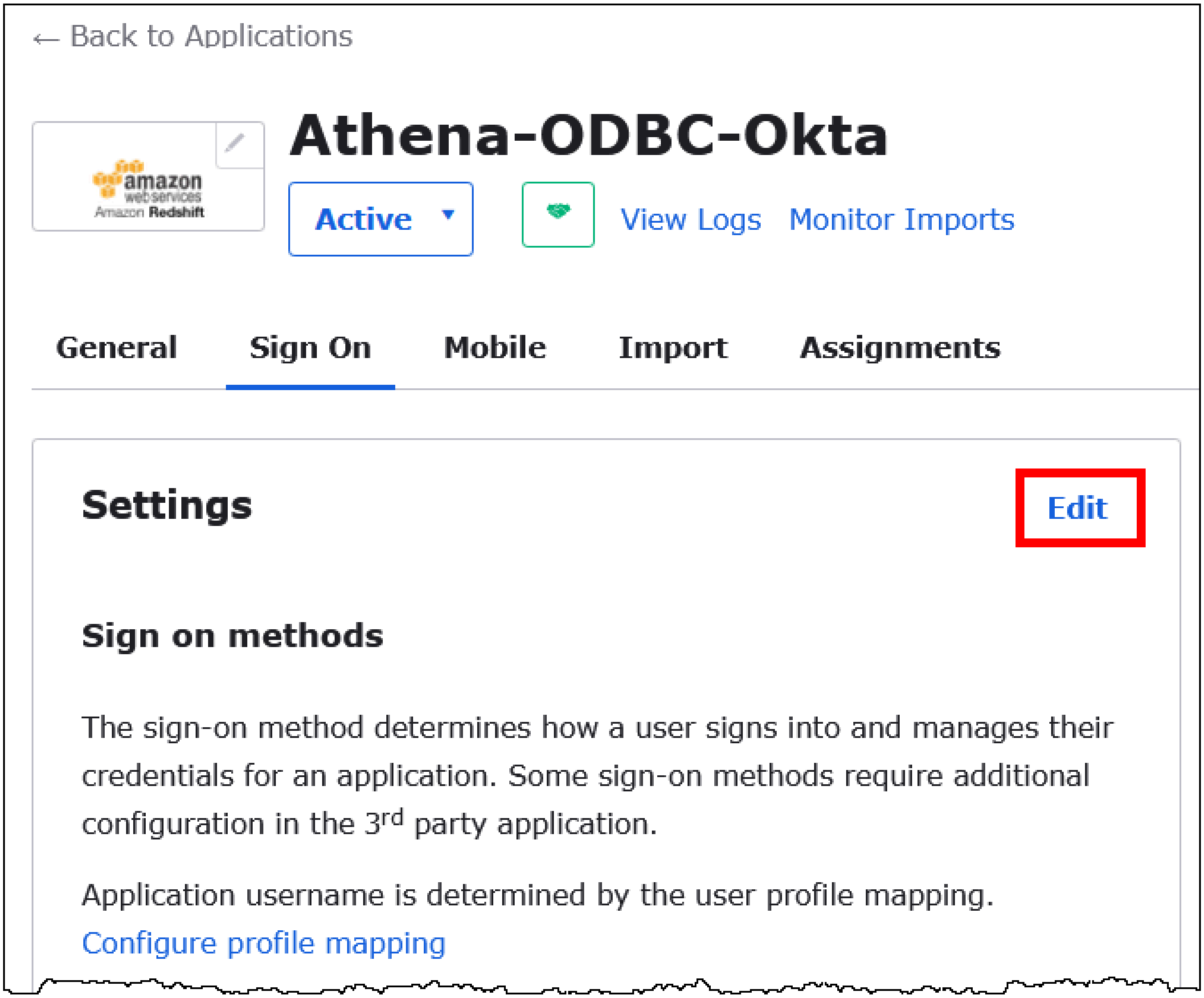


8. Choose **Done**.

9. On the page for your Okta application (for example, **Athena-ODBC-Okta**), choose **Sign On**.

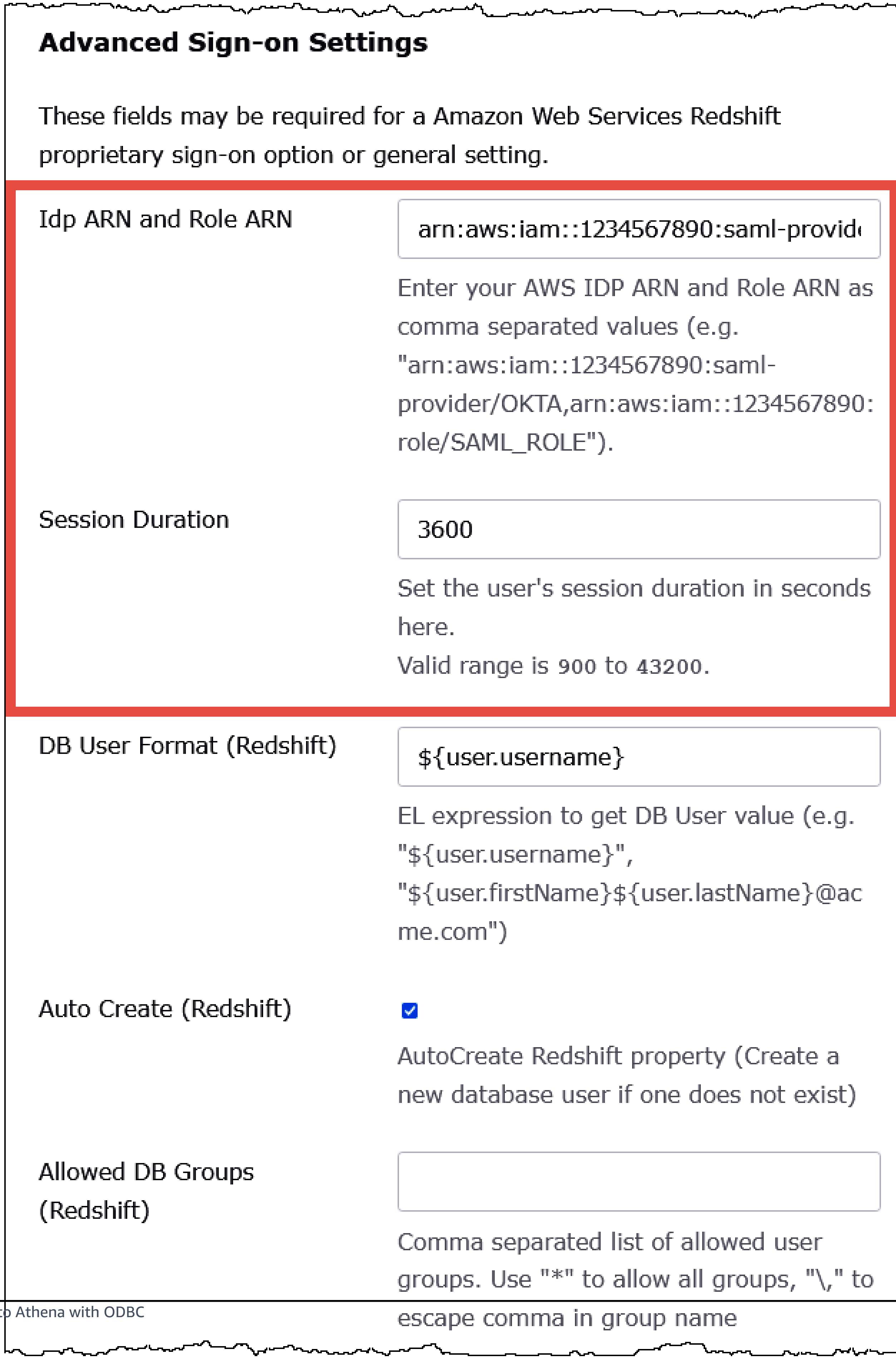


10. In the **Settings** section, choose **Edit**.



11. In the **Advanced Sign-on Settings** section, configure the following values.

    - For **IdP ARN and Role ARN**, enter your AWS IDP ARN and Role ARN as comma-separated values. For information about the IAM role format, see [Configuring SAML assertions for the authentication response](#) in the *IAM User Guide*.
    - For **Session Duration**, enter a value between 900 and 43200 seconds. This tutorial uses the default of 3600 (1 hour).


Advanced Sign-on Settings
These fields may be required for a Amazon Web Services Redshift proprietary sign-on option or general setting.
Idp ARN and Role ARN
arn:aws:iam::1234567890:saml-provid
Enter your AWS IDP ARN and Role ARN as comma separated values (e.g. "arn:aws:iam::1234567890:saml-provider/OKTA,arn:aws:iam::1234567890:role/SAML_ROLE").
Session Duration
3600
Set the user's session duration in seconds here.
Valid range is 900 to 43200.
DB User Format (Redshift)
${user.username}
EL expression to get DB User value (e.g. "${user.username}", "${user.firstName}${user.lastName}@acme.com")
Auto Create (Redshift)
AutoCreate Redshift property (Create a new database user if one does not exist)
Allowed DB Groups (Redshift)
Comma separated list of allowed user groups. Use "*" to allow all groups, "\," to escape comma in group name

The **DbUser Format**, **AutoCreate**, and **Allowed DBGroups** settings aren't used by Athena. You don't have to configure them.

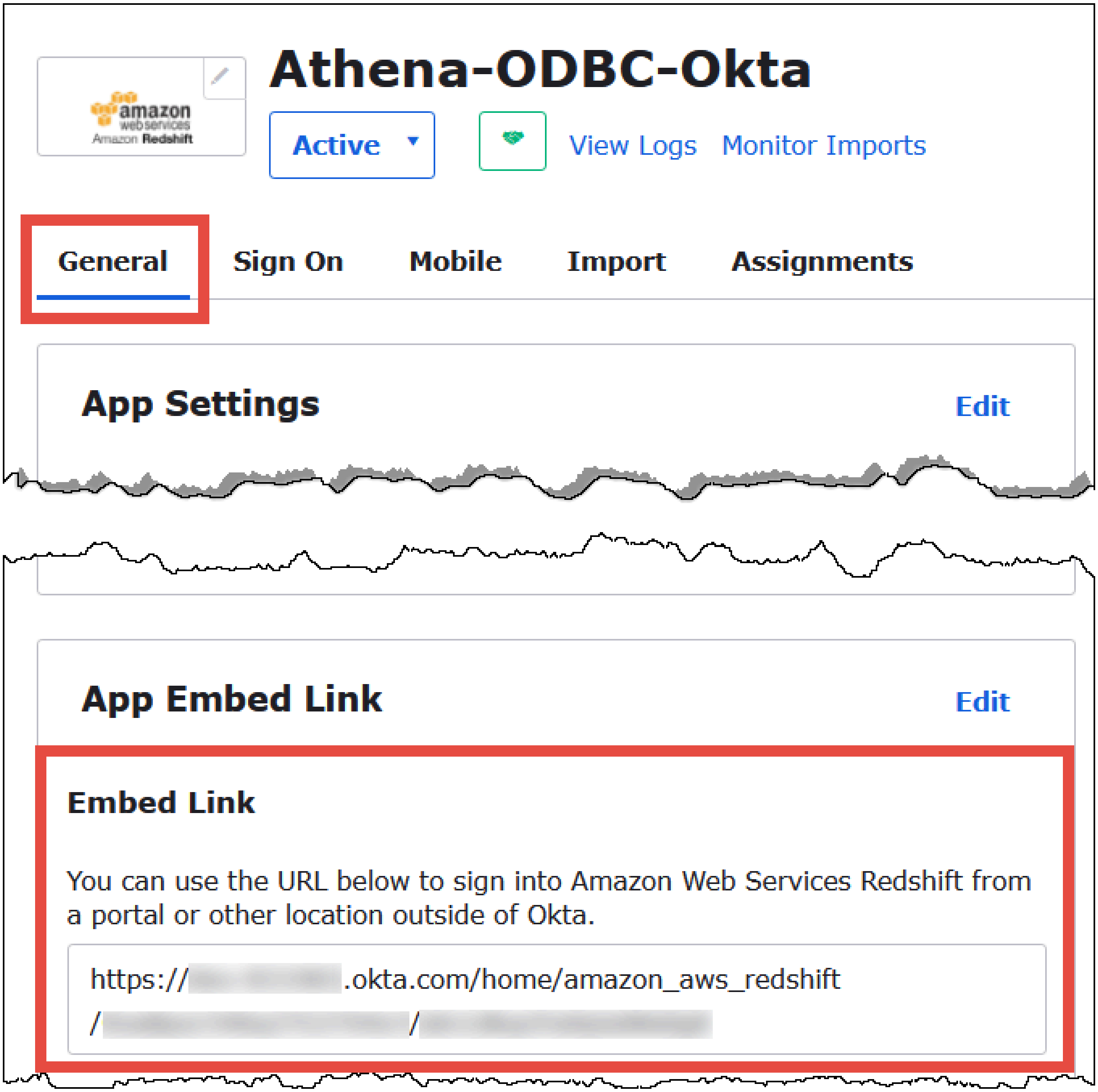
12. Choose **Save**.

Retrieve ODBC configuration information from Okta

Now that you created the Okta application, you're ready to retrieve the application's ID and IdP host URL. You will require these later when you configure ODBC for connection to Athena.

To retrieve configuration information for ODBC from Okta

1. Choose the **General** tab of your Okta application, and then scroll down to the **App Embed Link** section.



Your **Embed Link** URL is in the following format:

```
https://trial-1234567.okta.com/home/amazon_aws_redshift/Abc1de2fghi3J45kL678/
abc1defghij2klmNo3p4
```

2. From your **Embed Link** URL, extract and save the following pieces:

- The first segment after `https://`, up to and including `okta.com` (for example, **trial-1234567.okta.com**). This is your IdP host.
- The last two segments of the URL, including the forward slash in the middle. The segments are two 20-character strings with a mix of numbers and upper and lowercase letters (for example, **Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4**). This is your application ID.

Add a user to the Okta application

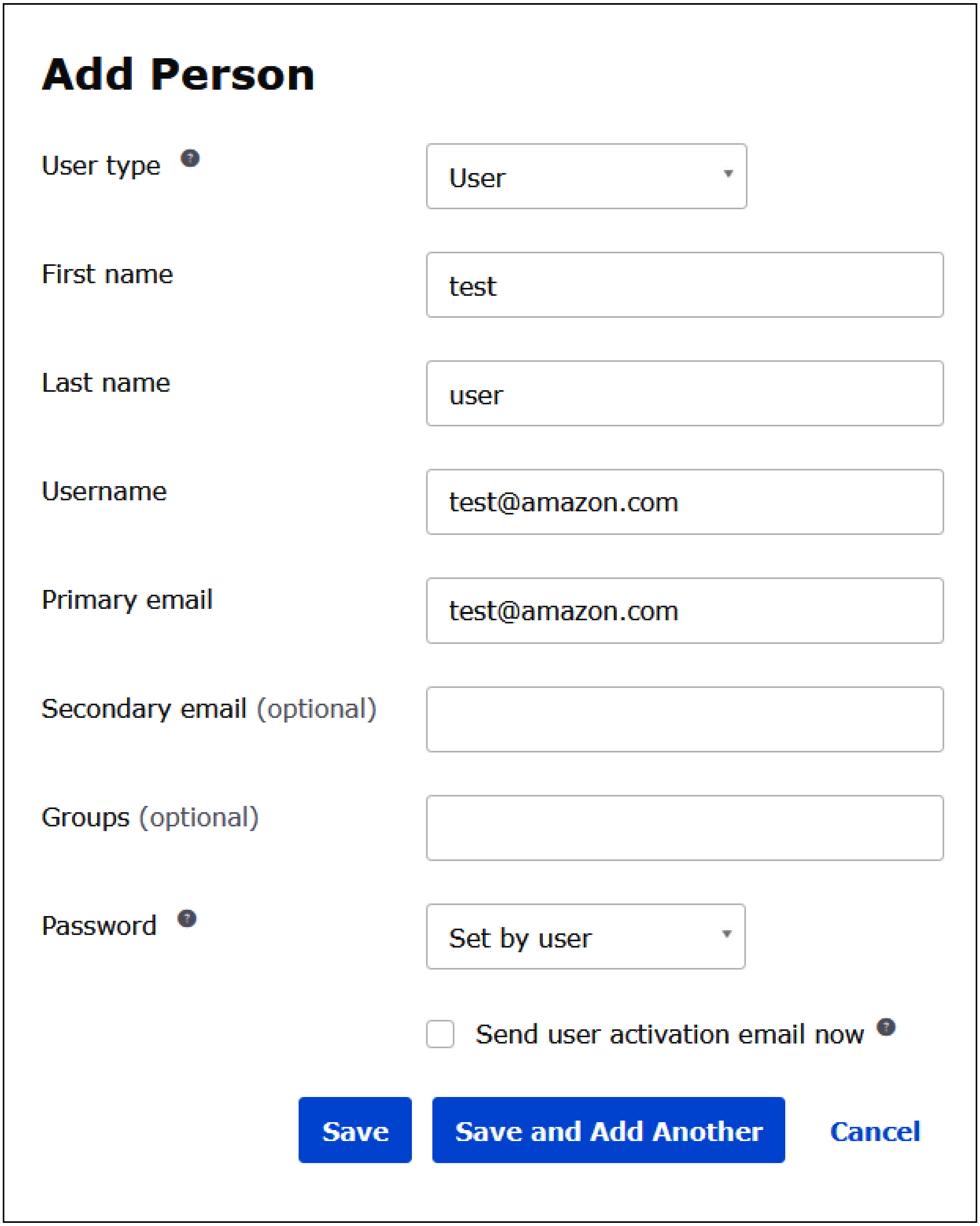
Now you're ready to add a user to your Okta application.

To add a user to the Okta application

1. In the left navigation pane, choose **Directory**, and then choose **People**.
2. Choose **Add person**.

3. In the **Add Person** dialog box, enter the following information.

   - Enter values for **First name** and **Last name**. This tutorial uses **`test user`**.
   - Enter values for **Username** and **Primary email**. This tutorial uses **`test@amazon.com`** for both. Your security requirements for passwords might vary.

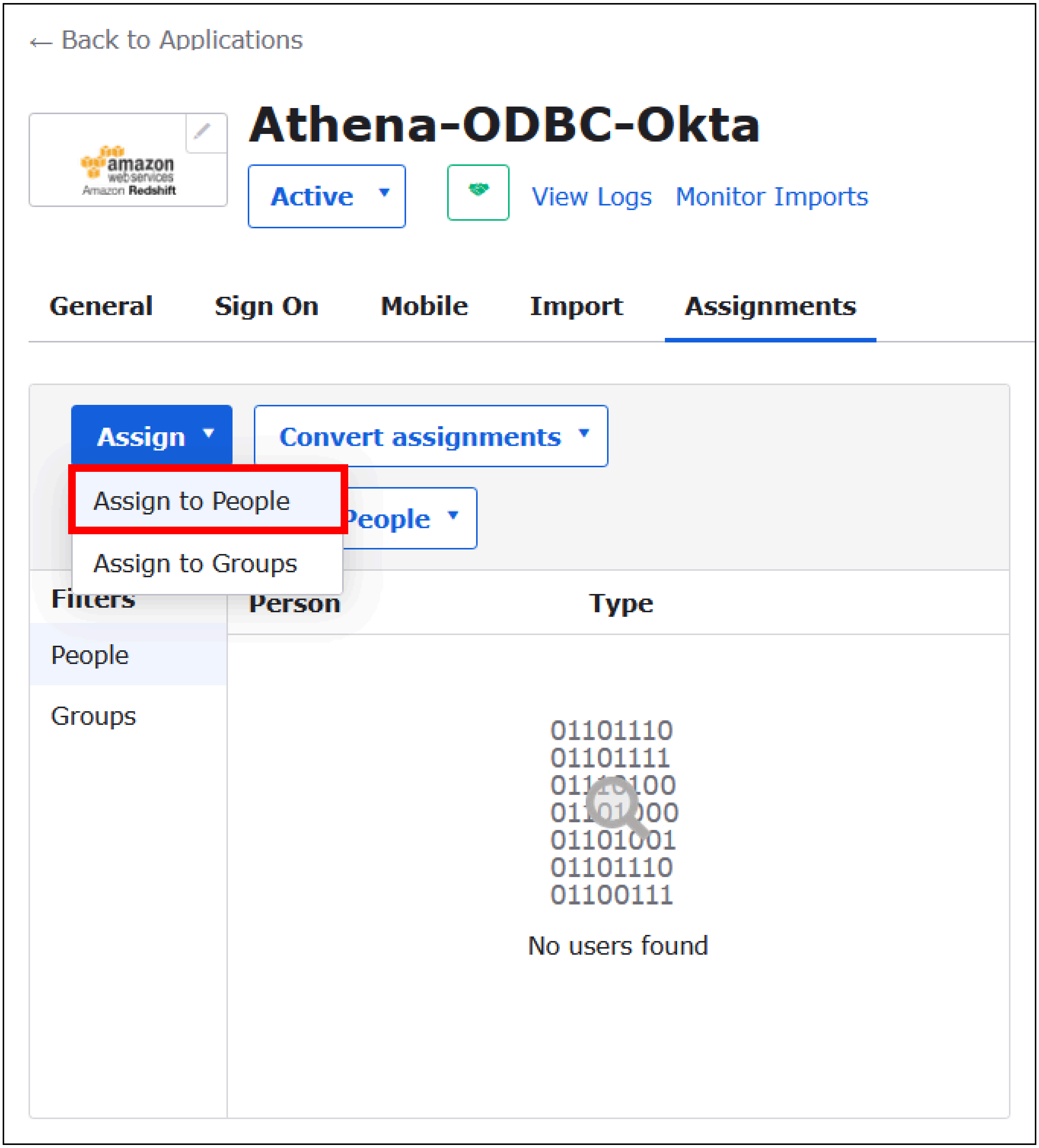


4. Choose **Save**.

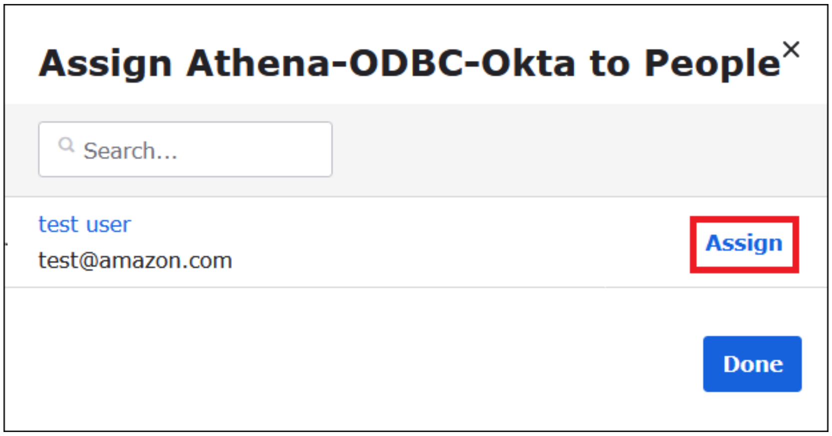
Now you're ready to assign the user that you created to your application.

To assign the user to your application:

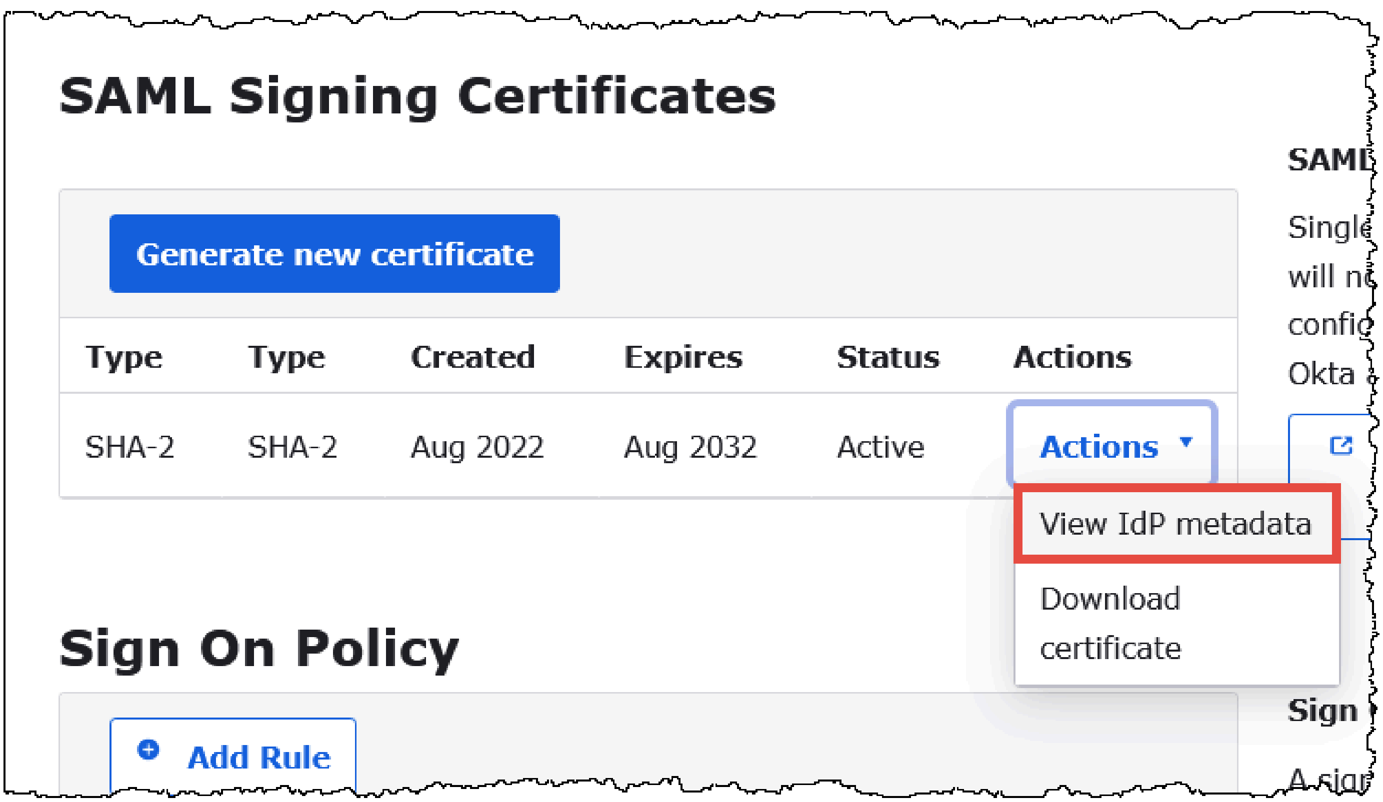
1. In the navigation pane, choose **Applications**, **Applications**, and then choose the name of your application (for example, **Athena-ODBC-Okta**).
2. Choose **Assign,** and then choose **Assign to People**.



3. Choose the **Assign** option for your user, and then choose **Done**.



4. At the prompt, choose **Save and Go Back**. The dialog box shows the user's status as **Assigned**.
5. Choose **Done**.
6. Choose the **Sign On** tab.
7. Scroll down to the **SAML Signing Certificates** section.
8. Choose **Actions**.
9. Open the context (right-click) menu for **View IdP metadata**, and then choose the browser option to save the file.
10. Save the file with an `.xml` extension.

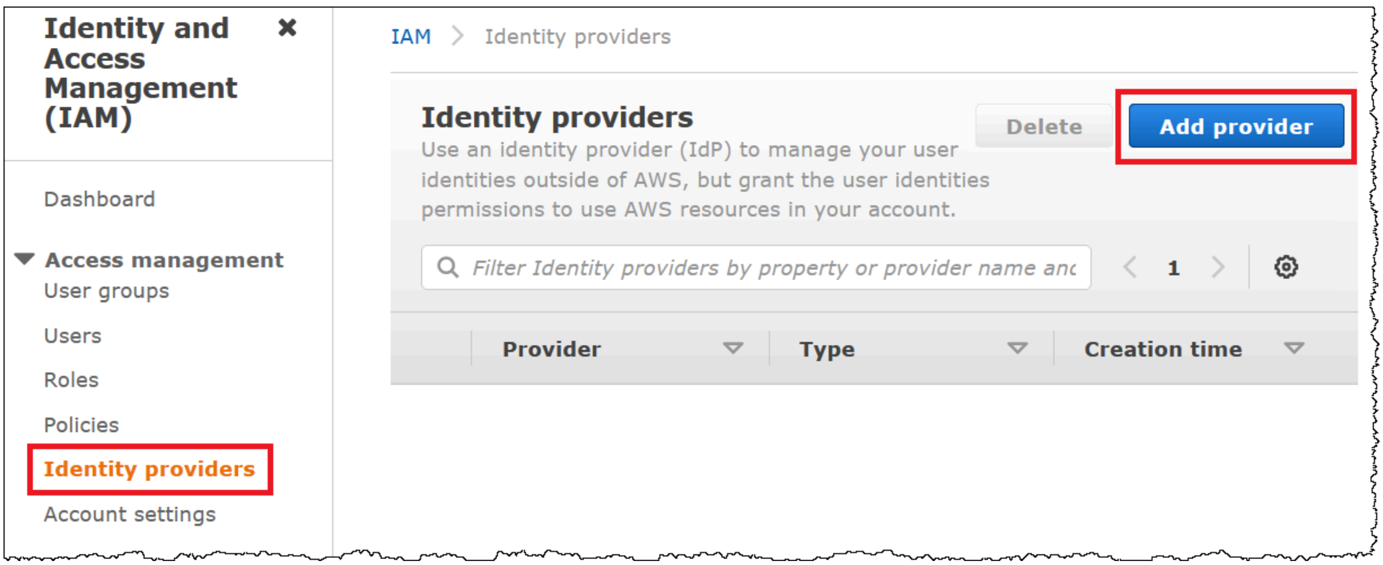


Create an AWS SAML Identity Provider and Role

Now you are ready to upload the metadata XML file to the IAM console in AWS. You will use this file to create an AWS SAML identity provider and role. Use an AWS Services administrator account to perform these steps.

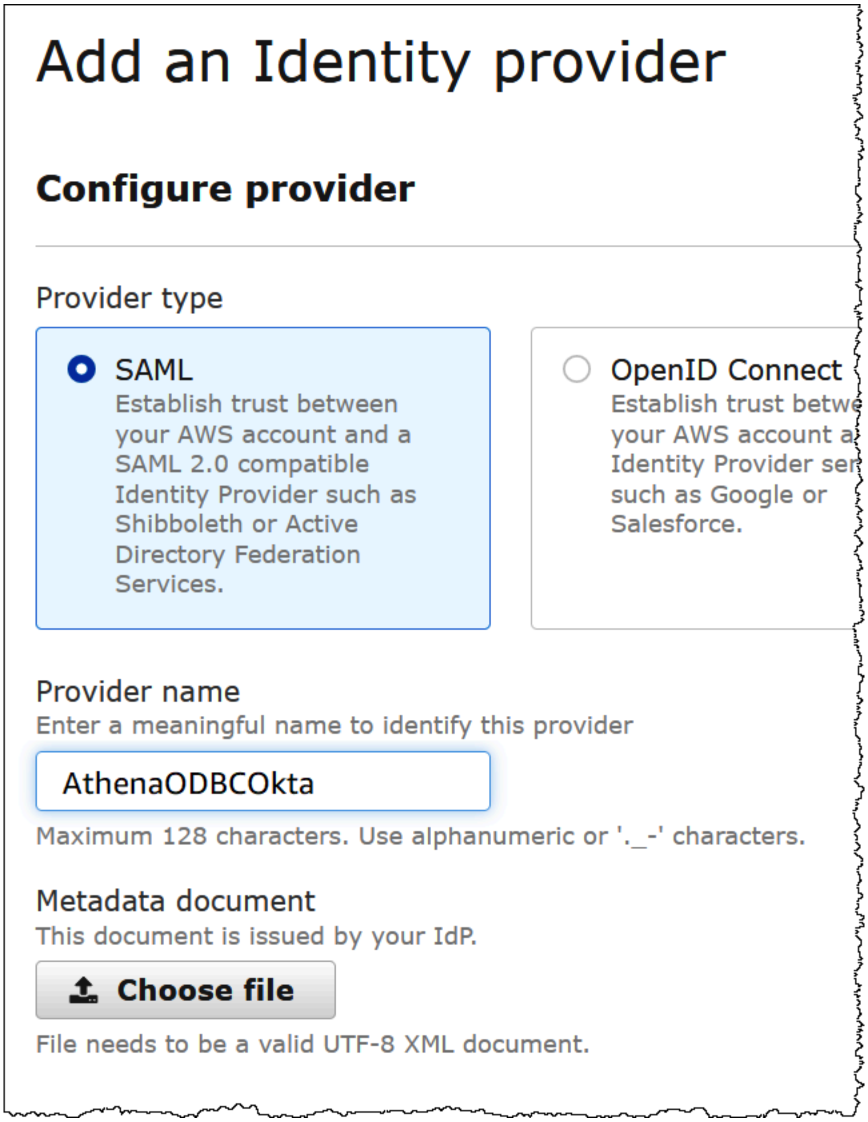
To create a SAML identity provider and role in AWS

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/IAM/.
2. In the navigation pane, choose **Identity providers**, and then choose **Add provider**.



3. On the **Add an Identity provider** page, for **Configure provider**, enter the following information.

   - For **Provider type**, choose **SAML**.
   - For **Provider name**, enter a name for your provider (for example, `AthenaODBCOkta`).
   - For **Metadata document**, use the **Choose file** option to upload the identity provider (IdP) metadata XML file that you downloaded.



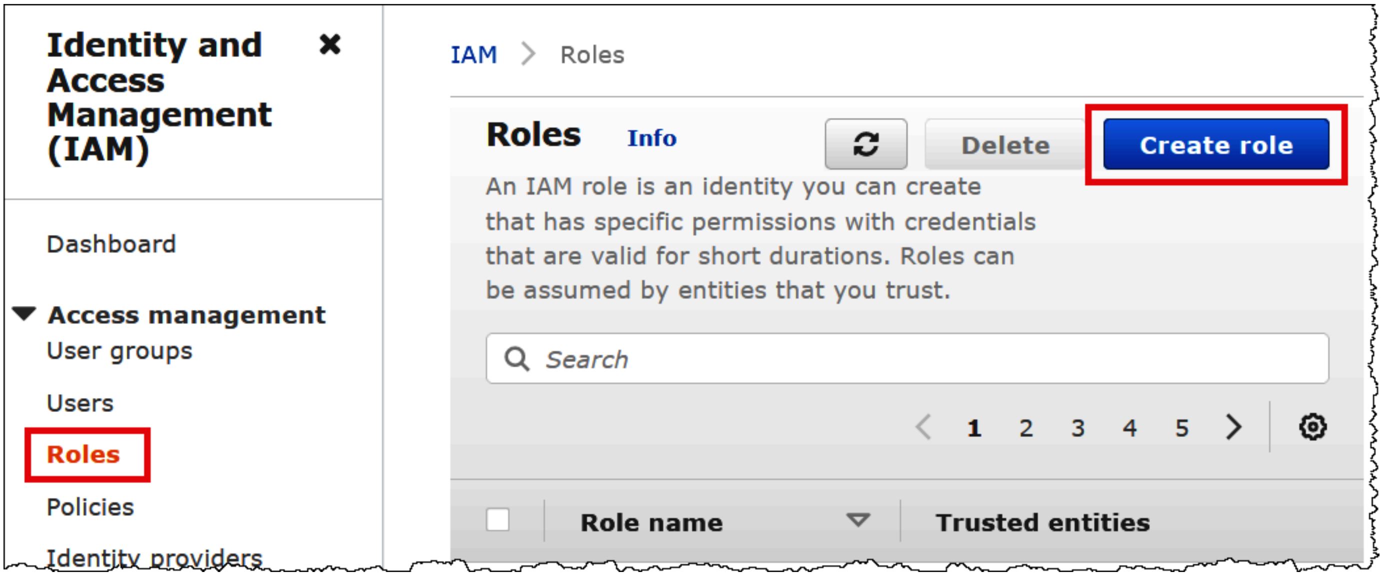
4. Choose **Add provider**.

Creating an IAM role for Athena and Amazon S3 access

Now you are ready to create an IAM role for Athena and Amazon S3 access. You will assign this role to your user. That way, you can provide the user with single sign-on access to Athena.

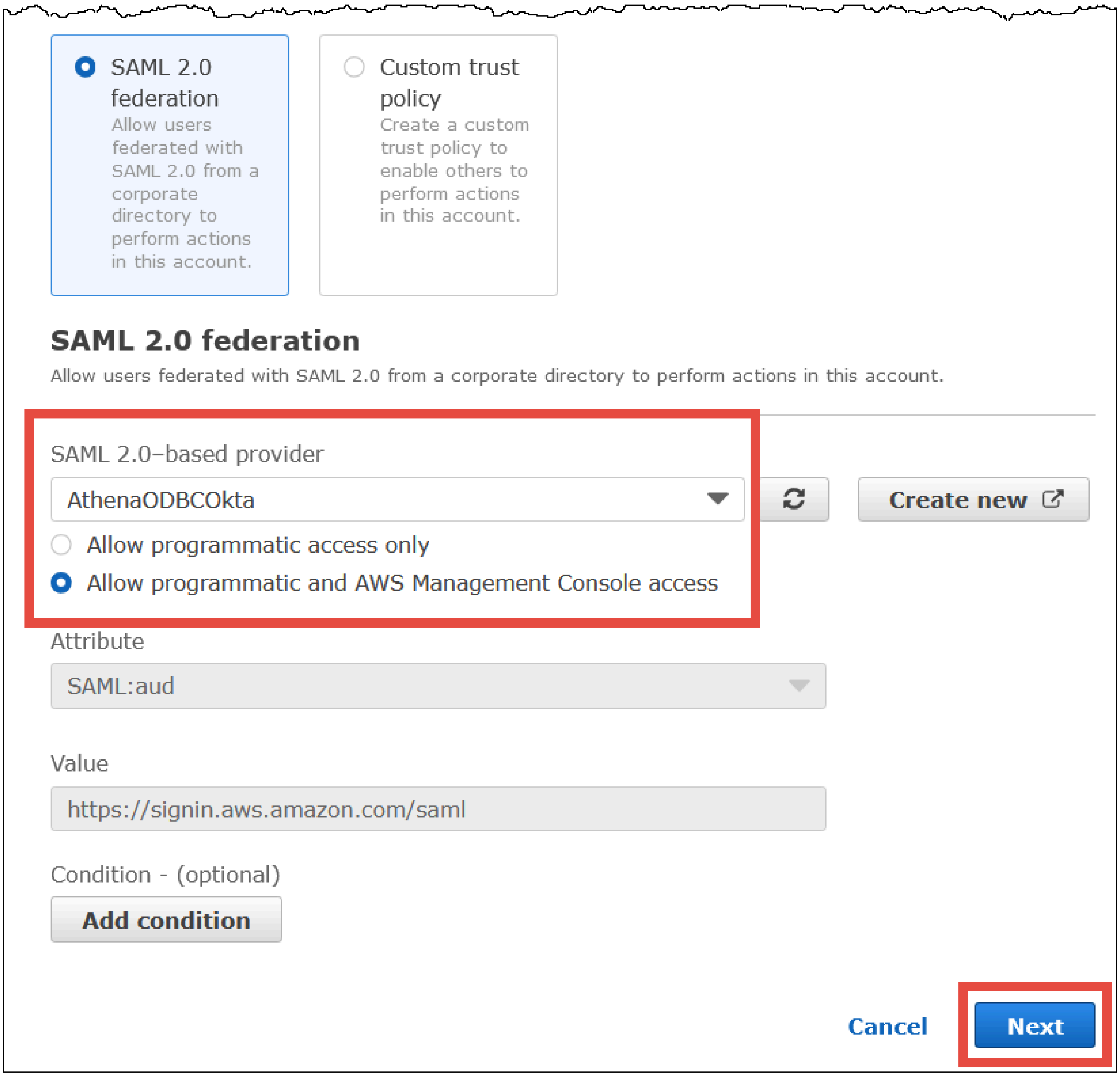
To create an IAM role for your user

1. In the IAM console navigation pane, choose **Roles**, and then choose **Create role**.



2. On the **Create role** page, choose the following options:

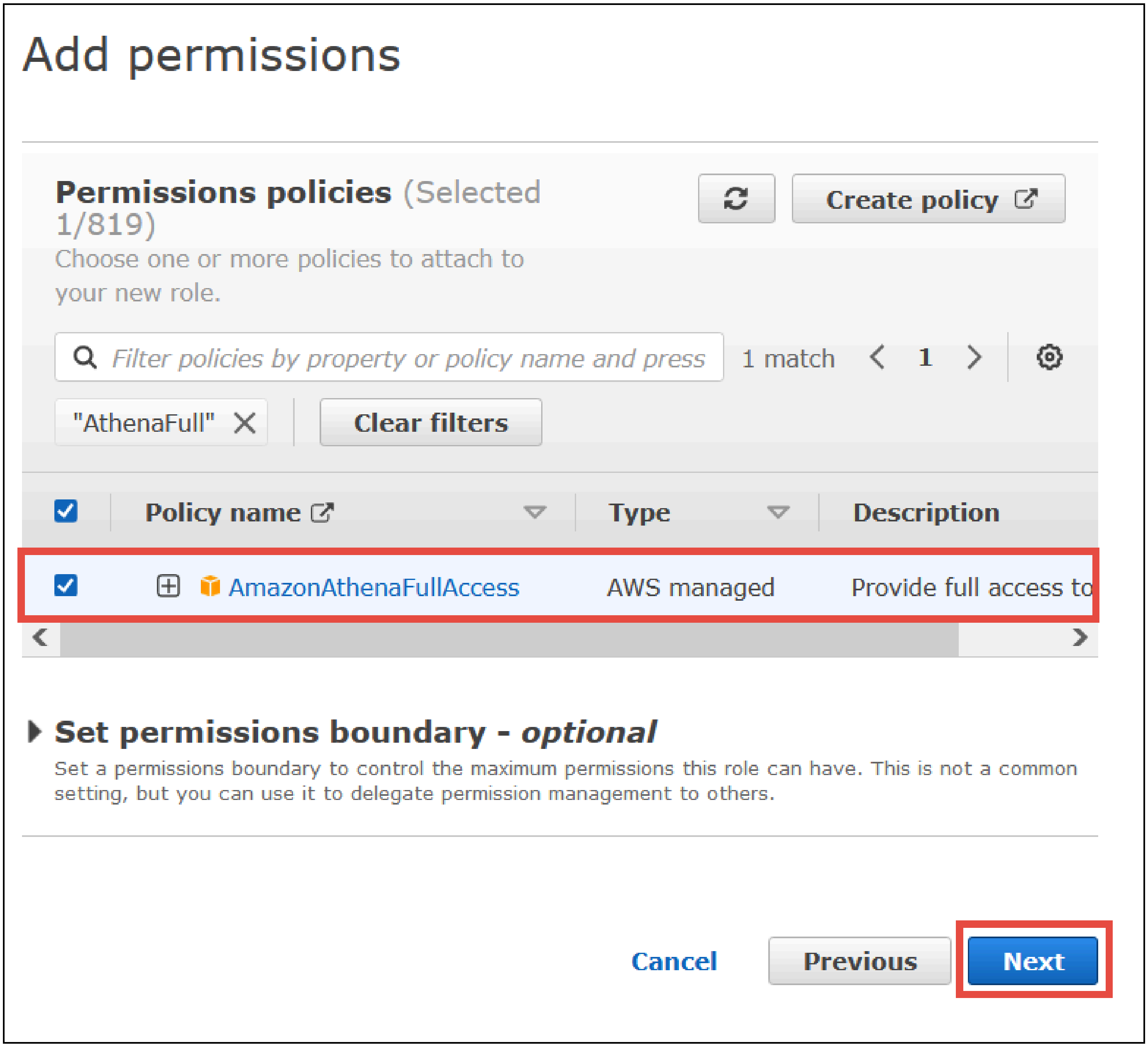
   - For **Select type of trusted entity**, choose **SAML 2.0 Federation.**
   - For **SAML 2.0–based provider**, choose the SAML identity provider that you created (for example, **AthenaODBCOkta**).
   - Select **Allow programmatic and AWS Management Console access**.



3. Choose **Next**.

4. On the **Add Permissions** page, for **Filter policies**, enter **`AthenaFull`**, and then press ENTER.

5. Select the `AmazonAthenaFullAccess` managed policy, and then choose **Next**.



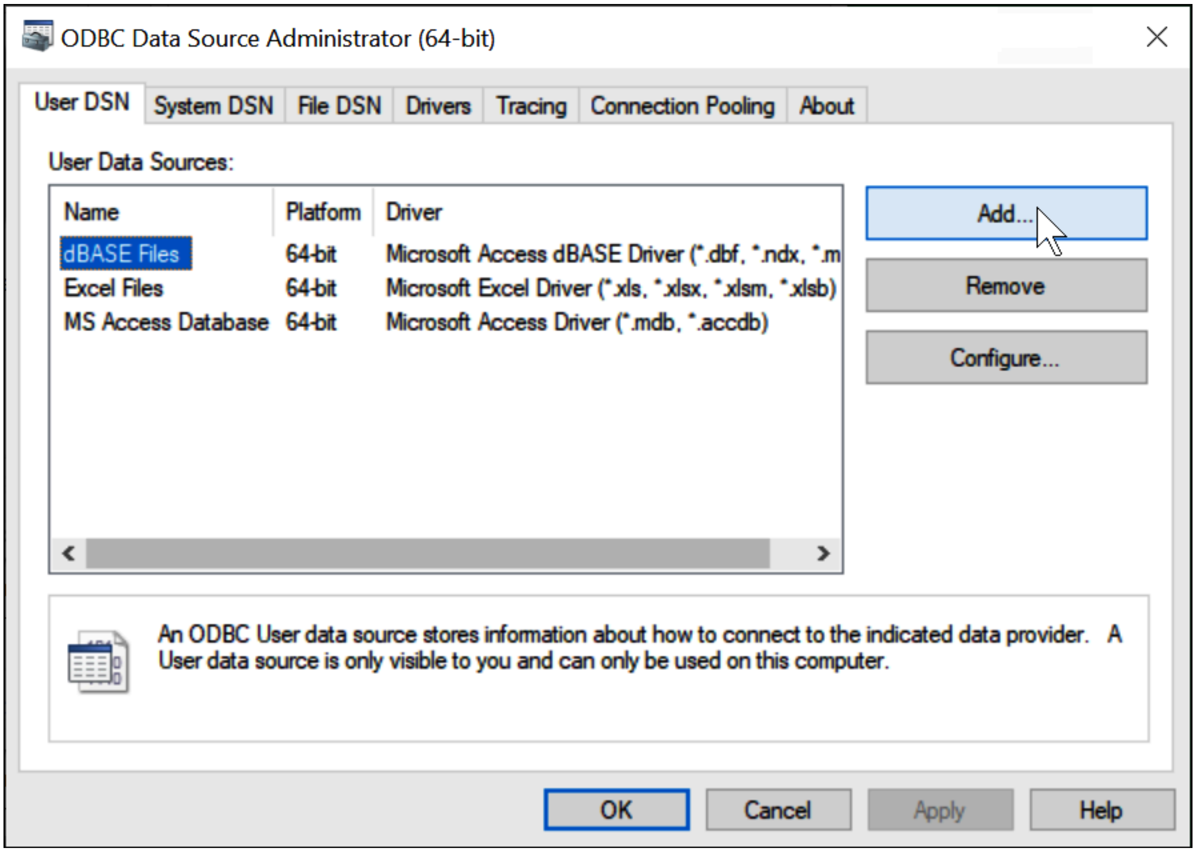
6. On the **Name, review, and create** page, for **Role name**, enter a name for the role (for example, **`Athena-ODBC-OktaRole`**), and then choose **Create role**.

Configuring the Okta ODBC connection to Athena

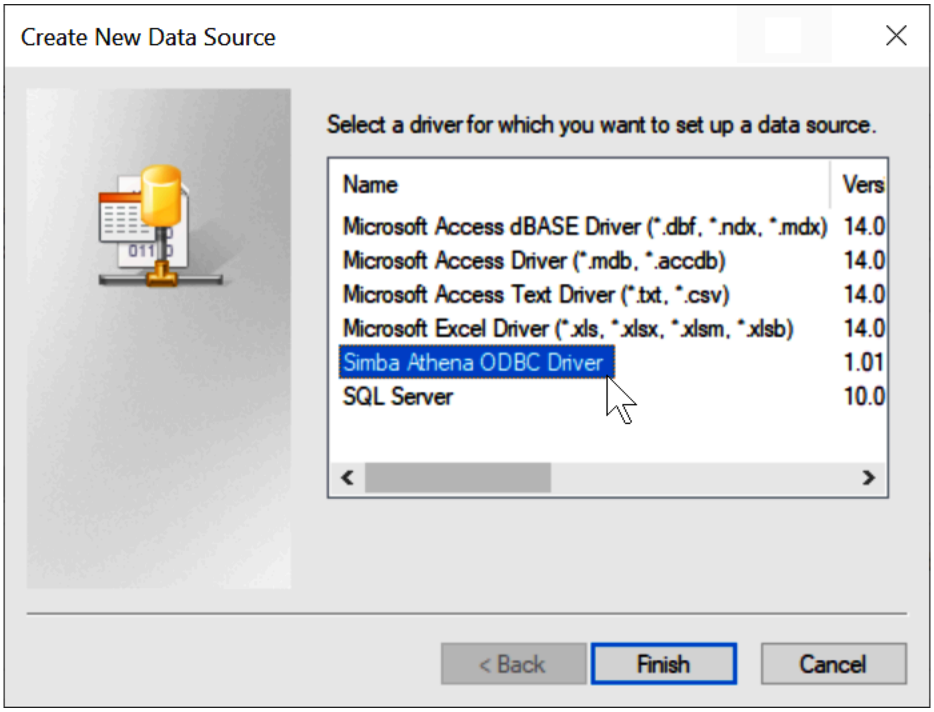
Now you're ready to configure the Okta ODBC connection to Athena using the ODBC Data Sources program in Windows.

To configure your Okta ODBC connection to Athena

1. In Windows, launch the **ODBC Data Sources** program.
2. In the **ODBC Data Source Administrator** program, choose **Add**.

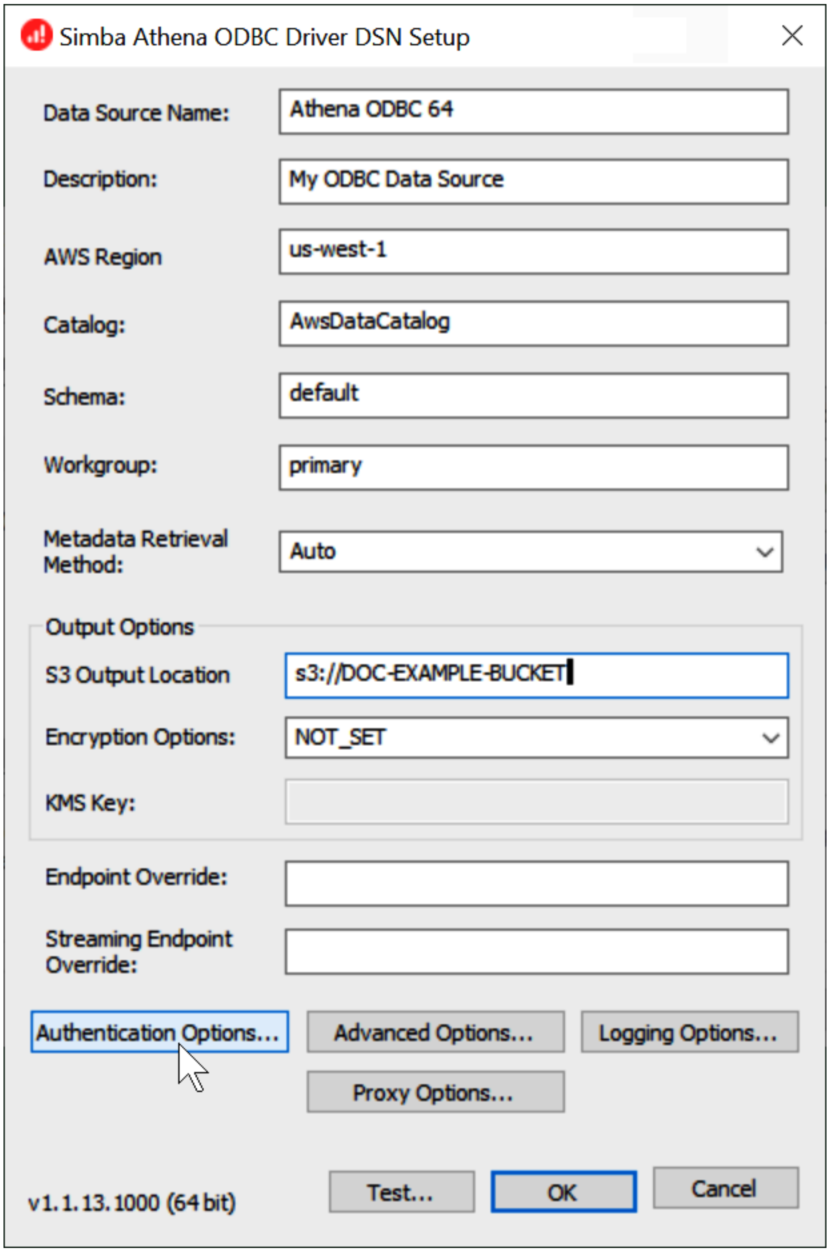


3. Choose **Simba Athena ODBC Driver**, and then choose **Finish**.



4. In the **Simba Athena ODBC Driver DSN Setup** dialog, enter the values described.

   - For **Data Source Name,** enter a name for your data source (for example, **`Athena ODBC 64`**).
   - For **Description**, enter a description for your data source.
   - For **AWS Region**, enter the AWS Region that you're using (for example, **`us-west-1`**).
   - For **S3 Output Location**, enter the Amazon S3 path where you want your output to be stored.

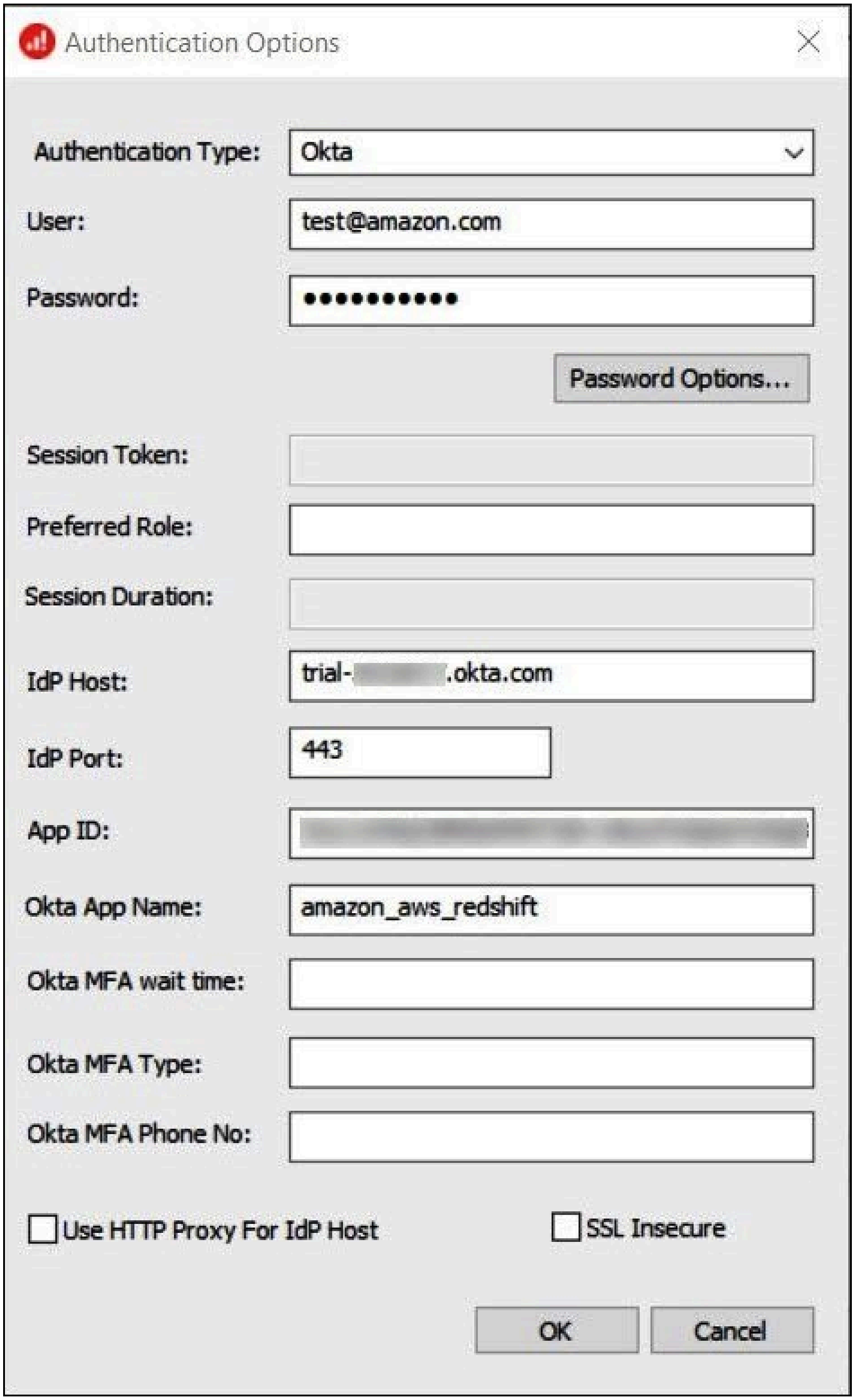


5. Choose **Authentication Options**.

6. In the **Authentication Options** dialog box, choose or enter the following values.

   - For **Authentication Type**, choose **Okta**.
   - For **User**, enter your Okta user name.
   - For **Password**, enter your Okta password.
   - For **IdP Host**, enter the value that you recorded earlier (for example, **`trial-1234567.okta.com`**).
   - For **IdP Port**, enter **443**.

- For **App ID**, enter the value that you recorded earlier (the last two segments of your Okta embed link).
- For **Okta App Name**, enter **`amazon_aws_redshift`**.



7. Choose **OK**.
8. Choose **Test** to test the connection or **OK** to finish.

Configure single sign-on using ODBC, SAML 2.0, and the Okta Identity Provider

To connect to data sources, you can use Amazon Athena with identity providers (IdPs) like PingOne, Okta, OneLogin, and others. Starting with Athena ODBC driver version 1.1.13 and Athena JDBC driver version 2.0.25, a browser SAML plugin is included that you can configure to work with any SAML 2.0 provider. This topic shows you how to configure the Amazon Athena ODBC driver and the browser-based SAML plugin to add single sign-on (SSO) capability using the Okta identity provider.

Prerequisites

Completing the steps in this tutorial requires the following:

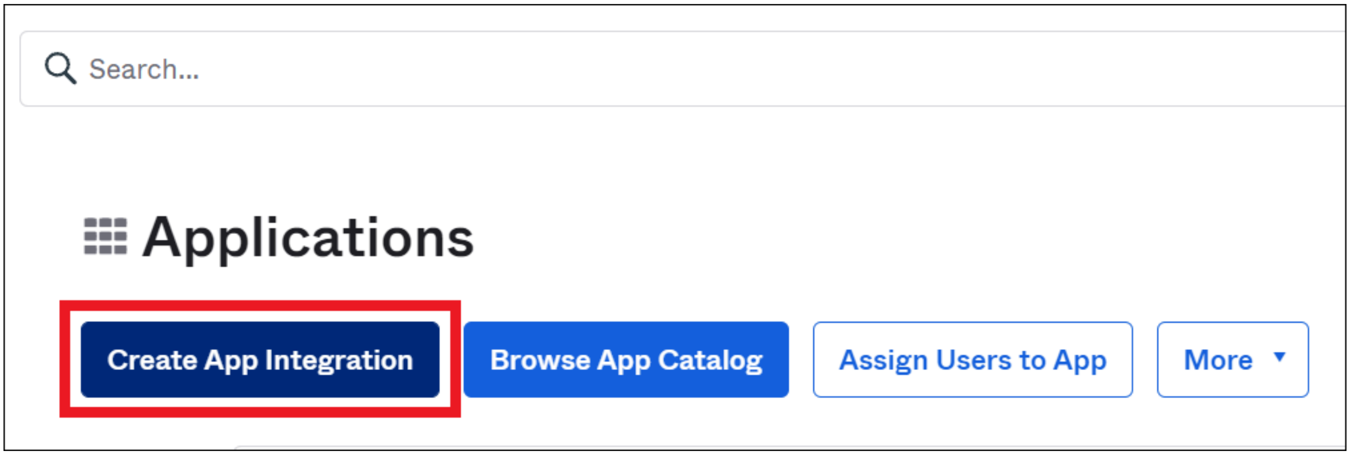
- Athena ODBC driver version 1.1.13 or later. Versions 1.1.13 and later include browser SAML support. For download links, see Connecting to Amazon Athena with ODBC.
- An IAM Role that you want to use with SAML. For more information, see Creating a role for SAML 2.0 federation in the *IAM User Guide*.
- An Okta account. For information, visit okta.com.

Creating an app integration in Okta

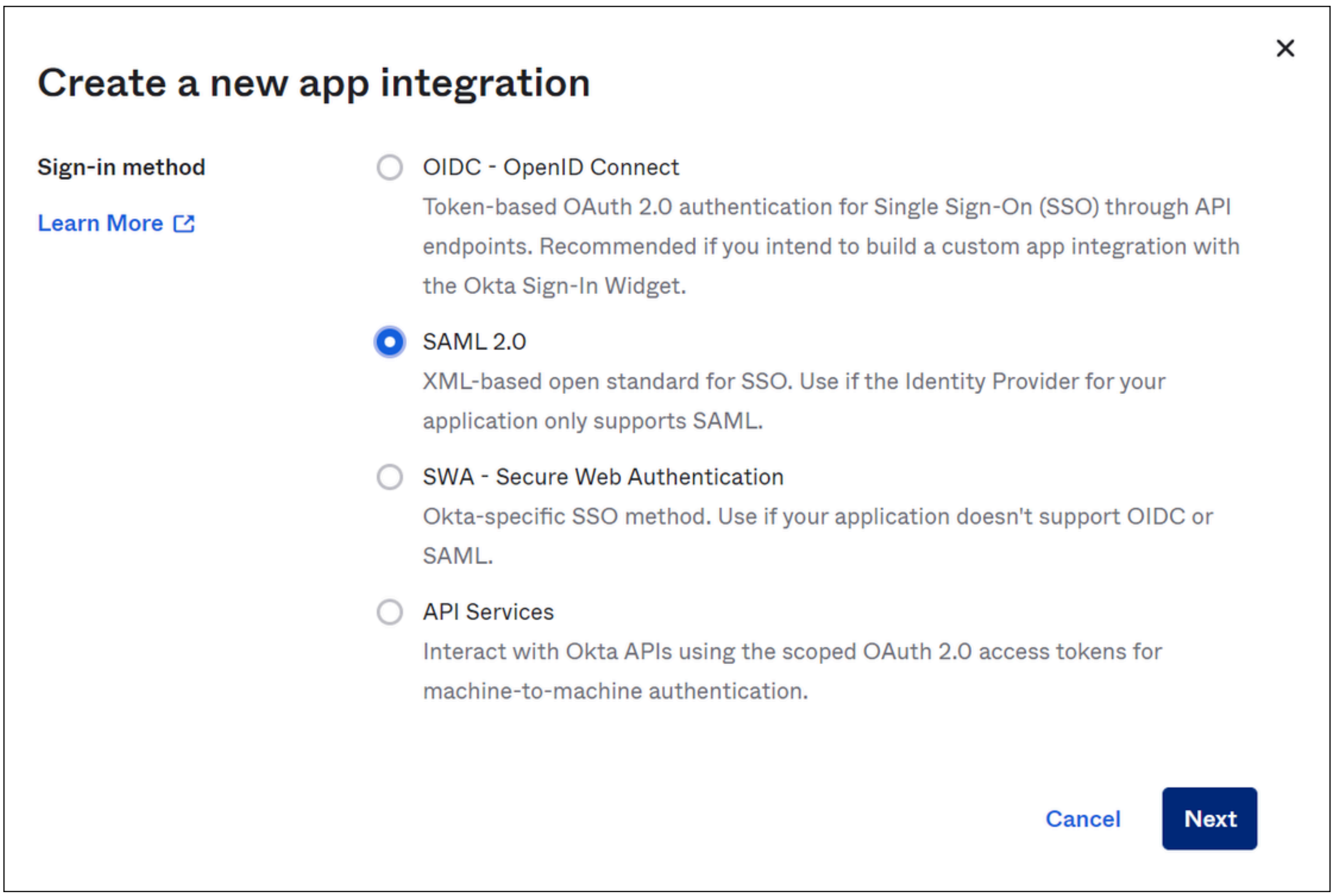
First, use the Okta dashboard to create and configure a SAML 2.0 app for single sign-on to Athena.

To use the Okta dashboard to set up single sign-on for Athena

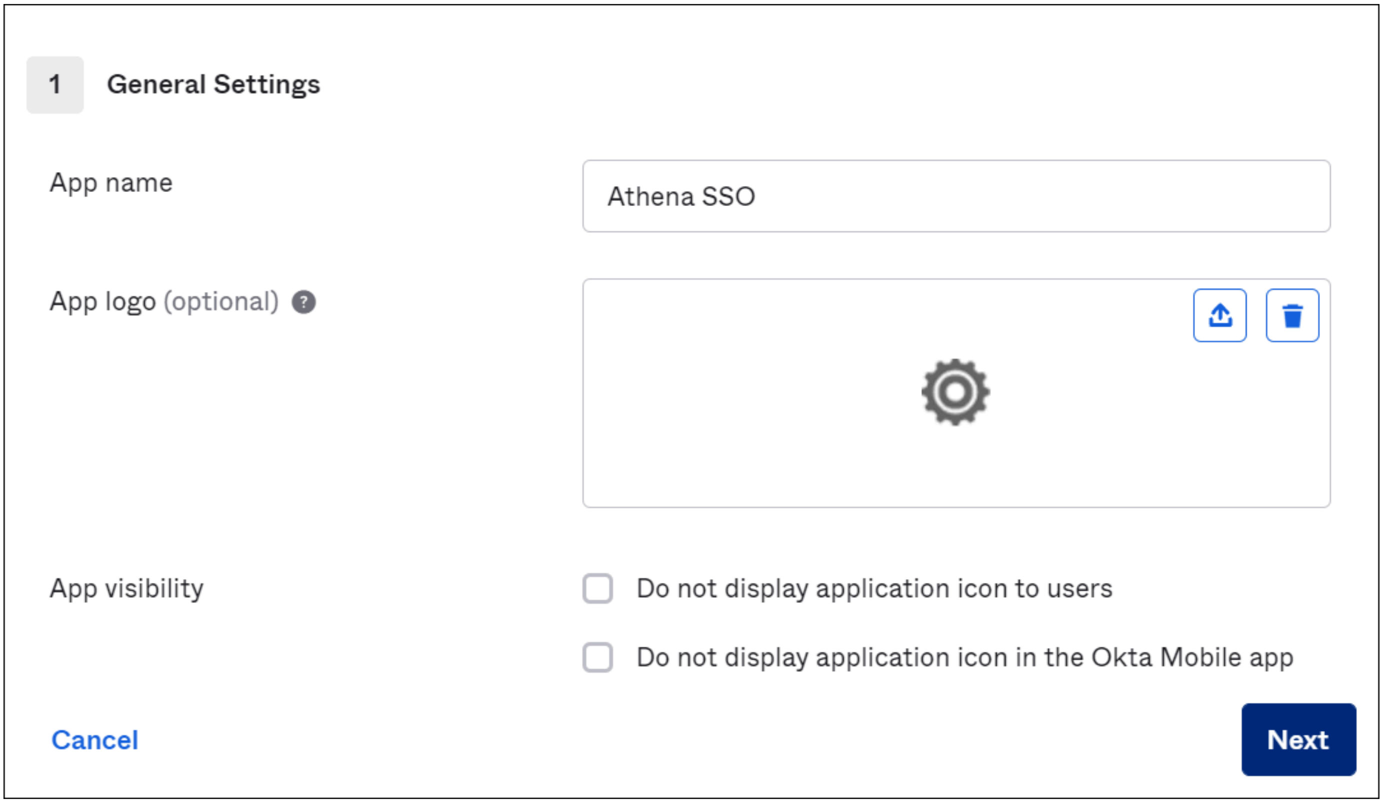
1. Login to the Okta admin page on `okta.com`.
2. In the navigation pane, choose **Applications**, **Applications**.
3. On the **Applications** page, choose **Create App Integration**.



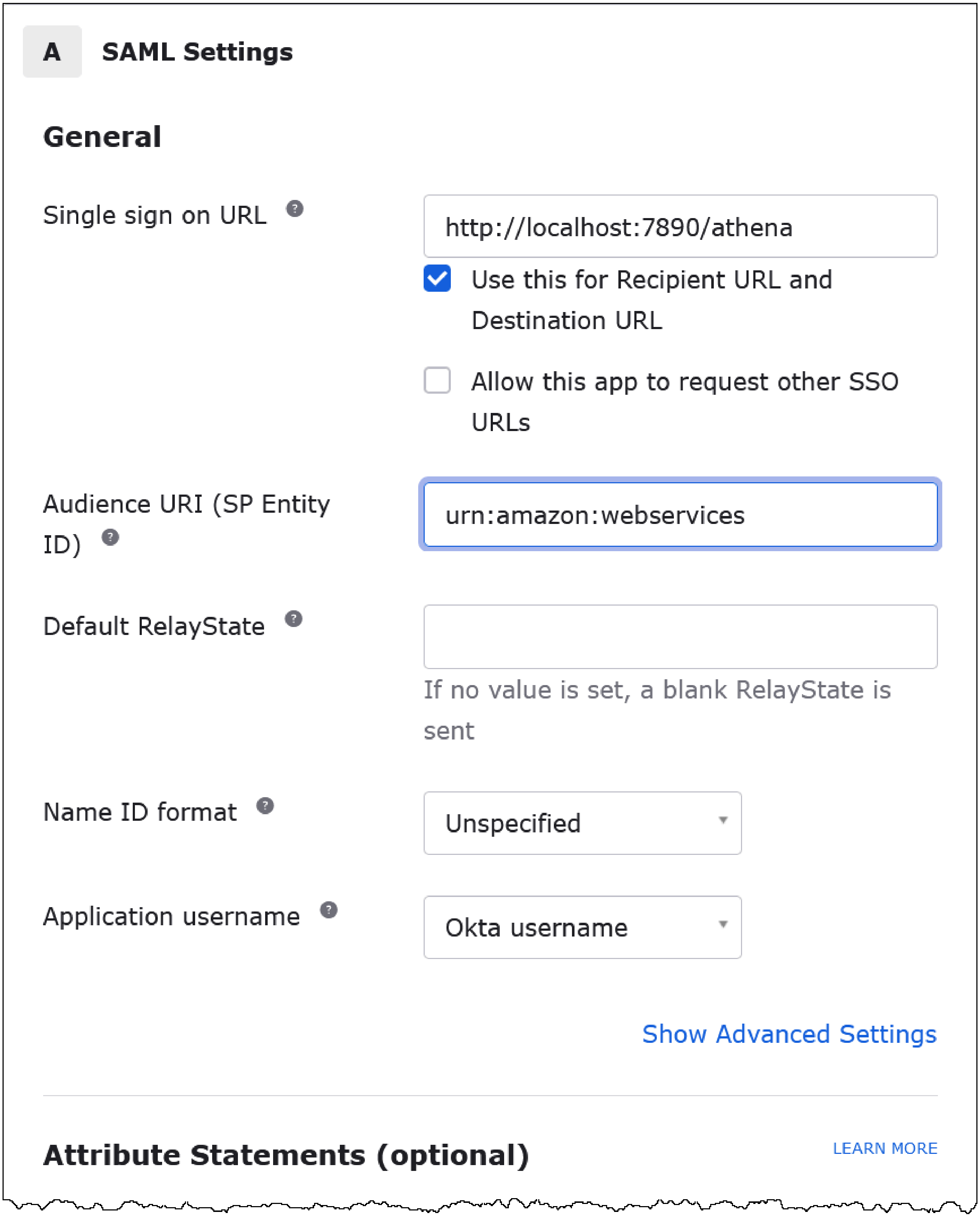
4. In the **Create a new app integration** dialog box, for **Sign-in method**, select **SAML 2.0**, and then choose **Next**.



5. On the **Create SAML Integration** page, in the **General Settings** section, enter a name for the application. This tutorial uses the name **Athena SSO**.



6. Choose **Next**.
7. On the **Configure SAML** page, in the **SAML Settings** section, enter the following values:
   - For **Single sign on URL**, enter **`http://localhost:7890/athena`**
   - For **Audience URI**, enter **`urn:amazon:webservices`**


A SAML Settings
General
Single sign on URL
http://localhost:7890/athena
Use this for Recipient URL and Destination URL
Allow this app to request other SSO URLs
Audience URI (SP Entity ID)
urn:amazon:webservices
Default RelayState
If no value is set, a blank RelayState is sent
Name ID format
Unspecified
Application username
Okta username
Show Advanced Settings
Attribute Statements (optional)
LEARN MORE

8. For **Attribute Statements (optional)**, enter the following two name/value pairs. These are required mapping attributes.

   - For **Name**, enter the following URL:

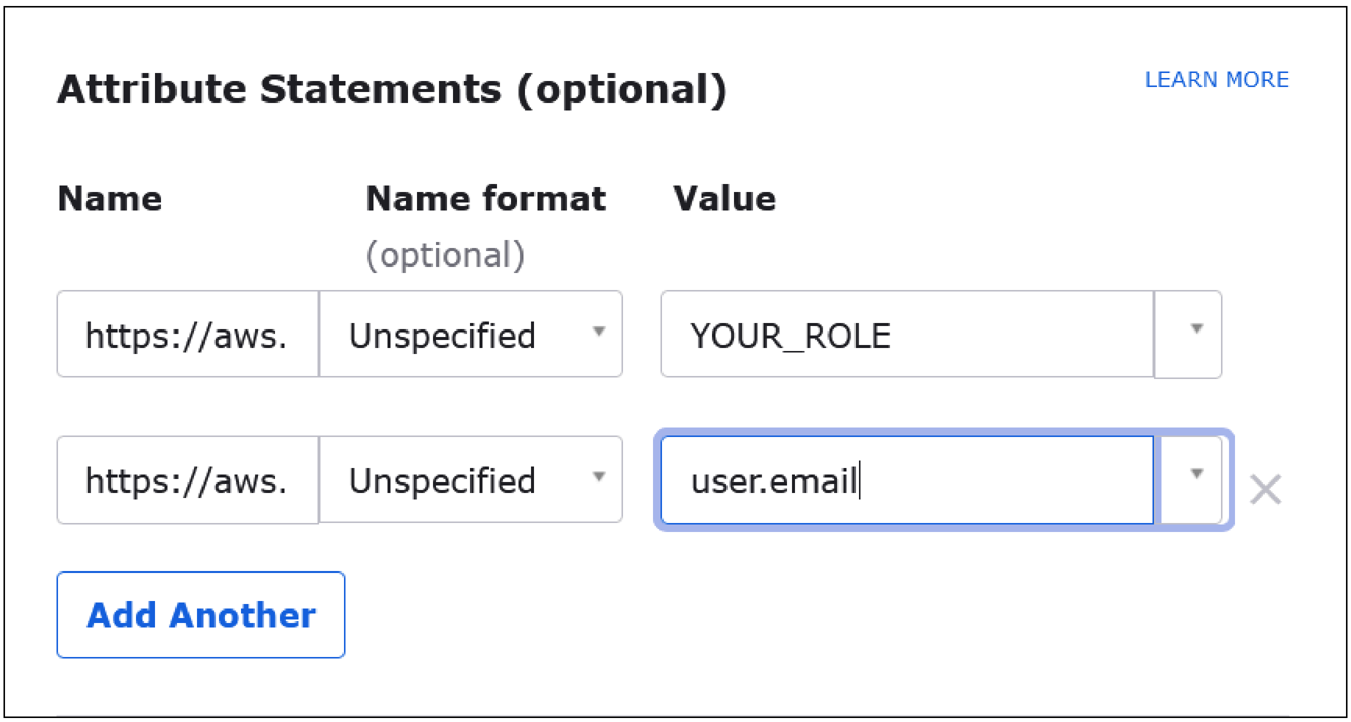
     ```
     https://aws.amazon.com/SAML/Attributes/Role
     ```

     For **Value**, enter the name of your IAM role. For information about the IAM role format, see [Configuring SAML assertions for the authentication response](#) in the *IAM User Guide*.

   - For **Name**, enter the following URL:

     ```
     https://aws.amazon.com/SAML/Attributes/RoleSessionName
     ```

     For **Value**, enter **user.email**.



9. Choose **Next**, and then choose **Finish**.

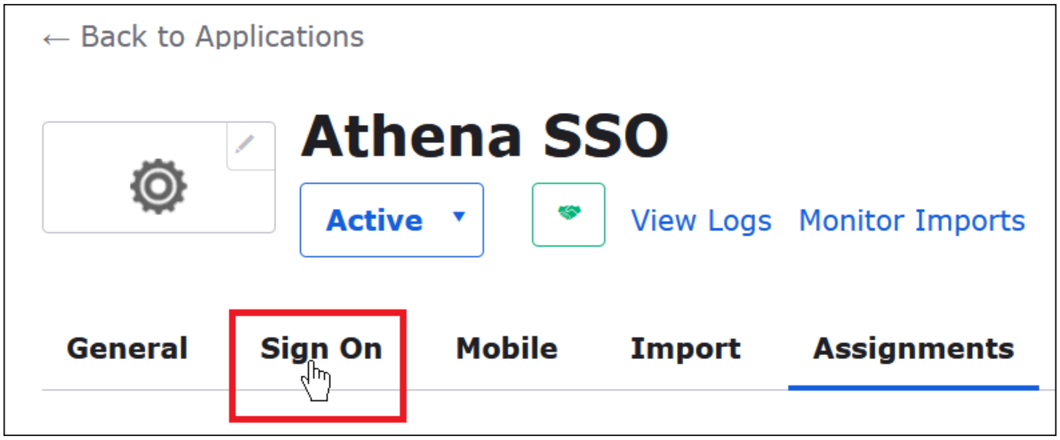
   When Okta creates the application, it also creates your login URL, which you will retrieve next.

Getting the login URL from the Okta dashboard

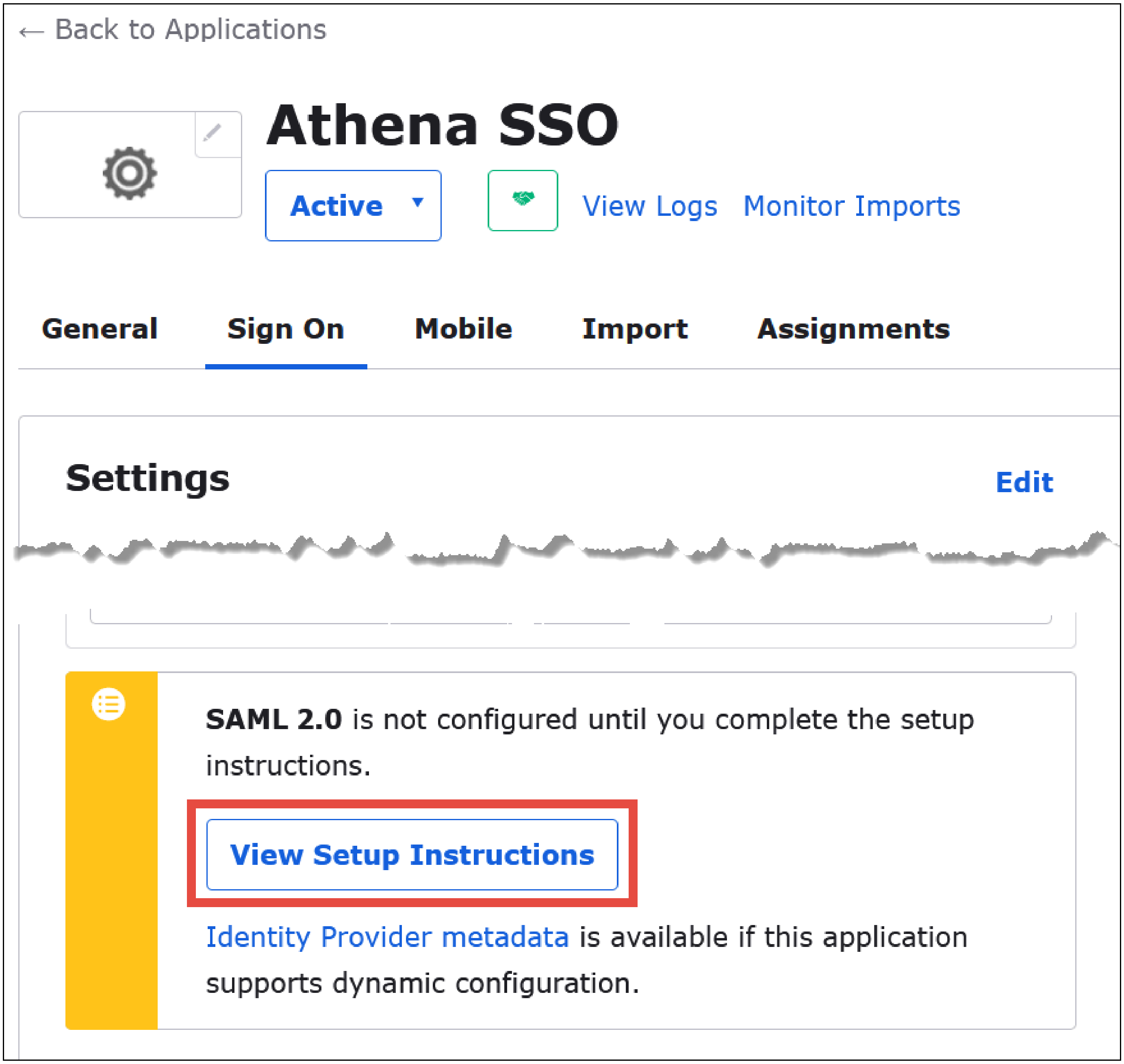
Now that your application has been created, you can obtain its login URL and other metadata from the Okta dashboard.

To get the login URL from the Okta dashboard

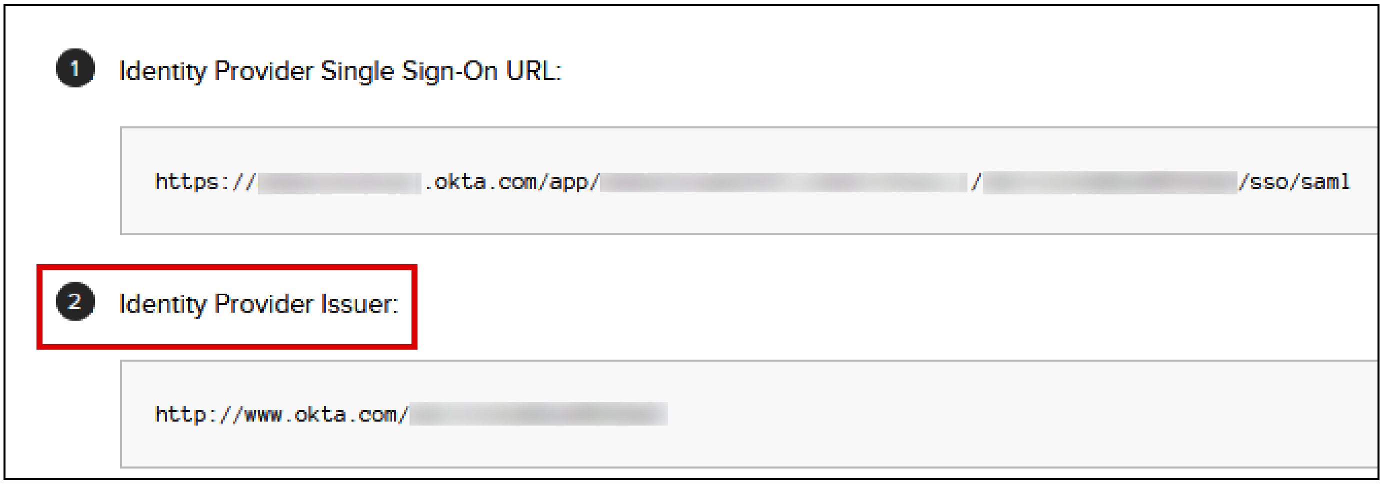
1. In the Okta navigation pane, choose **Applications**, **Applications**.
2. Choose the application for which you want to find the login URL (for example, **AthenaSSO**).
3. On the page for your application, choose **Sign On**.



4. Choose **View Setup Instructions**.



5. On the **How to Configure SAML 2.0 for Athena SSO** page, find the URL for **Identity Provider Issuer**. Some places in the Okta dashboard refer to this URL as the **SAML issuer ID**.



6. Copy or store the value for **Identity Provider Single Sign-On URL**.

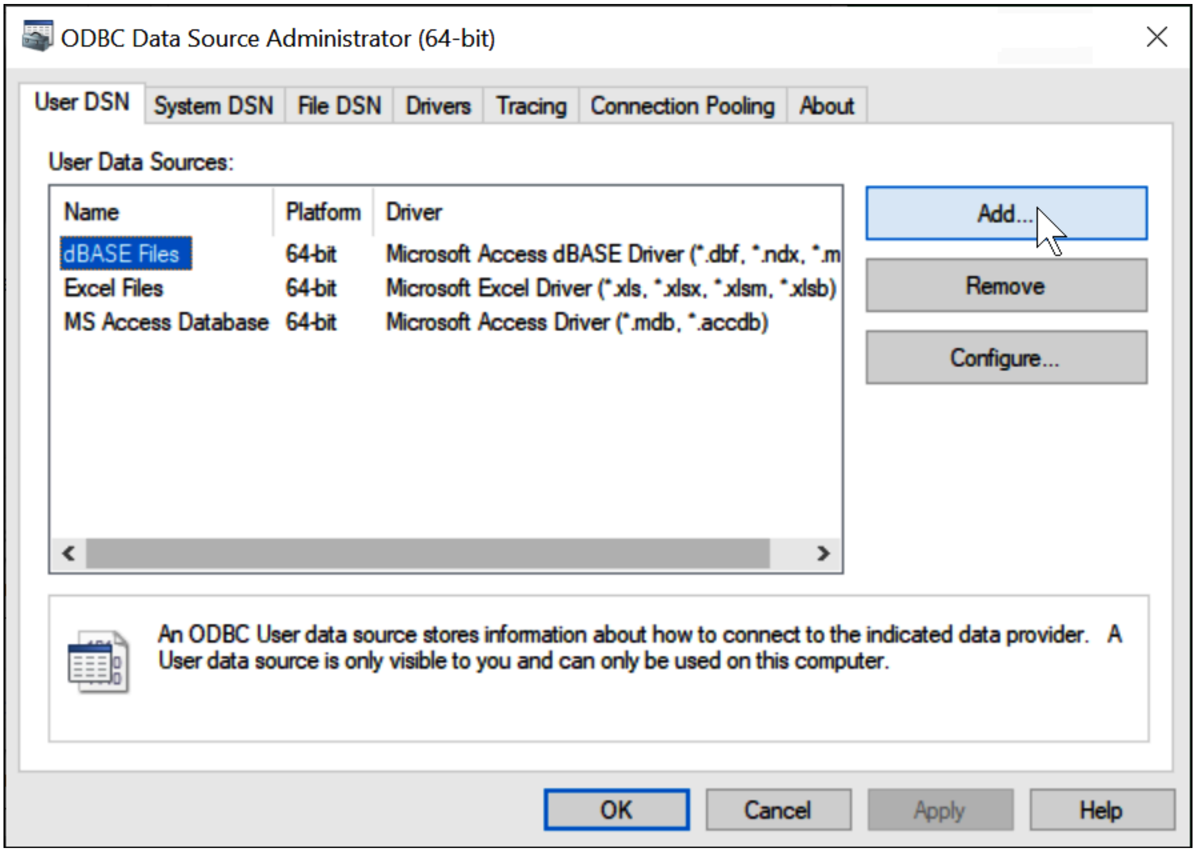
   In the next section, when you configure the ODBC connection, you will provide this value as the **Login URL** connection parameter for the browser SAML plugin.

Configuring the browser SAML ODBC connection to Athena

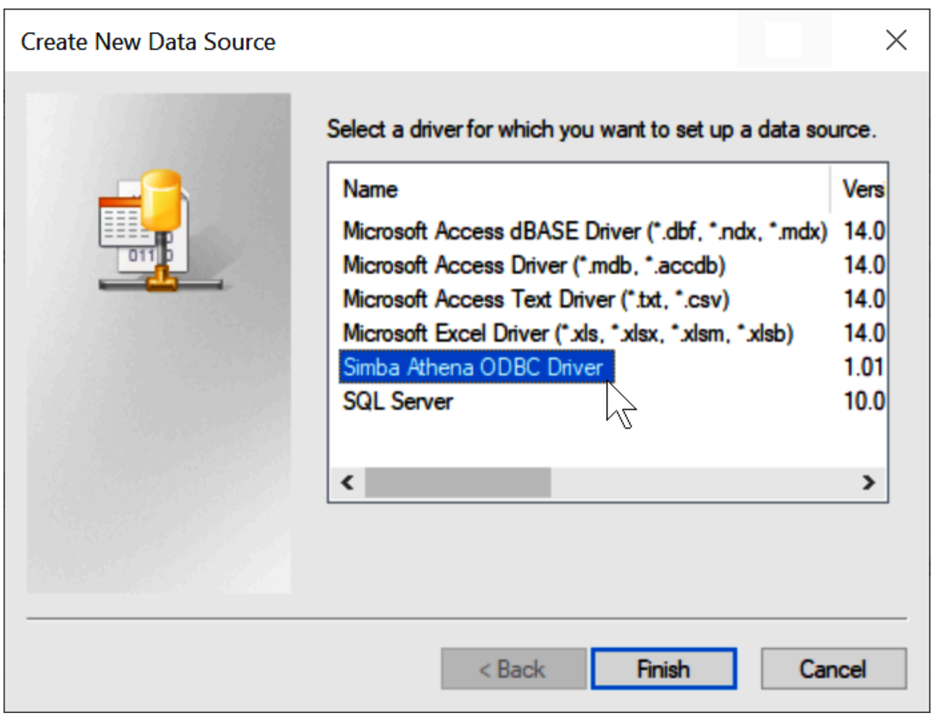
Now you are ready to configure the browser SAML connection to Athena using the ODBC Data Sources program in Windows.

To configure the browser SAML ODBC connection to Athena

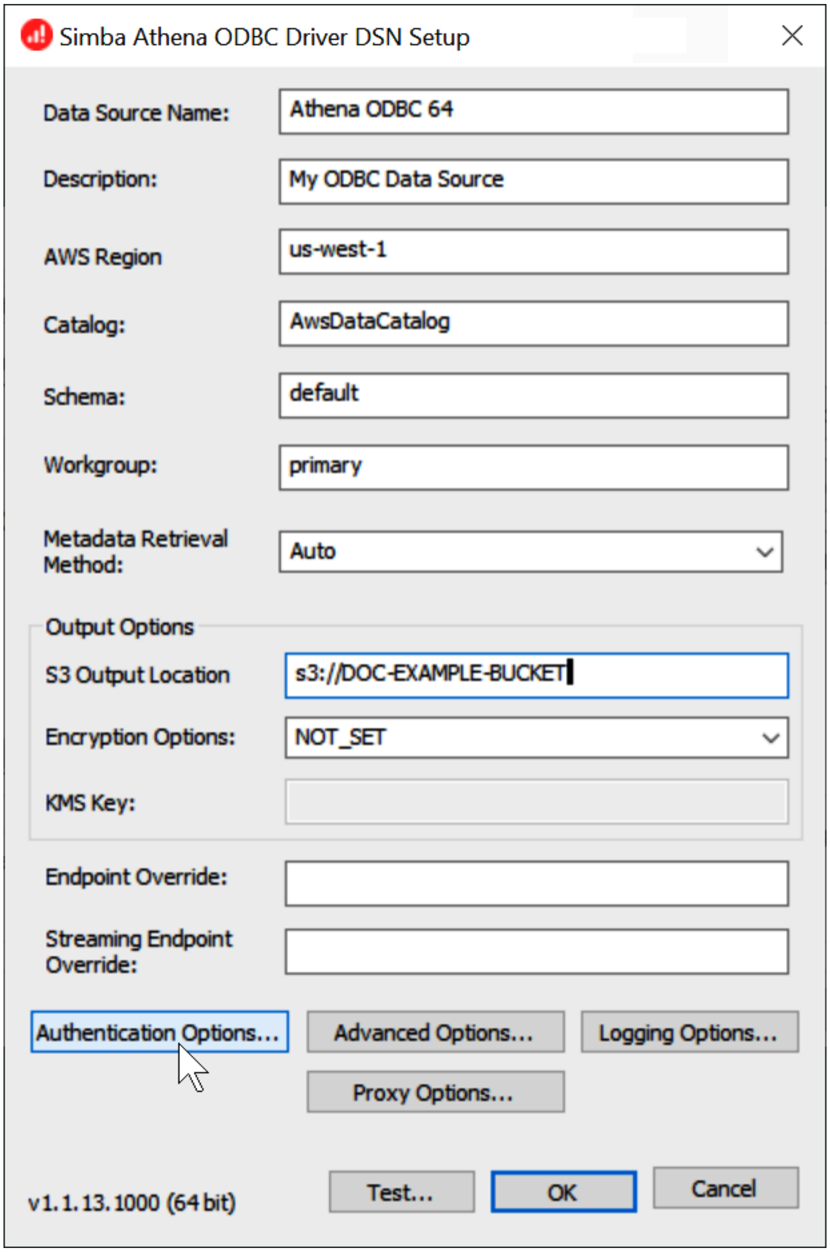
1. In Windows, launch the **ODBC Data Sources** program.
2. In the **ODBC Data Source Administrator** program, choose **Add**.



3. Choose **Simba Athena ODBC Driver**, and then choose **Finish**.

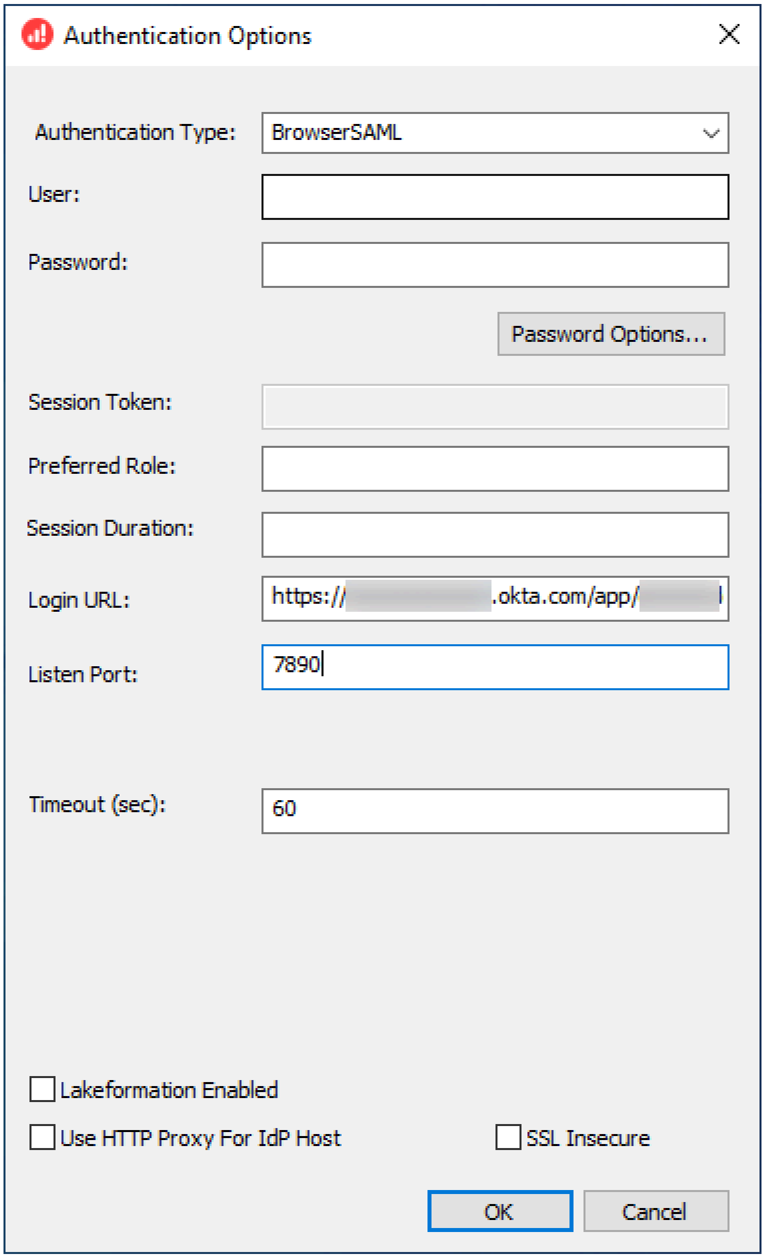


4. In the **Simba Athena ODBC Driver DSN Setup** dialog, enter the values described.



- For **Data Source Name,** enter a name for your data source (for example, **Athena ODBC 64**).
- For **Description**, enter a description for your data source.
- For **AWS Region**, enter the AWS Region that you are using (for example, **`us-west-1`**).
- For **S3 Output Location**, enter the Amazon S3 path where you want your output to be stored.

5. Choose **Authentication Options**.
6. In the **Authentication Options** dialog box, choose or enter the following values.



- For **Authentication Type**, choose **BrowserSAML**.
- For **Login URL**, enter the **Identity Provider Single Sign-On URL** that you obtained from the Okta dashboard.
- For **Listen Port**, enter **7890**.
- For **Timeout (sec)**, enter a connection timeout value in seconds.

7. Choose **OK** to close **Authentication Options**.
8. Choose **Test** to test the connection, or **OK** to finish.

Use the Amazon Athena Power BI connector

On Windows operating systems, you can use the Microsoft Power BI connector for Amazon Athena to analyze data from Amazon Athena in Microsoft Power BI Desktop. For information about Power BI, see Microsoft power BI. After you publish content to the Power BI service, you can use the July 2021 or later release of Power BI gateway to keep the content up to date through on-demand or scheduled refreshes.

Prerequisites

Before you begin, make sure that your environment meets the following requirements. The Amazon Athena ODBC driver is required.

- AWS account
- Permissions to use Athena
- Amazon Athena ODBC driver
- Power BI desktop

Capabilities supported

- **Import** – Selected tables and columns are imported into Power BI Desktop for querying.
- **DirectQuery** – No data is imported or copied into Power BI Desktop. Power BI Desktop queries the underlying data source directly.
- **Power BI gateway** – An on-premises data gateway in your AWS account that works like a bridge between the Microsoft Power BI Service and Athena. The gateway is required to see your data on the Microsoft Power BI Service.

Connect to Amazon Athena

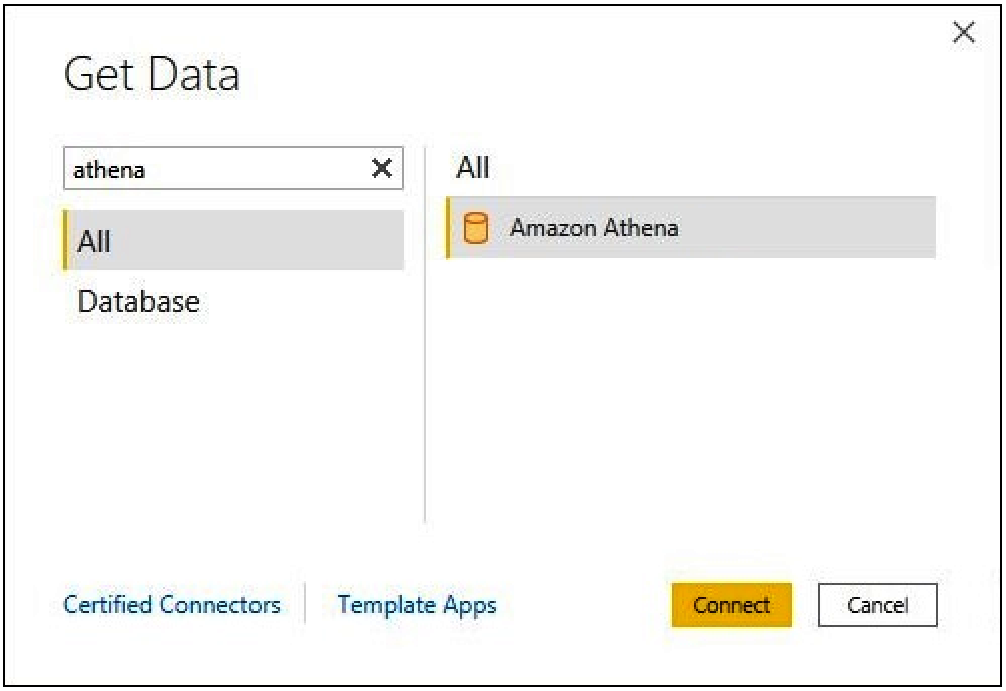
To connect Power BI desktop to your Amazon Athena data, perform the following steps.

To connect to Athena data from power BI desktop

1. Launch Power BI Desktop.
2. Do one of the following:
   - Choose **File**, **Get Data**

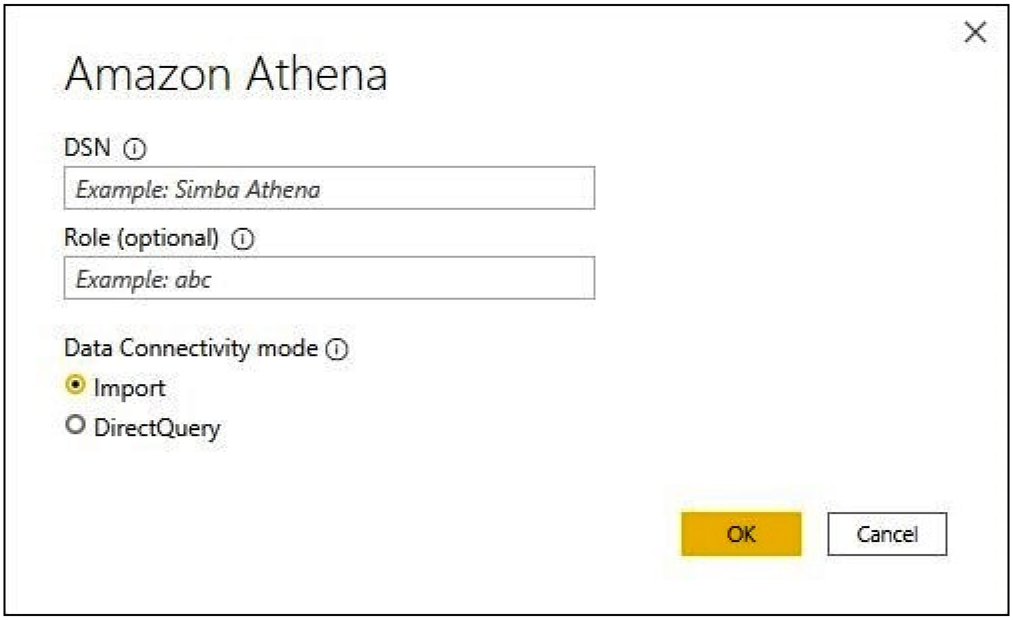
- From the **Home** ribbon, choose **Get Data**.

3. In the search box, enter **Athena**.
4. Select **Amazon Athena**, and then choose **Connect**.



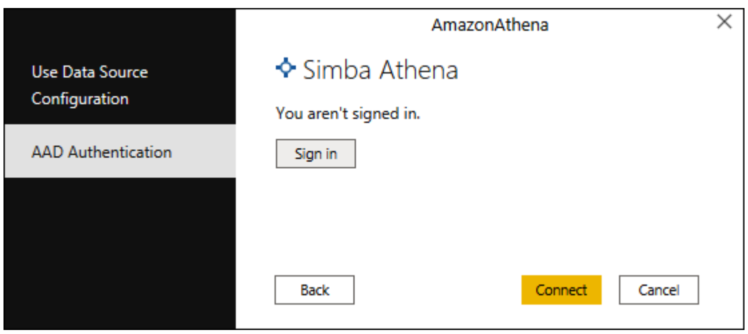
5. On the **Amazon Athena** connection page, enter the following information.

   - For **DSN**, enter the name of the ODBC DSN that you want to use. For instructions on configuring your DSN, see the ODBC driver documentation.
   - For **Data Connectivity mode**, choose a mode that is appropriate for your use case, following these general guidelines:
     - For smaller datasets, choose **Import**. When using Import mode, Power BI works with Athena to import the contents of the entire dataset for use in your visualizations.
     - For larger datasets, choose **DirectQuery**. In DirectQuery mode, no data is downloaded to your workstation. While you create or interact with a visualization, Microsoft Power BI works with Athena to dynamically query the underlying data source so that you're always viewing current data. For more information about DirectQuery, see Use DirectQuery in power BI desktop in the Microsoft documentation.

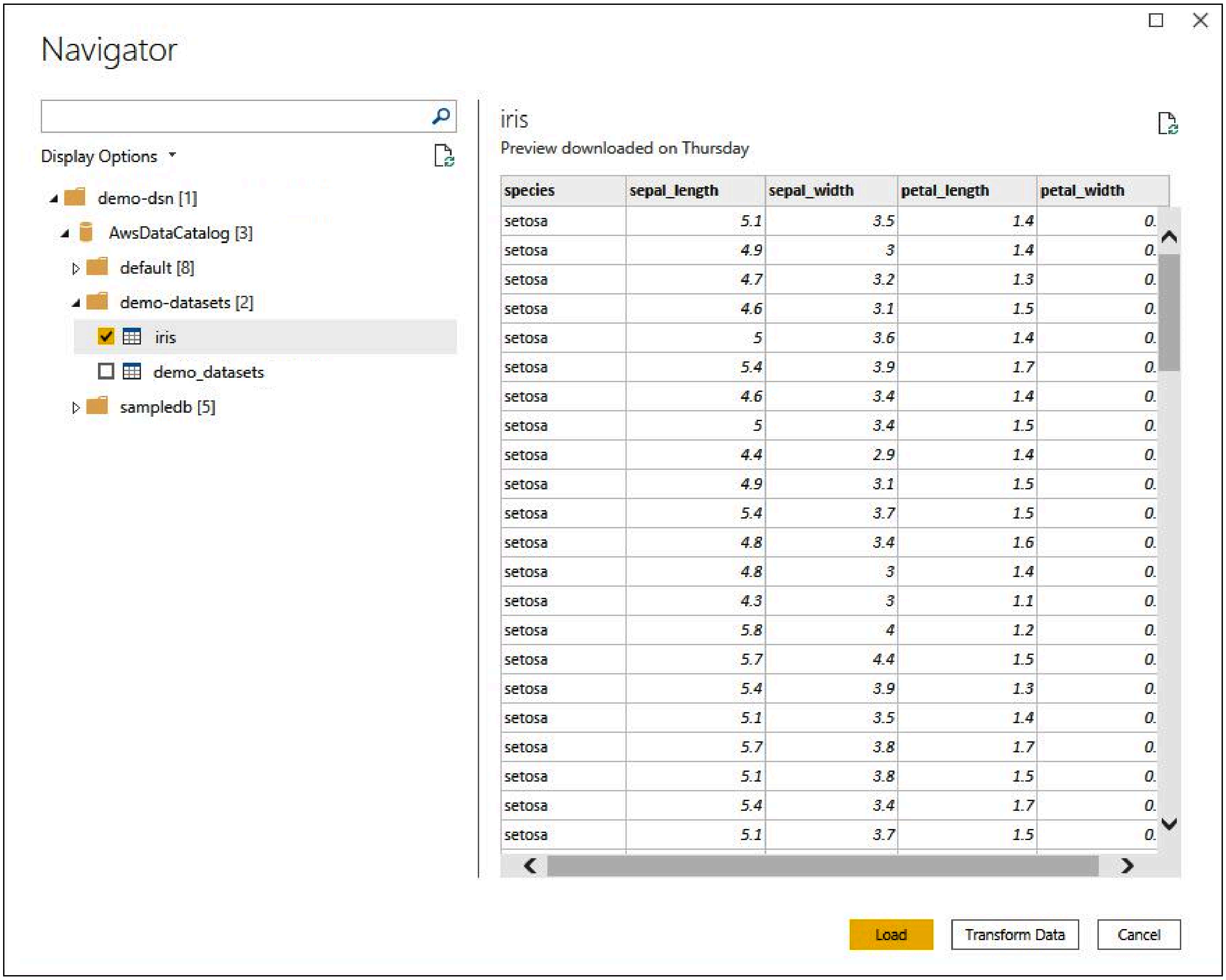


6. Choose **OK**.

7. At the prompt to configure data source authentication, choose either **Use Data Source Configuration** or **AAD Authentication**, and then choose **Connect**.



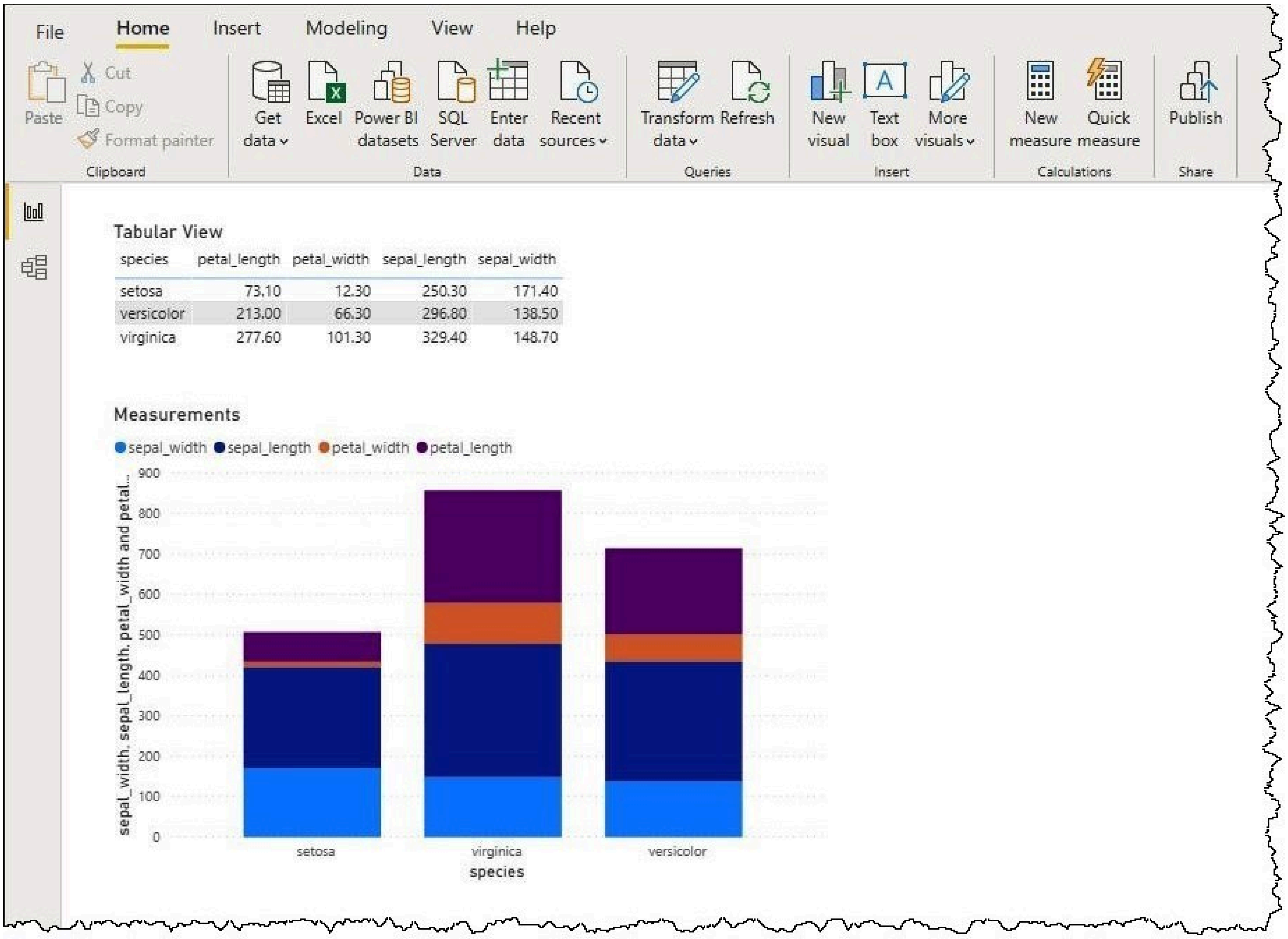
Your data catalog, databases, and tables appear in the **Navigator** dialog box.



8. In the **Display Options** pane, select the check box for the dataset that you want to use.

9. If you want to transform the dataset before you import it, go to the bottom of the dialog box and choose **Transform Data**. This opens the Power Query Editor so that you can filter and refine the set of data you want to use.

10. Choose **Load**. After the load is complete, you can create visualizations like the one in the following image. If you selected **DirectQuery** as the import mode, Power BI issues a query to Athena for the visualization that you requested.



Setting up an on-premises gateway

You can publish dashboards and datasets to the Power BI service so that other users can interact with them through web, mobile, and embedded apps. To see your data in the Microsoft Power BI Service, you install the Microsoft Power BI on-premises data gateway in your AWS account. The gateway works like a bridge between the Microsoft Power BI Service and Athena.

To download, install, and test an on-premises data gateway

1. Visit the Microsoft power BI gateway download page and choose either personal mode or standard mode. Personal mode is useful for testing the Athena connector locally. Standard mode is appropriate in a multiuser production setting.
2. To install an on-premises gateway (either personal or standard mode), see Install an on-premises data gateway in the Microsoft documentation.

3. To test the gateway, follow the steps in Use custom data connectors with the on-premises data gateway in the Microsoft documentation.

For more information about on-premises data gateways, see the following Microsoft resources.

- What is an on-premises data gateway?
- Guidance for deploying a data gateway for power BI

For an example of configuring Power BI Gateway for use with Athena, see the AWS Big Data Blog article Creating dashboards quickly on Microsoft power BI using amazon Athena.

Create databases and tables

Amazon Athena supports a subset of data definition language (DDL) statements and ANSI SQL functions and operators to define and query external tables where data resides in Amazon Simple Storage Service.

When you create a database and table in Athena, you describe the schema and the location of the data, making the data in the table ready for real-time querying.

To improve query performance and reduce costs, we recommend that you partition your data and use open source columnar formats for storage in Amazon S3, such as Apache parquet or ORC.

Topics

Create databases in Athena

A database in Athena is a logical grouping for tables you create in it. Before you create a database, you create a query output location.

Topics

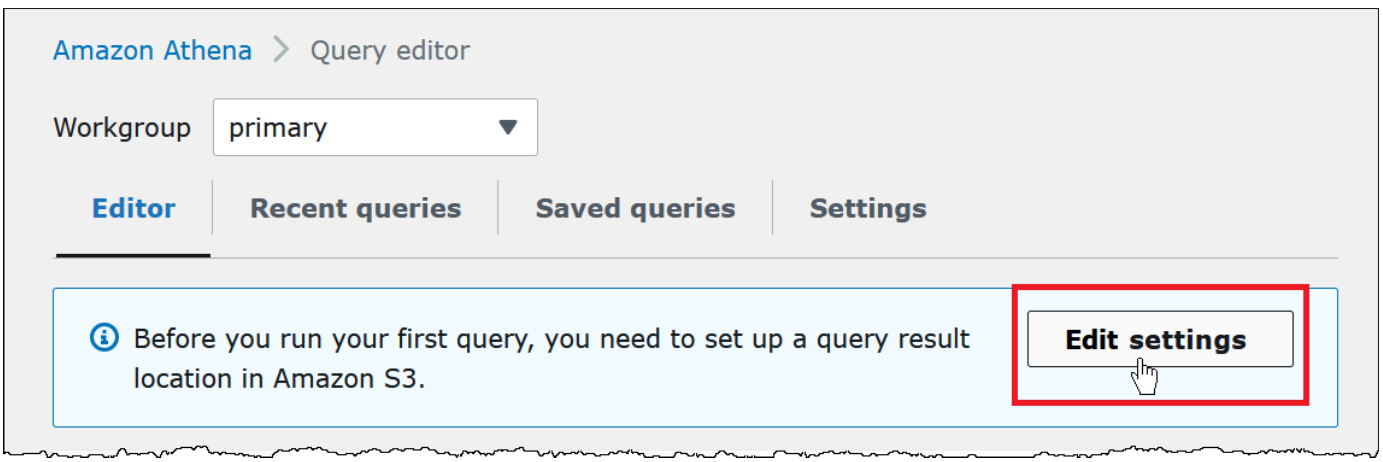
- [Create a database]

Create a query output location

If you do not already have a query output location set up in Amazon S3, perform the following prerequisite steps to do so.

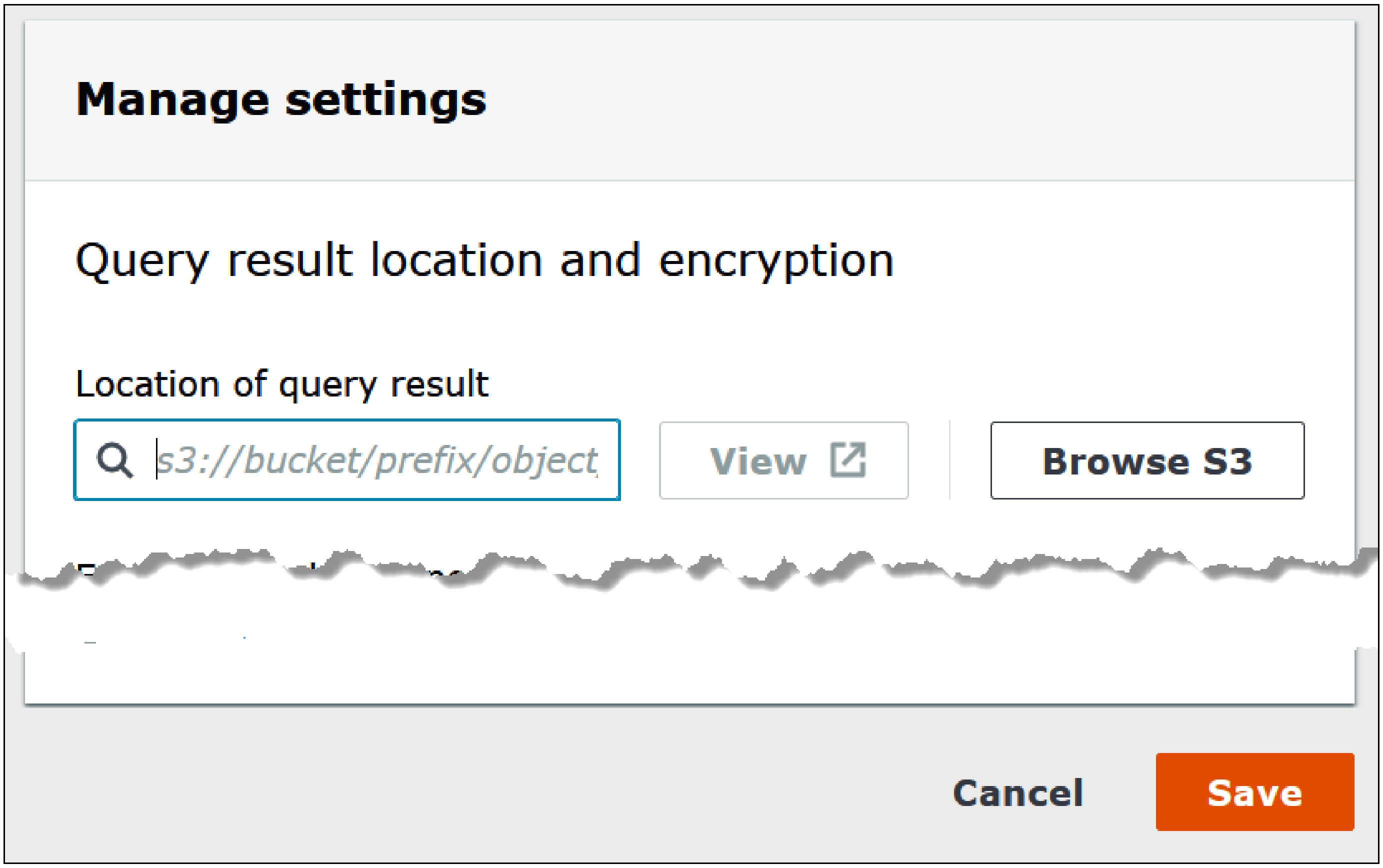
To create a query output location

1. Using the same AWS Region and account that you are using for Athena, follow the steps (for example, by using the Amazon S3 console) to create a bucket in Amazon S3 to hold your Athena query results. You will configure this bucket to be your query output location.
2. Open the Athena console at https://console.aws.amazon.com/athena/.
3. If this is your first time to visit the Athena console in this AWS Region, choose **Explore the query editor** to open the query editor. Otherwise, Athena opens in the query editor.
4. Choose **Edit Settings** to set up a query result location in Amazon S3.

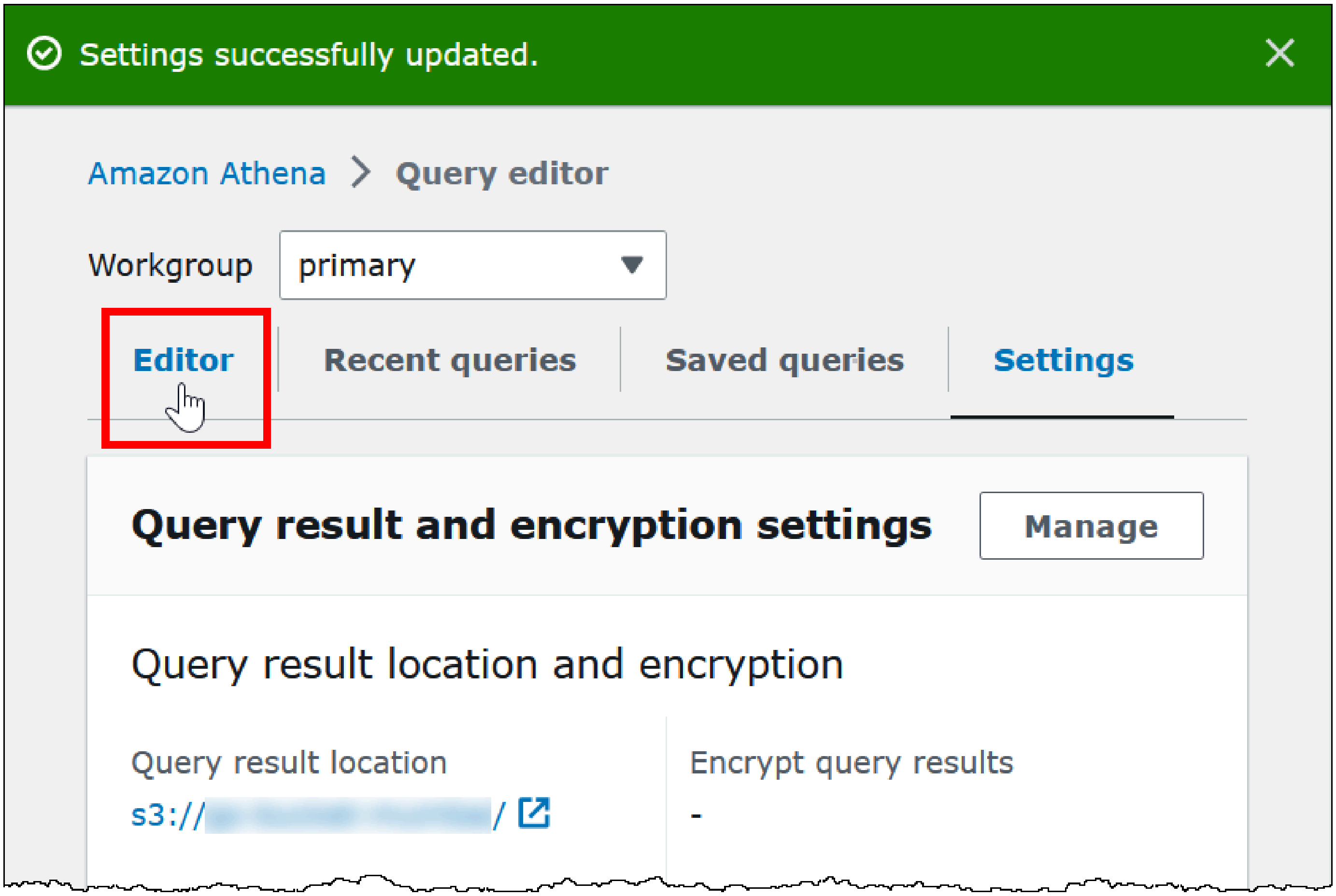


5. For **Manage settings**, do one of the following:

   - In the **Location of query result** box, enter the path to the bucket that you created in Amazon S3 for your query results. Prefix the path with `s3://`.
   - Choose **Browse S3**, choose the Amazon S3 bucket that you created for your current Region, and then choose **Choose**.



6. Choose **Save**.
7. Choose **Editor** to switch to the query editor.



Create a database

After you have set up a query results location, creating a database in the Athena console query editor is straightforward.

To create a database using the Athena query editor

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. On the **Editor** tab, in the query editor, enter the Hive data definition language (DDL) command `CREATE DATABASE myDataBase`. Replace *`myDatabase`* with the name that you want to use. For restrictions on database names, see [Name databases, tables, and columns](#).
3. Choose **Run** or press **`Ctrl+ENTER`**.
4. To make your database the current database, select it from the **Database** menu on the left of the query editor.

For information about controlling permissions to Athena databases, see Configure access to databases and tables in the AWS Glue Data Catalog.

Create tables in Athena

To create tables, you can run DDL statements in the Athena console, use the Athena **Create table form**, or use a JDBC or an ODBC driver. Athena uses Apache Hive to define tables and create databases, which are essentially a logical namespace of tables. Athena supports a variety of serializer-deserializer (SerDe) libraries for creating tables for specific data formats. For a list of supported SerDe libraries, see Choose a SerDe for your data.

When you create a database and table in Athena, you are simply describing the schema and the location where the table data are located in Amazon S3 for read-time querying. Athena does not modify your data in Amazon S3. Database and table, therefore, have a slightly different meaning than they do for traditional relational database systems because the data isn't stored along with the schema definition for the database and table.

Athena stores the schema in the AWS Glue Data Catalog and uses it to read the data when you query the table using SQL. This *schema-on-read* approach, which projects a schema onto your data when you run a query, eliminates the need for data loading or transformation.

Considerations and limitations

Following are some important limitations and considerations for tables in Athena.

Amazon S3 considerations

When you create a table, you specify an Amazon S3 bucket location for the underlying data using the `LOCATION` clause. Consider the following:

- Athena can only query the latest version of data on a versioned Amazon S3 bucket, and cannot query previous versions of the data.
- You must have permissions to work with data in the Amazon S3 location. For more information, see Control access to Amazon S3 from Athena.
- Athena supports querying objects that are stored with multiple storage classes in the same bucket specified by the `LOCATION` clause. For example, you can query data in objects that are stored in different Storage classes (Standard, Standard-IA and Intelligent-Tiering) in Amazon S3.
- Athena supports Requester Pays buckets. For information how to enable Requester Pays for buckets with source data you intend to query in Athena, see Create a workgroup.

- You can use Athena to query restored objects from the S3 Glacier Flexible Retrieval (formerly Glacier) and S3 Glacier Deep Archive Amazon S3 storage classes but you must enable the capability on a per-table basis. If you do not enable the feature on a table before you run a query, Athena skips all of the table's S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive objects during query execution. For more information, see Query restored Amazon S3 Glacier objects.

  For information about storage classes, see Storage classes, Changing the storage class of an object in amazon S3, Transitioning to the GLACIER storage class (object archival), and Requester Pays buckets in the *Amazon Simple Storage Service User Guide*.
- If you issue queries against Amazon S3 buckets with a large number of objects and the data is not partitioned, such queries may affect the Get request rate limits in Amazon S3 and lead to Amazon S3 exceptions. To prevent errors, partition your data. Additionally, consider tuning your Amazon S3 request rates. For more information, see Request rate and performance considerations.

For more information about specifying a location for your data in Amazon S3, see Specify a table location in Amazon S3.

Other considerations

- **Transactional data transformations not supported** – Athena does not support transaction-based operations (such as the ones found in Hive or Presto) on table data. For a full list of keywords not supported, see Unsupported DDL.
- **Operations on tables are ACID** – When you create, update, or delete tables, those operations are guaranteed ACID-compliant. For example, if multiple users or clients attempt to create or alter an existing table at the same time, only one will be successful.
- **Tables are EXTERNAL** – Except when creating Iceberg tables, always use the `EXTERNAL` keyword. If you use `CREATE TABLE` without the `EXTERNAL` keyword for non-Iceberg tables, Athena issues an error. When you drop a table in Athena, only the table metadata is removed; the data remains in Amazon S3.
- **Maximum query string length** – The maximum query string length is 256 KB.
- If you use the AWS Glue CreateTable API operation or the AWS CloudFormation `AWS::Glue::Table` template to create a table for use in Athena without specifying the `TableType` property and then run a DDL query like `SHOW CREATE TABLE` or `MSCK REPAIR TABLE`, you can receive the error message FAILED: NullPointerException Name is null.

To resolve the error, specify a value for the TableInput `TableType` attribute as part of the AWS Glue `CreateTable` API call or AWS CloudFormation template. Possible values for `TableType` include `EXTERNAL_TABLE` or `VIRTUAL_VIEW`.

This requirement applies only when you create a table using the AWS Glue `CreateTable` API operation or the `AWS::Glue::Table` template. If you create a table for Athena by using a DDL statement or an AWS Glue crawler, the `TableType` property is defined for you automatically.

Topics

Create tables using AWS Glue or the Athena console

You can create tables in Athena by using AWS Glue, the add table form, or by running a DDL statement in the Athena query editor.

To create a table using the AWS Glue crawler

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the query editor, next to **Tables and views**, choose **Create**, and then choose **AWS Glue crawler**.
3. Follow the steps on the **Add crawler** page of the AWS Glue console to add a crawler.

   For more information, see Use a crawler to add a table.

To create a table using the Athena create table form

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the query editor, next to **Tables and views**, choose **Create**, and then choose **S3 bucket data.**
3. In the **Create Table From S3 bucket data** form, enter the information to create your table, and then choose **Create table**. For more information about the fields in the form, see Use a form in the Athena console to add an AWS Glue table.

To create a table using a CREATE TABLE statement in the Athena query editor

1. From the **Database** menu, choose the database for which you want to create a table. If you don't specify a database in your CREATE TABLE statement, the table is created in the database that is currently selected in the query editor.
2. In the query editor, enter a statement as shown in the following example and then choose **Run**.

```
CREATE EXTERNAL TABLE myopencsvtable (
   firstname string,
   lastname string,
   job string,
   country string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   'separatorChar' = ',',
   'quoteChar' = '"',
   'escapeChar' = '\\'
   )
STORED AS TEXTFILE
LOCATION 's3://amzn-s3-demo-bucket/mycsv/';
```

Specify a table location in Amazon S3

When you run a CREATE TABLE query in Athena, Athena registers your table with the AWS Glue Data Catalog, which is where Athena stores your metadata.

To specify the path to your data in Amazon S3, use the LOCATION property in your CREATE TABLE statement, as in the following example:

```
CREATE EXTERNAL TABLE `test_table`(
...
)
ROW FORMAT ...
STORED AS INPUTFORMAT ...
OUTPUTFORMAT ...
LOCATION s3://amzn-s3-demo-bucket/folder/
```

- For information about naming buckets, see Bucket restrictions and limitations in the *Amazon Simple Storage Service User Guide*.

- For information about using folders in Amazon S3, see Using folders in the *Amazon Simple Storage Service User Guide.*

The `LOCATION` in Amazon S3 specifies *all* of the files representing your table.

> **Important**
>
> Athena reads *all* data stored in the Amazon S3 folder that you specify. If you have data that you do *not* want Athena to read, do not store that data in the same Amazon S3 folder as the data that you do want Athena to read.

When you specify the `LOCATION` in the `CREATE TABLE` statement, use the following guidelines:

- Use a trailing slash.
- You can use a path to an Amazon S3 folder or an Amazon S3 access point alias. For information about Amazon S3 access point aliases, see Using a bucket-style alias for your access point in the *Amazon S3 User Guide*.

Use:

```
s3://amzn-s3-demo-bucket/folder/
```

```
s3://amzn-s3-demo-bucket-metadata-s3alias/folder/
```

Do not use any of the following items for specifying the `LOCATION` for your data.

- Do not use filenames, underscores, wildcards, or glob patterns for specifying file locations.
- Do not add the full HTTP notation, such as `s3.amazon.com` to the Amazon S3 bucket path.
- Do not use empty folders like `//` in the path, as follows: `S3://amzn-s3-demo-bucket/folder//folder/`.
- Do not use paths like the following:

```
s3://amzn-s3-demo-bucket
s3://amzn-s3-demo-bucket/*
s3://amzn-s3-demo-bucket/mySpecialFile.dat
```

```
s3://amzn-s3-demo-bucket/prefix/filename.csv
s3://amzn-s3-demo-bucket.s3.amazon.com
S3://amzn-s3-demo-bucket/prefix//prefix/
arn:aws:s3:::amzn-s3-demo-bucket/prefix
s3://arn:aws:s3:<region>:<account_id>:accesspoint/<accesspointname>
https://<accesspointname>-<number>.s3-accesspoint.<region>.amazonaws.com
```

Show table information after creation

After you have created a table in Athena, its name displays in the **Tables** list on the left in the Athena console. To show information about the table and manage it, choose the vertical three dots next to the table name in the Athena console.

- **Preview table** – Shows the first 10 rows of all columns by running the `SELECT * FROM "database_name"."table_name" LIMIT 10` statement in the Athena query editor.
- **Generate table DDL** – Generates a DDL statement that you can use to re-create the table by running the `SHOW CREATE TABLE` `table_name` statement in the Athena query editor.
- **Load partitions** – Runs the `MSCK REPAIR TABLE` `table_name` statement in the Athena query editor. This option is available only if the table has partitions.
- **Insert into editor** – Inserts the name of the table into the query editor at the current editing location.
- **Delete table** – Displays a confirmation dialog box asking if you want to delete the table. If you agree, runs the `DROP TABLE` `table_name` statement in the Athena query editor.
- **Table properties** – Shows the table name, database name, time created, and whether the table has encrypted data.

Name databases, tables, and columns

Use these guidelines for naming databases, tables, and columns in Athena.

Database, table, and column name requirements

- Acceptable characters for database names, table names, and column names in AWS Glue must be a UTF-8 string and should be in lower case. Note that Athena automatically lowers any upper case names in DDL queries when it creates databases, tables, or columns. The string must not be less than 1 or more than 255 bytes long.

- Currently, it is possible to have leading spaces at the start of names. Because these leading spaces can be hard to detect and can cause usability issues after creation, avoid inadvertently creating object names that have leading spaces.
- If you use an AWS::Glue::Database AWS CloudFormation template to create an AWS Glue database and do not specify a database name, AWS Glue automatically generates a database name in the format *resource_name-random_string* that is not compatible with Athena.
- You can use the AWS Glue Catalog Manager to rename columns, but not table names or database names. To work around this limitation, you must use a definition of the old database to create a database with the new name. Then you use definitions of the tables from the old database to re-create the tables in the new database. To do this, you can use the AWS CLI or AWS Glue SDK. For steps, see Use the AWS CLI to recreate an AWS Glue database and its tables.

Use lower case for table names and table column names in Athena

Athena accepts mixed case in DDL and DML queries, but lower cases the names when it executes the query. For this reason, avoid using mixed case for table or column names, and do not rely on casing alone in Athena to distinguish such names. For example, if you use a DDL statement to create a column named `Castle`, the column created will be lowercased to `castle`. If you then specify the column name in a DML query as `Castle` or `CASTLE`, Athena will lowercase the name for you to run the query, but display the column heading using the casing that you chose in the query.

Database, table, and column names must be less than or equal to 255 characters long.

Names that begin with an underscore

When creating tables, use backticks to enclose table, view, or column names that begin with an underscore. For example:

```
CREATE EXTERNAL TABLE IF NOT EXISTS `_myunderscoretable`(
  `_id` string, `_index` string)
LOCATION 's3://amzn-s3-demo-bucket/'
```

Table, view, or column names that begin with numbers

When running `SELECT`, `CTAS`, or `VIEW` queries, put quotation marks around identifiers like table, view, or column names that start with a digit. For example:

```
CREATE OR REPLACE VIEW "123view" AS
SELECT "123columnone", "123columntwo"
FROM "234table"
```

Column names and complex types

For complex types, only alphanumeric characters, underscore (_), and period (.) are allowed in column names. To create a table and mappings for keys that have restricted characters, you can use a custom DDL statement. For more information, see the article Create tables in Amazon Athena from nested JSON and mappings using JSONSerDe in the *AWS Big Data Blog*.

Reserved words

Certain reserved words in Athena must be escaped. To escape reserved keywords in DDL statements, enclose them in backticks (`). To escape reserved keywords in SQL SELECT statements and in queries on views, enclose them in double quotes (").

For more information, see Escape reserved keywords in queries.

Additional resources

For full database and table creation syntax, see the following pages.

- CREATE DATABASE
- CREATE TABLE

For more information about databases and tables in AWS Glue, see Databases and Tables in the *AWS Glue Developer Guide*.

Escape reserved keywords in queries

When you run queries in Athena that include reserved keywords, you must escape them by enclosing them in special characters. Use the lists in this topic to check which keywords are reserved in Athena.

To escape reserved keywords in DDL statements, enclose them in backticks (`). To escape reserved keywords in SQL SELECT statements and in queries on views, enclose them in double quotes (").

- Reserved keywords to escape in DDL statements

- Reserved keywords to escape in SQL SELECT statements
- Examples of queries with reserved words

Reserved keywords to escape in DDL statements

Athena uses the following list of reserved keywords in its DDL statements. If you use them without escaping them, Athena issues an error. To escape them, enclose them in backticks (`).

You cannot use DDL reserved keywords as identifier names in DDL statements without enclosing them in backticks (`).

```
ALL, ALTER, AND, ARRAY, AS, AUTHORIZATION, BETWEEN, BIGINT,
BINARY, BOOLEAN, BOTH, BY, CASE, CASHE, CAST, CHAR, COLUMN,
CONF, CONSTRAINT, COMMIT, CREATE, CROSS, CUBE, CURRENT,
CURRENT_DATE, CURRENT_TIMESTAMP, CURSOR, DATABASE, DATE,
DAYOFWEEK, DECIMAL, DELETE, DESCRIBE, DISTINCT, DIV, DOUBLE,
DROP, ELSE, END, EXCHANGE, EXISTS, EXTENDED, EXTERNAL, EXTRACT,
FALSE, FETCH, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN, FROM,
FULL, FUNCTION, GRANT, GROUP, GROUPING, HAVING, IF, IMPORT,
IN, INNER, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO,
IS, JOIN, LATERAL, LEFT, LESS, LIKE, LOCAL, MACRO, MAP, MORE,
NONE, NOT, NULL, NUMERIC, OF, ON, ONLY, OR, ORDER, OUT,
OUTER, OVER, PARTIALSCAN, PARTITION, PERCENT, PRECEDING,
PRECISION, PRESERVE, PRIMARY, PROCEDURE, RANGE, READS,
REDUCE, REGEXP, REFERENCES, REVOKE, RIGHT, RLIKE, ROLLBACK,
ROLLUP, ROW, ROWS, SELECT, SET, SMALLINT, START,TABLE,
TABLESAMPLE, THEN, TIME, TIMESTAMP, TO, TRANSFORM, TRIGGER,
TRUE, TRUNCATE, UNBOUNDED,UNION, UNIQUEJOIN, UPDATE, USER,
USING, UTC_TIMESTAMP, VALUES, VARCHAR, VIEWS, WHEN, WHERE,
WINDOW, WITH
```

Reserved keywords to escape in SQL SELECT statements

Athena uses the following list of reserved keywords in SQL SELECT statements and in queries on views.

If you use these keywords as identifiers, you must enclose them in double quotes (") in your query statements.

```
ALTER, AND, AS, BETWEEN, BY, CASE, CAST, CONSTRAINT, CREATE,
```

```
CROSS, CUBE, CURRENT_CATALOG, CURRENT_DATE, CURRENT_PATH,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER,
DEALLOCATE, DELETE, DESCRIBE, DISTINCT, DROP, ELSE, END, ESCAPE,
EXCEPT, EXECUTE, EXISTS, EXTRACT, FALSE, FIRST, FOR, FROM,
FULL, GROUP, GROUPING, HAVING, IN, INNER, INSERT, INTERSECT,
INTO, IS, JOIN, JSON_ARRAY, JSON_EXISTS, JSON_OBJECT,
JSON_QUERY, JSON_TABLE, JSON_VALUE, LAST, LEFT, LIKE,
LISTAGG, LOCALTIME, LOCALTIMESTAMP, NATURAL, NORMALIZE,
NOT, NULL, OF, ON, OR, ORDER, OUTER, PREPARE, RECURSIVE, RIGHT,
ROLLUP, SELECT, SKIP, TABLE, THEN, TRIM, TRUE, UESCAPE, UNION,
UNNEST, USING, VALUES, WHEN, WHERE, WITH
```

Examples of queries with reserved words

The query in the following example uses backticks (`) to escape the DDL-related reserved keywords *partition* and *date* that are used for a table name and one of the column names:

```
CREATE EXTERNAL TABLE `partition` (
`date` INT,
col2 STRING
)
PARTITIONED BY (year STRING)
STORED AS TEXTFILE
LOCATION 's3://amzn-s3-demo-bucket/test_examples/';
```

The following example queries include a column name containing the DDL-related reserved keywords in `ALTER TABLE ADD PARTITION` and `ALTER TABLE DROP PARTITION` statements. The DDL reserved keywords are enclosed in backticks (`):

```
ALTER TABLE test_table
ADD PARTITION (`date` = '2018-05-14')
```

```
ALTER TABLE test_table
DROP PARTITION (`partition` = 'test_partition_value')
```

The following example query includes a reserved keyword (end) as an identifier in a `SELECT` statement. The keyword is escaped in double quotes:

```
SELECT *
FROM TestTable
```

```
WHERE "end" != nil;
```

The following example query includes a reserved keyword (first) in a SELECT statement. The keyword is escaped in double quotes:

```
SELECT "itemId"."first"
FROM testTable
LIMIT 10;
```

Create a table from query results (CTAS)

A CREATE TABLE AS SELECT (CTAS) query creates a new table in Athena from the results of a SELECT statement from another query. Athena stores data files created by the CTAS statement in a specified location in Amazon S3. For syntax, see CREATE TABLE AS.

CREATE TABLE AS combines a CREATE TABLE DDL statement with a SELECT DML statement and therefore technically contains both DDL and DML. However, note that for Service Quotas purposes, CTAS queries in Athena are treated as DML. For information about Service Quotas in Athena, see Service Quotas.

Use CTAS queries to:

- Create tables from query results in one step, without repeatedly querying raw data sets. This makes it easier to work with raw data sets.
- Transform query results and migrate tables into other table formats such as Apache Iceberg. This improves query performance and reduces query costs in Athena. For information, see Create Iceberg tables.
- Transform query results into storage formats such as Parquet and ORC. This improves query performance and reduces query costs in Athena. For information, see Use columnar storage formats.
- Create copies of existing tables that contain only the data you need.

Topics

- Use CTAS and INSERT INTO for ETL and data analysis
- Use CTAS and INSERT INTO to work around the 100 partition limit

Considerations and limitations for CTAS queries

The following sections describe considerations and limitations to keep in mind when you use `CREATE TABLE AS SELECT` (CTAS) queries in Athena.

Learn the CTAS query syntax

The CTAS query syntax differs from the syntax of `CREATE [EXTERNAL] TABLE` used for creating tables. See CREATE TABLE AS.

The difference between views and CTAS queries

CTAS queries write new data to a specified location in Amazon S3. Views do not write any data.

Specify a location for your CTAS query results

If your workgroup overrides the client-side setting for query results location, Athena creates your table in the location `s3://amzn-s3-demo-bucket/tables/<query-id>/`. To see the query results location specified for the workgroup, view the workgroup's details.

If your workgroup does not override the query results location, you can use the syntax `WITH (external_location ='s3://amzn-s3-demo-bucket/')` in your CTAS query to specify where your CTAS query results are stored.

> **Note**
>
> The `external_location` property must specify a location that is empty. A CTAS query checks that the path location (prefix) in the bucket is empty and never overwrites the data if the location already has data in it. To use the same location again, delete the data in the key prefix location in the bucket.

If you omit the `external_location` syntax and are not using the workgroup setting, Athena uses your client-side setting for the query results location and creates your table in the location `s3://amzn-s3-demo-bucket/<Unsaved-or-query-name>/<year>/<month/<date>/tables/<query-id>/`.

Locate orphaned files

If a CTAS or INSERT INTO statement fails, it is possible that orphaned data are left in the data location. Because Athena in some cases does not delete data or partial data from your bucket, you might be able to read this partial data in subsequent queries. To locate orphaned files for inspection or deletion, you can use the data manifest file that Athena provides to track the list of files to be written. For more information, see Identify query output files and DataManifestLocation.

Remember that ORDER BY clauses are ignored

In a CTAS query, Athena ignores ORDER BY clauses in the SELECT portion of the query.

According to the SQL specification (ISO 9075 Part 2), the ordering of the rows of a table specified by a query expression is guaranteed only for the query expression that immediately contains the ORDER BY clause. Tables in SQL are in any case inherently unordered, and implementing the ORDER BY in sub query clauses would both cause the query to perform poorly and not result in ordered output. Thus, in Athena CTAS queries, there is no guarantee that the order specified by the ORDER BY clause will be preserved when the data is written.

Choose a format to store your query results

You can store CTAS results in PARQUET, ORC, AVRO, JSON, and TEXTFILE. Multi-character delimiters are not supported for the CTAS TEXTFILE format. If you don't specify a data storage format, CTAS query results are stored in Parquet by default.

CTAS queries do not require specifying a SerDe to interpret format transformations. See Example: Writing query results to a different format.

Consider compression formats

GZIP compression is used for CTAS query results in JSON and TEXTFILE formats. For Parquet, you can use GZIP or SNAPPY, and the default is GZIP. For ORC, you can use LZ4, SNAPPY, ZLIB, or ZSTD, and the default is ZLIB. For CTAS examples that specify compression, see Example: Specifying data storage and compression formats. For more information about compression in Athena, see Use compression in Athena.

Partition and bucket your results

You can partition and bucket the results data of a CTAS query. To specify properties of the destination table, include partitioning and bucketing predicates at the end of the WITH clause.

For more information, see Use partitioning and bucketing and Example: Creating bucketed and partitioned tables.

When you use CTAS to create a partitioned table, Athena has a write limit of 100 partitions. For information about working around the 100-partition limitation, see Use CTAS and INSERT INTO to work around the 100 partition limit.

Encrypt your results

You can encrypt CTAS query results in Amazon S3, similar to the way you encrypt other query results in Athena. For more information, see Encrypt Athena query results stored in Amazon S3.

The expected bucket owner setting does not apply to CTAS

For CTAS statements, the expected bucket owner setting does not apply to the destination table location in Amazon S3. The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. For more information, see Specify a query result location using the Athena console.

Column data types are preserved

Column data types for a CTAS query are the same as specified for the original query.

Create CTAS queries in the Athena console

In the Athena console, you can create a CTAS query from another query.

To create a CTAS query from another query

1. Run the query in the Athena console query editor.
2. At the bottom of the query editor, choose the **Create** option, and then choose **Table from query**.
3. In the **Create table as select** form, complete the fields as follows:

    a. For **Table name**, enter the name for your new table. Use only lowercase and underscores, such as `my_select_query_parquet`.

    b. For **Database configuration**, use the options to choose an existing database or create a database.

    c. (Optional) In **Result configuration**, for **Location of CTAS query results**, if your workgroup query results location setting does not override this option, do one of the following:

- Enter the path to an existing S3 location in the search box, or choose **Browse S3** to choose a location from a list.
- Choose **View** to open the **Buckets** page of the Amazon S3 console where you can view more information about your existing buckets and choose or create a bucket with your own settings.

You should specify an empty location in Amazon S3 where the data will be output. If data already exists in the location that you specify, the query fails with an error.

If your workgroup query results location setting overrides this location setting, Athena creates your table in the location `s3://amzn-s3-demo-bucket/tables/`*`query_id`*`/`

d. For **Data format**, specify the format that your data is in.

- **Table type** – The default table type in Athena is Apache Hive.
- **File format** – Choose among options like CSV, TSV, JSON, Parquet, or ORC. For information about the Parquet and ORC formats, see Use columnar storage formats.
- **Write compression** – (Optional) Choose a compression format. Athena supports a variety of compression formats for reading and writing data, including reading from a table that uses multiple compression formats. For example, Athena can successfully read the data in a table that uses Parquet file format when some Parquet files are compressed with Snappy and other Parquet files are compressed with GZIP. The same principle applies for ORC, text file, and JSON storage formats. For more information, see Use compression in Athena.
- **Partitions** – (Optional) Select the columns that you want to partition. Partitioning your data restricts the amount of data scanned by each query, thus improving performance and reducing cost. You can partition your data by any key. For more information, see Partition your data.
- **Buckets** – (Optional) Select the columns that you want to bucket. Bucketing is a technique that groups data based on specific columns together within a single partition. These columns are known as *bucket keys*. By grouping related data into a single bucket (a file within a partition), you significantly reduce the amount of data scanned by Athena, thus improving query performance and reducing cost. For more information, see Use partitioning and bucketing.

e. For **Preview table query**, review your query. For query syntax, see CREATE TABLE AS.

f. Choose **Create table**.

The Athena console has a SQL template that you can also use to create a CTAS query.

To create a CTAS query using a SQL template

Use the `CREATE TABLE AS SELECT` template to create a CTAS query in the query editor.

1. In the Athena console, next to **Tables and views**, choose **Create table**, and then choose **CREATE TABLE AS SELECT**. This populates the query editor with a CTAS query with placeholder values.
2. In the query editor, edit the query as needed. For query syntax, see CREATE TABLE AS.
3. Choose **Run**.

For examples, see Examples of CTAS queries.

Examples of CTAS queries

Use the following examples to create CTAS queries. For information about the CTAS syntax, see CREATE TABLE AS.

In this section:

- Example: Duplicating a table by selecting all columns
- Example: Selecting specific columns from one or more tables
- Example: Creating an empty copy of an existing table
- Example: Specifying data storage and compression formats
- Example: Writing query results to a different format
- Example: Creating unpartitioned tables
- Example: Creating partitioned tables
- Example: Creating bucketed and partitioned tables
- Example: Creating an Iceberg table with Parquet data
- Example: Creating an Iceberg table with Avro data

Example Example: Duplicating a table by selecting all columns

The following example creates a table by copying all columns from a table:

```
CREATE TABLE new_table AS
SELECT *
FROM old_table;
```

In the following variation of the same example, your SELECT statement also includes a WHERE clause. In this case, the query selects only those rows from the table that satisfy the WHERE clause:

```
CREATE TABLE new_table AS
SELECT *
FROM old_table
WHERE condition;
```

Example Example: Selecting specific columns from one or more tables

The following example creates a new query that runs on a set of columns from another table:

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
FROM old_table;
```

This variation of the same example creates a new table from specific columns from multiple tables:

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
FROM old_table_1, old_table_2, ... old_table_n;
```

Example Example: Creating an empty copy of an existing table

The following example uses WITH NO DATA to create a new table that is empty and has the same schema as the original table:

```
CREATE TABLE new_table
AS SELECT *
FROM old_table
WITH NO DATA;
```

Example Example: Specifying data storage and compression formats

With CTAS, you can use a source table in one storage format to create another table in a different storage format.

Use the format property to specify ORC, PARQUET, AVRO, JSON, or TEXTFILE as the storage format for the new table.

For the PARQUET, ORC, TEXTFILE, and JSON storage formats, use the write_compression property to specify the compression format for the new table's data. For information about the compression formats that each file format supports, see Use compression in Athena.

The following example specifies that data in the table new_table be stored in Parquet format and use Snappy compression. The default compression for Parquet is GZIP.

```
CREATE TABLE new_table
WITH (
      format = 'Parquet',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table;
```

The following example specifies that data in the table new_table be stored in ORC format using Snappy compression. The default compression for ORC is ZLIB.

```
CREATE TABLE new_table
WITH (format = 'ORC',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

The following example specifies that data in the table new_table be stored in textfile format using Snappy compression. The default compression for both the textfile and JSON formats is GZIP.

```
CREATE TABLE new_table
WITH (format = 'TEXTFILE',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

Example Example: Writing query results to a different format

The following CTAS query selects all records from old_table, which could be stored in CSV or another format, and creates a new table with underlying data saved to Amazon S3 in ORC format:

```
CREATE TABLE my_orc_ctas_table
```

```
WITH (
      external_location = 's3://amzn-s3-demo-bucket/my_orc_stas_table/',
      format = 'ORC')
AS SELECT *
FROM old_table;
```

Example Example: Creating unpartitioned tables

The following examples create tables that are not partitioned. The table data is stored in different formats. Some of these examples specify the external location.

The following example creates a CTAS query that stores the results as a text file:

```
CREATE TABLE ctas_csv_unpartitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://amzn-s3-demo-bucket/ctas_csv_unpartitioned/')
AS SELECT key1, name1, address1, comment1
FROM table1;
```

In the following example, results are stored in Parquet, and the default results location is used:

```
CREATE TABLE ctas_parquet_unpartitioned
WITH (format = 'PARQUET')
AS SELECT key1, name1, comment1
FROM table1;
```

In the following query, the table is stored in JSON, and specific columns are selected from the original table's results:

```
CREATE TABLE ctas_json_unpartitioned
WITH (
     format = 'JSON',
     external_location = 's3://amzn-s3-demo-bucket/ctas_json_unpartitioned/')
AS SELECT key1, name1, address1, comment1
FROM table1;
```

In the following example, the format is ORC:

```
CREATE TABLE ctas_orc_unpartitioned
```

```
WITH (
     format = 'ORC')
AS SELECT key1, name1, comment1
FROM table1;
```

In the following example, the format is Avro:

```
CREATE TABLE ctas_avro_unpartitioned
WITH (
     format = 'AVRO',
     external_location = 's3://amzn-s3-demo-bucket/ctas_avro_unpartitioned/')
AS SELECT key1, name1, comment1
FROM table1;
```

Example Example: Creating partitioned tables

The following examples show `CREATE TABLE AS SELECT` queries for partitioned tables in different storage formats, using `partitioned_by`, and other properties in the `WITH` clause. For syntax, see CTAS table properties. For more information about choosing the columns for partitioning, see Use partitioning and bucketing.

> **Note**
>
> List partition columns at the end of the list of columns in the `SELECT` statement. You can partition by more than one column, and have up to 100 unique partition and bucket combinations. For example, you can have 100 partitions if no buckets are specified.

```
CREATE TABLE ctas_csv_partitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://amzn-s3-demo-bucket/ctas_csv_partitioned/',
     partitioned_by = ARRAY['key1'])
AS SELECT name1, address1, comment1, key1
FROM tables1;
```

```
CREATE TABLE ctas_json_partitioned
WITH (
     format = 'JSON',
```

```
    external_location = 's3://amzn-s3-demo-bucket/ctas_json_partitioned/',
    partitioned_by = ARRAY['key1'])
AS select name1, address1, comment1, key1
FROM table1;
```

Example Example: Creating bucketed and partitioned tables

The following example shows a `CREATE TABLE AS SELECT` query that uses both partitioning and bucketing for storing query results in Amazon S3. The table results are partitioned and bucketed by different columns. Athena supports a maximum of 100 unique bucket and partition combinations. For example, if you create a table with five buckets, 20 partitions with five buckets each are supported. For syntax, see CTAS table properties.

For information about choosing the columns for bucketing, see Use partitioning and bucketing.

```
CREATE TABLE ctas_avro_bucketed
WITH (
      format = 'AVRO',
      external_location = 's3://amzn-s3-demo-bucket/ctas_avro_bucketed/',
      partitioned_by = ARRAY['nationkey'],
      bucketed_by = ARRAY['mktsegment'],
      bucket_count = 3)
AS SELECT key1, name1, address1, phone1, acctbal, mktsegment, comment1, nationkey
FROM table1;
```

Example Example: Creating an Iceberg table with Parquet data

The following example creates an Iceberg table with Parquet data files. The files are partitioned by month using the `dt` column in `table1`. The example updates the retention properties on the table so that 10 snapshots are retained by default on every branch in the table. Snapshots within the past 7 days are also retained. For more information about Iceberg table properties in Athena, see Specify table properties.

```
CREATE TABLE ctas_iceberg_parquet
WITH (table_type = 'ICEBERG',
      format = 'PARQUET',
      location = 's3://amzn-s3-demo-bucket/ctas_iceberg_parquet/',
      is_external = false,
      partitioning = ARRAY['month(dt)'],
      vacuum_min_snapshots_to_keep = 10,
      vacuum_max_snapshot_age_seconds = 604800
```

```
    )
AS SELECT key1, name1, dt FROM table1;
```

Example Example: Creating an Iceberg table with Avro data

The following example creates an Iceberg table with Avro data files partitioned by `key1`.

```
CREATE TABLE ctas_iceberg_avro
WITH ( format = 'AVRO',
       location = 's3://amzn-s3-demo-bucket/ctas_iceberg_avro/',
       is_external = false,
       table_type = 'ICEBERG',
       partitioning = ARRAY['key1'])
AS SELECT key1, name1, date FROM table1;
```

Use CTAS and INSERT INTO for ETL and data analysis

You can use Create Table as Select (CTAS) and INSERT INTO statements in Athena to extract, transform, and load (ETL) data into Amazon S3 for data processing. This topic shows you how to use these statements to partition and convert a dataset into columnar data format to optimize it for data analysis.

CTAS statements use standard SELECT queries to create new tables. You can use a CTAS statement to create a subset of your data for analysis. In one CTAS statement, you can partition the data, specify compression, and convert the data into a columnar format like Apache Parquet or Apache ORC. When you run the CTAS query, the tables and partitions that it creates are automatically added to the AWS Glue Data Catalog. This makes the new tables and partitions that it creates immediately available for subsequent queries.

INSERT INTO statements insert new rows into a destination table based on a SELECT query statement that runs on a source table. You can use INSERT INTO statements to transform and load source table data in CSV format into destination table data using all transforms that CTAS supports.

Overview

In Athena, use a CTAS statement to perform an initial batch conversion of the data. Then use multiple INSERT INTO statements to make incremental updates to the table created by the CTAS statement.

Steps

Step 1: Create a table based on the original dataset

The example in this topic uses an Amazon S3 readable subset of the publicly available NOAA global historical climatology network daily (GHCN-d) dataset. The data on Amazon S3 has the following characteristics.

```
Location: s3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/
Total objects: 41727
Size of CSV dataset: 11.3 GB
Region: us-east-1
```

The original data is stored in Amazon S3 with no partitions. The data is in CSV format in files like the following.

```
2019-10-31 13:06:57  413.1 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0000
2019-10-31 13:06:57  412.0 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0001
2019-10-31 13:06:57   34.4 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0002
2019-10-31 13:06:57  412.2 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0100
2019-10-31 13:06:57  412.7 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0101
```

The file sizes in this sample are relatively small. By merging them into larger files, you can reduce the total number of files, enabling better query performance. You can use CTAS and INSERT INTO statements to enhance query performance.

To create a database and table based on the sample dataset

1. In the Athena console, choose the **US East (N. Virginia)** AWS Region. Be sure to run all queries in this tutorial in `us-east-1`.
2. In the Athena query editor, run the CREATE DATABASE command to create a database.

   ```
   CREATE DATABASE blogdb
   ```

3. Run the following statement to create a table.

```
CREATE EXTERNAL TABLE `blogdb`.`original_csv` (
  `id` string,
  `date` string,
  `element` string,
  `datavalue` bigint,
  `mflag` string,
  `qflag` string,
  `sflag` string,
  `obstime` bigint)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/'
```

Step 2: Use CTAS to partition, convert, and compress the data

After you create a table, you can use a single CTAS statement to convert the data to Parquet format with Snappy compression and to partition the data by year.

The table you created in Step 1 has a `date` field with the date formatted as `YYYYMMDD` (for example, `20100104`). Because the new table will be partitioned on `year`, the sample statement in the following procedure uses the Presto function `substr("date",1,4)` to extract the `year` value from the `date` field.

To convert the data to parquet format with snappy compression, partitioning by year

- Run the following CTAS statement, replacing *`your-bucket`* with your Amazon S3 bucket location.

```
CREATE table new_parquet
WITH (format='PARQUET',
parquet_compression='SNAPPY',
partitioned_by=array['year'],
external_location = 's3://amzn-s3-demo-bucket/optimized-data/')
AS
```

```
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) >= 2015
        AND cast(substr("date",1,4) AS bigint) <= 2019
```

Note

In this example, the table that you create includes only the data from 2015 to 2019. In Step 3, you add new data to this table using the INSERT INTO command.

When the query completes, use the following procedure to verify the output in the Amazon S3 location that you specified in the CTAS statement.

To see the partitions and parquet files created by the CTAS statement

1. To show the partitions created, run the following AWS CLI command. Be sure to include the final forward slash (/).

   ```
   aws s3 ls s3://amzn-s3-demo-bucket/optimized-data/
   ```

   The output shows the partitions.

   ```
         PRE year=2015/
         PRE year=2016/
         PRE year=2017/
         PRE year=2018/
         PRE year=2019/
   ```

2. To see the Parquet files, run the following command. Note that the | *head -5* option, which restricts the output to the first five results, is not available on Windows.

```
aws s3 ls s3://amzn-s3-demo-bucket/optimized-data/ --recursive --human-readable |
 head -5
```

The output resembles the following.

```
2019-10-31 14:51:05    7.3 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_1be48df2-3154-438b-b61d-8fb23809679d
2019-10-31 14:51:05    7.0 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_2a57f4e2-ffa0-4be3-9c3f-28b16d86ed5a
2019-10-31 14:51:05    9.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_34381db1-00ca-4092-bd65-ab04e06dc799
2019-10-31 14:51:05    7.5 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_354a2bc1-345f-4996-9073-096cb863308d
2019-10-31 14:51:05    6.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_42da4cfd-6e21-40a1-8152-0b902da385a1
```

Step 3: Use INSERT INTO to add data

In Step 2, you used CTAS to create a table with partitions for the years 2015 to 2019. However, the original dataset also contains data for the years 2010 to 2014. Now you add that data using an INSERT INTO statement.

To add data to the table using one or more INSERT INTO statements

1. Run the following INSERT INTO command, specifying the years before 2015 in the WHERE clause.

```
INSERT INTO new_parquet
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) < 2015
```

2. Run the `aws s3 ls` command again, using the following syntax.

```
aws s3 ls s3://amzn-s3-demo-bucket/optimized-data/
```

The output shows the new partitions.

```
      PRE year=2010/
      PRE year=2011/
      PRE year=2012/
      PRE year=2013/
      PRE year=2014/
      PRE year=2015/
      PRE year=2016/
      PRE year=2017/
      PRE year=2018/
      PRE year=2019/
```

3. To see the reduction in the size of the dataset obtained by using compression and columnar storage in Parquet format, run the following command.

```
aws s3 ls s3://amzn-s3-demo-bucket/optimized-data/ --recursive --human-readable --
summarize
```

The following results show that the size of the dataset after parquet with Snappy compression is 1.2 GB.

```
...
2020-01-22 18:12:02 2.8 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_f0182e6c-38f4-4245-afa2-9f5bfa8d6d8f
2020-01-22 18:11:59 3.7 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_fd9906b7-06cf-4055-a05b-f050e139946e
Total Objects: 300
     Total Size: 1.2 GiB
```

4. If more CSV data is added to original table, you can add that data to the parquet table by using INSERT INTO statements. For example, if you had new data for the year 2020, you could run the following INSERT INTO statement. The statement adds the data and the relevant partition to the `new_parquet` table.

```
INSERT INTO new_parquet
```

```
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) = 2020
```

> **Note**
>
> The INSERT INTO statement supports writing a maximum of 100 partitions to the destination table. However, to add more than 100 partitions, you can run multiple INSERT INTO statements. For more information, see Use CTAS and INSERT INTO to work around the 100 partition limit.

Step 4: Measure performance and cost differences

After you transform the data, you can measure the performance gains and cost savings by running the same queries on the new and old tables and comparing the results.

> **Note**
>
> For Athena per-query cost information, see Amazon Athena pricing.

To measure performance gains and cost differences

1. Run the following query on the original table. The query finds the number of distinct IDs for every value of the year.

```
SELECT substr("date",1,4) as year,
       COUNT(DISTINCT id)
FROM original_csv
GROUP BY 1 ORDER BY 1 DESC
```

2. Note the time that the query ran and the amount of data scanned.
3. Run the same query on the new table, noting the query runtime and amount of data scanned.

```
SELECT year,
  COUNT(DISTINCT id)
FROM new_parquet
GROUP BY 1 ORDER BY 1 DESC
```

4. Compare the results and calculate the performance and cost difference. The following sample results show that the test query on the new table was faster and cheaper than the query on the old table.

Table	Runtime	Data scanned
Original	16.88 seconds	11.35 GB
New	3.79 seconds	428.05 MB

5. Run the following sample query on the original table. The query calculates the average maximum temperature (Celsius), average minimum temperature (Celsius), and average rainfall (mm) for the Earth in 2018.

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
FROM original_csv
WHERE element IN ('TMIN', 'TMAX', 'PRCP') AND substr("date",1,4) = '2018'
GROUP BY 1
```

6. Note the time that the query ran and the amount of data scanned.
7. Run the same query on the new table, noting the query runtime and amount of data scanned.

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
FROM new_parquet
WHERE element IN ('TMIN', 'TMAX', 'PRCP') and year = '2018'
GROUP BY 1
```

8. Compare the results and calculate the performance and cost difference. The following sample results show that the test query on the new table was faster and cheaper than the query on the old table.

Table	Runtime	Data scanned
Original	18.65 seconds	11.35 GB
New	1.92 seconds	68 MB

Summary

This topic showed you how to perform ETL operations using CTAS and INSERT INTO statements in Athena. You performed the first set of transformations using a CTAS statement that converted data to the Parquet format with Snappy compression. The CTAS statement also converted the dataset from non-partitioned to partitioned. This reduced its size and lowered the costs of running the queries. When new data becomes available, you can use an INSERT INTO statement to transform and load the data into the table that you created with the CTAS statement.

Use CTAS and INSERT INTO to work around the 100 partition limit

Athena has a limit of 100 partitions per `CREATE TABLE AS SELECT` (CTAS) query. Similarly, you can add a maximum of 100 partitions to a destination table with an INSERT INTO statement.

If you exceed this limitation, you may receive the error message HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets. To work around this limitation, you can use a CTAS statement and a series of `INSERT INTO` statements that create or insert up to 100 partitions each.

The example in this topic uses a database called `tpch100` whose data resides in the Amazon S3 bucket location s3://amzn-s3-demo-bucket/.

To use CTAS and INSERT INTO to create a table of more than 100 partitions

1. Use a `CREATE EXTERNAL TABLE` statement to create a table partitioned on the field that you want.

   The following example statement partitions the data by the column `l_shipdate`. The table has 2525 partitions.

   ```
   CREATE EXTERNAL TABLE `tpch100.lineitem_parq_partitioned`(
     `l_orderkey` int,
   ```

```
  `l_partkey` int,
  `l_suppkey` int,
  `l_linenumber` int,
  `l_quantity` double,
  `l_extendedprice` double,
  `l_discount` double,
  `l_tax` double,
  `l_returnflag` string,
  `l_linestatus` string,
  `l_commitdate` string,
  `l_receiptdate` string,
  `l_shipinstruct` string,
  `l_comment` string)
PARTITIONED BY (
  `l_shipdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe' STORED AS
 INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat' OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat' LOCATION
 's3://amzn-s3-demo-bucket/lineitem/'
```

2. Run a `SHOW PARTITIONS` *<table_name>* command like the following to list the partitions.

```
SHOW PARTITIONS lineitem_parq_partitioned
```

Following are partial sample results.

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1998-11-24
l_shipdate=1998-11-25
l_shipdate=1998-11-26
l_shipdate=1998-11-27
l_shipdate=1998-11-28
l_shipdate=1998-11-29
```

```
l_shipdate=1998-11-30
l_shipdate=1998-12-01
*/
```

3. Run a CTAS query to create a partitioned table.

   The following example creates a table called `my_lineitem_parq_partitioned` and uses the `WHERE` clause to restrict the `DATE` to earlier than `1992-02-01`. Because the sample dataset starts with January 1992, only partitions for January 1992 are created.

```
CREATE table my_lineitem_parq_partitioned
WITH (partitioned_by = ARRAY['l_shipdate']) AS
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
         l_receiptdate,
         l_shipinstruct,
         l_comment,
         l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) < DATE ('1992-02-01');
```

4. Run the `SHOW PARTITIONS` command to verify that the table contains the partitions that you want.

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

   The partitions in the example are from January 1992.

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
```

```
l_shipdate=1992-01-06
l_shipdate=1992-01-07
l_shipdate=1992-01-08
l_shipdate=1992-01-09
l_shipdate=1992-01-10
l_shipdate=1992-01-11
l_shipdate=1992-01-12
l_shipdate=1992-01-13
l_shipdate=1992-01-14
l_shipdate=1992-01-15
l_shipdate=1992-01-16
l_shipdate=1992-01-17
l_shipdate=1992-01-18
l_shipdate=1992-01-19
l_shipdate=1992-01-20
l_shipdate=1992-01-21
l_shipdate=1992-01-22
l_shipdate=1992-01-23
l_shipdate=1992-01-24
l_shipdate=1992-01-25
l_shipdate=1992-01-26
l_shipdate=1992-01-27
l_shipdate=1992-01-28
l_shipdate=1992-01-29
l_shipdate=1992-01-30
l_shipdate=1992-01-31
*/
```

5. Use an `INSERT INTO` statement to add partitions to the table.

   The following example adds partitions for the dates from the month of February 1992.

```
INSERT INTO my_lineitem_parq_partitioned
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
```

```
        l_receiptdate,
        l_shipinstruct,
        l_comment,
        l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) >= DATE ('1992-02-01')
AND cast(l_shipdate as timestamp) < DATE ('1992-03-01');
```

6. Run `SHOW PARTITIONS` again.

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

The sample table now has partitions from both January and February 1992.

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1992-02-20
l_shipdate=1992-02-21
l_shipdate=1992-02-22
l_shipdate=1992-02-23
l_shipdate=1992-02-24
l_shipdate=1992-02-25
l_shipdate=1992-02-26
l_shipdate=1992-02-27
l_shipdate=1992-02-28
l_shipdate=1992-02-29
*/
```

7. Continue using `INSERT INTO` statements that read and add no more than 100 partitions each. Continue until you reach the number of partitions that you require.

Important

When setting the WHERE condition, be sure that the queries don't overlap. Otherwise, some partitions might have duplicated data.

Use SerDes

Athena supports several SerDe (Serializer/Deserializer) libraries that parse data from a variety of data formats. When you create a table in Athena, you can specify a SerDe that corresponds to the format that your data is in. Athena does not support custom SerDes.

Athena can use SerDe libraries to create tables from CSV, TSV, custom-delimited, and JSON formats; data from the Hadoop-related formats ORC, Avro, and Parquet; logs from Logstash, AWS CloudTrail logs, and Apache WebServer logs. Each of these data formats has one or more serializer-deserializer (SerDe) libraries that Athena can use to parse the data.

Note

The formats listed in this section are used by Athena for reading data. For information about formats that Athena uses for writing data when it runs CTAS queries, see Create a table from query results (CTAS).

Topics

Choose a SerDe for your data

The following table lists the data formats supported in Athena and their corresponding SerDe libraries.

Supported data formats and SerDes

Data format	Description	SerDe types supported in Athena
Amazon Ion	Amazon Ion is a richly-typed, self-describing data format that is a superset of JSON, developed and open-sourced by Amazon.	Use the Amazon Ion Hive SerDe.
Apache Avro	A format for storing data in Hadoop that uses JSON-base d schemas for record values.	Use the Avro SerDe.
Apache Parquet	A format for columnar storage of data in Hadoop.	Use the Parquet SerDe and SNAPPY compression.
Apache WebServer logs	A format for storing logs in Apache WebServer.	Use the Grok SerDe or Regex SerDe.
CloudTrail logs	A format for storing logs in CloudTrail.	• Use the Hive JSON SerDe. For more information, see Query AWS CloudTrail logs.
CSV (Comma-Separated Values)	For data in CSV, each line represents a data record, and each record consists of one or more fields, separated by commas.	• Use the Lazy Simple SerDe for CSV, TSV, and custom-delimited files if your data does not include values enclosed in quotes or if it uses the `java.sql.Timestamp` format. • Use the Open CSV SerDe for processing CSV when

Data format	Description	SerDe types supported in Athena
		your data includes quotes in values or uses the UNIX numeric format for `TIMESTAMP` (for example, `1564610311`).
Custom-Delimited	For data in this format, each line represents a data record, and records are separated by a custom single-character delimiter.	Use the Lazy Simple SerDe for CSV, TSV, and custom-delimited files and specify a custom single-character delimiter.
JSON (JavaScript Object Notation)	For JSON data, each line represents a data record, and each record consists of attribute-value pairs and arrays, separated by commas.	• Use the Hive JSON SerDe. • Use the OpenX JSON SerDe.
Logstash logs	A format for storing logs in Logstash.	Use the Grok SerDe.
ORC (Optimized Row Columnar)	A format for optimized columnar storage of Hive data.	Use the ORC SerDe and ZLIB compression.
TSV (Tab-Separated Values)	For data in TSV, each line represents a data record, and each record consists of one or more fields, separated by tabs.	Use the Lazy Simple SerDe for CSV, TSV, and custom-delimited files and specify the separator character as `FIELDS TERMINATED BY '\t'`.

Use a SerDe to create a table

To use a SerDe when creating a table in Athena, use one of the following methods:

- Specify `ROW FORMAT DELIMITED` and then use DDL statements to specify field delimiters, as in the following example. When you specify `ROW FORMAT DELIMITED`, Athena uses the LazySimpleSerDe by default.

  ```
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  ESCAPED BY '\\'
  COLLECTION ITEMS TERMINATED BY '|'
  MAP KEYS TERMINATED BY ':'
  ```

  For examples of `ROW FORMAT DELIMITED`, see the following topics:

  Lazy Simple SerDe for CSV, TSV, and custom-delimited files

  Query Amazon CloudFront logs

  Query Amazon EMR logs

  Query Amazon VPC flow logs

  Use CTAS and INSERT INTO for ETL and data analysis

- Use `ROW FORMAT SERDE` to explicitly specify the type of SerDe that Athena should use when it reads and writes data to the table. The following example specifies the LazySimpleSerDe. To specify the delimiters, use `WITH SERDEPROPERTIES`. The properties specified by `WITH SERDEPROPERTIES` correspond to the separate statements (like `FIELDS TERMINATED BY`) in the `ROW FORMAT DELIMITED` example.

  ```
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
  WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ',',
  'collection.delim' = '|',
  'mapkey.delim' = ':',
  'escape.delim' = '\\'
  )
  ```

  For examples of `ROW FORMAT SERDE`, see the following topics:

  Avro SerDe

- Grok SerDe
- JSON SerDe libraries
- Open CSV SerDe for processing CSV
- Regex SerDe

Amazon Ion Hive SerDe

You can use the Amazon Ion Hive SerDe to query data stored in Amazon Ion format. Amazon Ion is a richly-typed, self-describing, open source data format. The Amazon Ion format is used by services such as Amazon Quantum Ledger Database (Amazon QLDB) and in the open source SQL query language PartiQL.

Amazon Ion has binary and text formats that are interchangeable. This feature combines the ease of use of text with the efficiency of binary encoding.

To query Amazon Ion data from Athena, you can use the Amazon Ion Hive SerDe, which serializes and deserializes Amazon Ion data. Deserialization allows you to run queries on the Amazon Ion data or read it for writing out into a different format like Parquet or ORC. Serialization lets you generate data in the Amazon Ion format by using `CREATE TABLE AS SELECT` (CTAS) or `INSERT INTO` queries to copy data from existing tables.

> **Note**
>
> Because Amazon Ion is a superset of JSON, you can use the Amazon Ion Hive SerDe to query non-Amazon Ion JSON datasets. Unlike other JSON SerDe libraries, the Amazon Ion SerDe does not expect each row of data to be on a single line. This feature is useful if you want to query JSON datasets that are in "pretty print" format or otherwise break up the fields in a row with newline characters.

For additional information and examples of querying Amazon Ion with Athena, see Analyze Amazon Ion datasets using Amazon Athena.

Serialization library name

The serialization library name for the Amazon Ion SerDe is `com.amazon.ionhiveserde.IonHiveSerDe`. For source code information, see Amazon Ion Hive SerDe on GitHub.com.

Considerations and limitations

- **Duplicated fields** – Amazon Ion structs are ordered and support duplicated fields, while Hive's `STRUCT<>` and `MAP<>` do not. Thus, when you deserialize a duplicated field from an Amazon Ion struct, a single value is chosen non deterministically, and the others are ignored.
- **External symbol tables unsupported** – Currently, Athena does not support external symbol tables or the following Amazon Ion Hive SerDe properties:
  - `ion.catalog.class`
  - `ion.catalog.file`
  - `ion.catalog.url`
  - `ion.symbol_table_imports`
- **File extensions** – Amazon Ion uses file extensions to determine which compression codec to use for deserializing Amazon Ion files. As such, compressed files must have the file extension that corresponds to the compression algorithm used. For example, if ZSTD is used, corresponding files should have the extension `.zst`.
- **Homogeneous data** – Amazon Ion has no restrictions on the data types that can be used for values in particular fields. For example, two different Amazon Ion documents might have a field with the same name that have different data types. However, because Hive uses a schema, all values that you extract to a single Hive column must have the same data type.
- **Map key type restrictions** – When you serialize data from another format into Amazon Ion, ensure that the map key type is one of `STRING`, `VARCHAR`, or `CHAR`. Although Hive allows you to use any primitive data type as a map key, Amazon Ion symbols must be a string type.
- **Union type** – Athena does not currently support the Hive union type.
- **Double data type** – Amazon Ion does not currently support the `double` data type.

Topics

- Create Amazon Ion tables
- Use CTAS and INSERT INTO to create Amazon Ion tables

- Amazon Ion SerDe property reference
- Use path extractors

Create Amazon Ion tables

To create a table in Athena from data stored in Amazon Ion format, you can use one of the following techniques in a CREATE TABLE statement:

- Specify `STORED AS ION`. In this usage, you do not have to specify the Amazon Ion Hive SerDe explicitly. This choice is the more straightforward option.
- Specify the Amazon Ion class paths in the `ROW FORMAT SERDE`, `INPUTFORMAT`, and `OUTPUTFORMAT` fields.

You can also use `CREATE TABLE AS SELECT` (CTAS) statements to create Amazon Ion tables in Athena. For information, see Use CTAS and INSERT INTO to create Amazon Ion tables.

Specify STORED AS ION

The following example `CREATE TABLE` statement uses `STORED AS ION` before the `LOCATION` clause to create a table based on flight data in Amazon Ion format. The `LOCATION` clause specifies the bucket or folder where the input files in Ion format are located. All files in the specified location are scanned.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
STORED AS ION
LOCATION 's3://amzn-s3-demo-bucket/'
```

Specify the Amazon Ion class paths

Instead of using the `STORED AS ION` syntax, you can explicitly specify the Ion class path values for the `ROW FORMAT SERDE`, `INPUTFORMAT`, and `OUTPUTFORMAT` clauses as follows.

Parameter	Ion class path
`ROW FORMAT SERDE`	`'com.amazon.ionhiveserde.IonHiveSerDe'`
`STORED AS INPUTFORMAT`	`'com.amazon.ionhiveserde.formats.IonInputFormat'`
`OUTPUTFORMAT`	`'com.amazon.ionhiveserde.formats.IonOutputFormat'`

The following DDL query uses this technique to create the same external table as in the previous example.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS INPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/'
```

For information about the SerDe properties for `CREATE TABLE` statements in Athena, see Amazon Ion SerDe property reference.

Use CTAS and INSERT INTO to create Amazon Ion tables

You can use the CREATE TABLE AS SELECT (CTAS) and INSERT INTO statements to copy or insert data from a table into a new table in Amazon Ion format in Athena.

In a CTAS query, specify format='ION' in the WITH clause, as in the following example.

```
CREATE TABLE new_table
WITH (format='ION')
AS SELECT * from existing_table
```

By default, Athena serializes Amazon Ion results in Ion binary format, but you can also use text format. To use text format, specify ion_encoding = 'TEXT' in the CTAS WITH clause, as in the following example.

```
CREATE TABLE new_table
WITH (format='ION', ion_encoding = 'TEXT')
AS SELECT * from existing_table
```

For more information about Amazon Ion specific properties in the CTAS WITH clause, see Amazon Ion properties for the CTAS WITH clause.

Amazon Ion properties for the CTAS WITH clause

In a CTAS query, you can use the WITH clause to specify the Amazon Ion format and optionally specify the Amazon Ion encoding and/or write compression algorithm to use.

format

You can specify the ION keyword as the format option in the WITH clause of a CTAS query. When you do so, the table that you create uses the format that you specify for IonInputFormat for reads, and it serializes data in the format that you specify for IonOutputFormat.

The following example specifies that the CTAS query use Amazon Ion format.

```
WITH (format='ION')
```

ion_encoding

Optional

Default: BINARY

Values: BINARY, TEXT

Specifies whether data is serialized in Amazon Ion binary format or Amazon Ion text format. The following example specifies Amazon Ion text format.

```
WITH (format='ION', ion_encoding='TEXT')
```

write_compression

Optional

Default: GZIP

Values: GZIP, ZSTD, BZIP2, SNAPPY, NONE

Specifies the compression algorithm to use to compress output files.

The following example specifies that the CTAS query write its output in Amazon Ion format using the Zstandard compression algorithm.

```
WITH (format='ION', write_compression = 'ZSTD')
```

For information about using compression in Athena, see Use compression in Athena.

For information about other CTAS properties in Athena, see CTAS table properties.

Amazon Ion SerDe property reference

This topic contains information about the SerDe properties for CREATE TABLE statements in Athena. For more information and examples of Amazon Ion SerDe property usage, see SerDe properties in the Amazon Ion Hive SerDe documentation on GitHub.

How to specify Amazon Ion SerDe properties

To specify properties for the Amazon Ion Hive SerDe in your CREATE TABLE statement, use the WITH SERDEPROPERTIES clause. Because WITH SERDEPROPERTIES is a subfield of the ROW FORMAT SERDE clause, you must specify ROW FORMAT SERDE and the Amazon Ion Hive SerDe class path first, as the following syntax shows.

```
...
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'property' = 'value',
 'property' = 'value',
...
)
```

Note that although the `ROW FORMAT SERDE` clause is required if you want to use `WITH SERDEPROPERTIES`, you can use either `STORED AS ION` or the longer `INPUTFORMAT` and `OUTPUTFORMAT` syntax to specify the Amazon Ion format.

Amazon Ion SerDe properties

Following are the Amazon Ion SerDe properties that can be used in `CREATE TABLE` statements in Athena.

ion.encoding

Optional

Default: `BINARY`

Values: `BINARY`, `TEXT`

This property declares whether new values added are serialized as [Amazon Ion binary](#) or Amazon Ion text format.

The following SerDe property example specifies Amazon Ion text format.

```
'ion.encoding' = 'TEXT'
```

ion.fail_on_overflow

Optional

Default: `true`

Values: `true`, `false`

Amazon Ion allows for arbitrarily large numerical types while Hive does not. By default, the SerDe fails if the Amazon Ion value does not fit the Hive column, but you can use the `fail_on_overflow` configuration option to let the value overflow instead of failing.

This property can be set at either the table or column level. To specify it at the table level, specify `ion.fail_on_overflow` as in the following example. This sets the default behavior for all columns.

```
'ion.fail_on_overflow' = 'true'
```

To control a specific column, specify the column name between `ion` and `fail_on_overflow`, delimited by periods, as in the following example.

```
'ion.<column>.fail_on_overflow' = 'false'
```

ion.path_extractor.case_sensitive

Optional

Default: `false`

Values: `true`, `false`

Determines whether to treat Amazon Ion field names as case sensitive. When `false`, the SerDe ignores case parsing Amazon Ion field names.

For example, suppose you have a Hive table schema that defines a field `alias` in lower case and an Amazon Ion document with both an `alias` field and an `ALIAS` field, as in the following example.

```
-- Hive Table Schema
alias: STRING

-- Amazon Ion Document
{ 'ALIAS': 'value1'}
{ 'alias': 'value2'}
```

The following example shows SerDe properties and the resulting extracted table when case sensitivity is set to `false`:

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
'ion.path_extractor.case_sensitive' = 'false'

--Extracted Table
| alias    |
|----------|
| "value1" |
| "value2" |
```

The following example shows SerDe properties and the resulting extracted table when case sensitivity is set to `true`:

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
'ion.path_extractor.case_sensitive' = 'true'

--Extracted Table
| alias    |
|----------|
| "value2" |
```

In the second case, `value1` for the `ALIAS` field is ignored when case sensitivity is set to `true` and the path extractor is specified as `alias`.

ion.*<column>*.path_extractor

Optional

Default: NA

Values: String with search path

Creates a path extractor with the specified search path for the given column. Path extractors map Amazon Ion fields to Hive columns. If no path extractors are specified, Athena dynamically creates path extractors at run time based on column names.

The following example path extractor maps the `example_ion_field` to the `example_hive_column`.

```
'ion.example_hive_column.path_extractor' = '(example_ion_field)'
```

For more information about path extractors and search paths, see Use path extractors.

ion.timestamp.serialization_offset

Optional

Default: `'Z'`

Values: `OFFSET`, where `OFFSET` is represented as *<signal>*`hh:mm`. Example values: `01:00`, `+01:00`, `-09:30`, `Z` (UTC, same as 00:00)

Unlike Apache Hive timestamps, which have no built-in time zone and are stored as an offset from the UNIX epoch, Amazon Ion timestamps do have an offset. Use this property to specify the offset when you serialize to Amazon Ion.

The following example adds an offset of one hour.

```
'ion.timestamp.serialization_offset' = '+01:00'
```

ion.serialize_null

Optional

Default: `OMIT`

Values: `OMIT`, `UNTYPED`, `TYPED`

The Amazon Ion SerDe can be configured to either serialize or omit columns that have null values. You can choose to write out strongly typed nulls (`TYPED`) or untyped nulls (`UNTYPED`). Strongly typed nulls are determined based on the default Amazon Ion to Hive type mapping.

The following example specifies strongly typed nulls.

```
'ion.serialize_null'='TYPED'
```

ion.ignore_malformed

Optional

Default: `false`

Values: `true`, `false`

When `true`, ignores malformed entries or the whole file if the SerDe is unable to read it. For more information, see Ignore malformed in the documentation on GitHub.

ion.*<column>*.serialize_as

Optional

Default: Default type for the column.

Values: String containing Amazon Ion type

Determines the Amazon Ion data type in which a value is serialized. Because Amazon Ion and Hive types do not always have a direct mapping, a few Hive types have multiple valid data types for serialization. To serialize data as a non-default data type, use this property. For more information about type mapping, see the Amazon Ion Type mapping page on GitHub.

By default, binary Hive columns are serialized as Amazon Ion blobs, but they can also be serialized as an Amazon Ion clob (character large object). The following example serializes the column `example_hive_binary_column` as a clob.

```
'ion.example_hive_binary_column.serialize_as' = 'clob'
```

Use path extractors

Amazon Ion is a document style file format, but Apache Hive is a flat columnar format. You can use special Amazon Ion SerDe properties called `path extractors` to map between the two formats. Path extractors flatten the hierarchical Amazon Ion format, map Amazon Ion values to Hive columns, and can be used to rename fields.

Athena can generate the extractors for you, but you can also define your own extractors if necessary.

Topics

- Path extractor examples

Use Athena generated path extractors

By default, Athena searches for top level Amazon Ion values that match Hive column names and creates path extractors at runtime based on these matching values. If your Amazon Ion data format matches the Hive table schema, Athena dynamically generates the extractors for you, and you do not need to add any additional path extractors. These default path extractors are not stored in the table metadata.

The following example shows how Athena generates extractors based on column name.

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}

-- Example DDL
CREATE EXTERNAL TABLE example_schema2 (
    identification MAP<STRING, STRING>,
    alias STRING
)
STORED AS ION
LOCATION 's3://amzn-s3-demo-bucket/path_extraction1/'
```

The following example extractors are generated by Athena. The first extracts the `identification` field to the `identification` column, and the second extracts the `alias` field to the `alias` column.

```
'ion.identification.path_extractor' = '(identification)'
'ion.alias.path_extractor' = '(alias)'
```

The following example shows the extracted table.

```
|                   identification                    |  alias    |
```

```
|--------------------------------------------------|----------|
|{["name", "driver_license"],["John Smith", "XXXX"]} | "Johnny" |
```

Specify your own path extractors

If your Amazon Ion fields do not map neatly to Hive columns, you can specify your own path extractors. In the `WITH SERDEPROPERTIES` clause of your `CREATE TABLE` statement, use the following syntax.

```
WITH SERDEPROPERTIES (
   "ion.path_extractor.case_sensitive" = "<Boolean>", 
   "ion.<column_name>.path_extractor" = "<path_extractor_expression>"
)
```

> **Note**
>
> By default, path extractors are case insensitive. To override this setting, set the ion.path_extractor.case_sensitive SerDe property to `true`.

Use search paths in path extractors

The SerDe property syntax for path extractor contains a *<path_extractor_expression>*:

```
"ion.<column_name>.path_extractor" = "<path_extractor_expression>"
```

You can use the *<path_extractor_expression>* to specify a search path that parses the Amazon Ion document and finds matching data. The search path is enclosed in parenthesis and can contain one or more of the following components separated by spaces.

- **Wild card** – Matches all values.
- **Index** – Matches the value at the specified numerical index. Indices are zero-based.
- **Text** – Matches all values whose field names match are equivalent to the specified text.
- **Annotations** – Matches values specified by a wrapped path component that has the annotations specified.

The following example shows an Amazon Ion document and some example search paths.

```
-- Amazon Ion document
{
    foo: ["foo1", "foo2"] ,
    bar: "myBarValue",
    bar: A::"annotatedValue"
}

-- Example search paths
(foo 0)       # matches "foo1"
(1)           # matches "myBarValue"
(*)           # matches ["foo1", "foo2"], "myBarValue" and A::"annotatedValue"
()            # matches {foo: ["foo1", "foo2"] , bar: "myBarValue", bar:
 A::"annotatedValue"}
(bar)         # matches "myBarValue" and A::"annotatedValue"
(A::bar)      # matches A::"annotatedValue"
```

Path extractor examples

The following path extractor examples show how to flatten and rename fields or extract data as Amazon Ion text.

Flatten and rename fields

The following example shows a set of search paths that flatten and rename fields. The example uses search paths to do the following:

- Map the `nickname` column to the `alias` field
- Map the `name` column to the `name` subfield located in the `identification` struct.

Following is the example Amazon Ion document.

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}
```

The following is the example CREATE TABLE statement that defines the path extractors.

```
-- Example DDL Query
CREATE EXTERNAL TABLE example_schema2 (
    name STRING,
    nickname STRING
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'ion.nickname.path_extractor' = '(alias)',
 'ion.name.path_extractor' = '(identification name)'
 )
STORED AS ION
LOCATION 's3://amzn-s3-demo-bucket/path_extraction2/'
```

The following example shows the extracted data.

```
-- Extracted Table
| name          |   nickname    |
|--------------|--------------|
| "John Smith" |  "Johnny"    |
```

For more information about search paths and additional search path examples, see the Ion Java Path Extraction page on GitHub.

Extract flight data to text format

The following example CREATE TABLE query uses WITH SERDEPROPERTIES to add path extractors to extract flight data and specify the output encoding as Amazon Ion text. The example uses the STORED AS ION syntax.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
```

```
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'ion.encoding' = 'TEXT',
 'ion.yr.path_extractor'='(year)',
 'ion.quarter.path_extractor'='(results quarter)',
 'ion.month.path_extractor'='(date month)')
STORED AS ION
LOCATION 's3://amzn-s3-demo-bucket/'
```

Avro SerDe

Use the Avro SerDe to create Athena tables from Avro data.

Serialization library name

The serialization library name for the Avro SerDe is `org.apache.hadoop.hive.serde2.avro.AvroSerDe`. For technical information, see AvroSerDe in the Apache documentation.

Use the Avro SerDe

For security reasons, Athena does not support using `avro.schema.url` to specify table schema; use `avro.schema.literal` instead.

To extract schema from data in Avro format, use the Apache `avro-tools-<version>.jar` file located in the `java` subdirectory of your installed Avro release. Use the `getschema` parameter to return a schema that you can use in your `WITH SERDEPROPERTIES` statement, as in the following example.

```
java -jar avro-tools-1.8.2.jar getschema my_data.avro
```

To download Avro, see Apache Avro releases. To download Apache Avro Tools directly, see the Apache Avro tools Maven repository.

After you obtain the schema, use a `CREATE TABLE` statement to create an Athena table based on the underlying Avro data stored in Amazon S3. To specify the Avro SerDe in your `CREATE TABLE` statement, use `ROW FORMAT SERDE` 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'. Specify the schema using the WITH SERDEPROPERTIES clause, as in the following example.

Note

Replace *myregion* in s3://athena-examples-*myregion*/path/to/data/ with the region identifier where you run Athena, for example, s3://athena-examples-us-west-1/path/to/data/.

```
CREATE EXTERNAL TABLE flights_avro_example (
   yr INT,
   flightdate STRING,
   uniquecarrier STRING,
   airlineid INT,
   carrier STRING,
   flightnum STRING,
   origin STRING,
   dest STRING,
   depdelay INT,
   carrierdelay INT,
   weatherdelay INT
)
PARTITIONED BY (year STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='
{
   "type" : "record",
   "name" : "flights_avro_subset",
   "namespace" : "default",
   "fields" : [ {
      "name" : "yr",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "flightdate",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "uniquecarrier",
      "type" : [ "null", "string" ],
      "default" : null
```

```
    }, {
       "name" : "airlineid",
       "type" : [ "null", "int" ],
       "default" : null
    }, {
       "name" : "carrier",
       "type" : [ "null", "string" ],
       "default" : null
    }, {
       "name" : "flightnum",
       "type" : [ "null", "string" ],
       "default" : null
    }, {
       "name" : "origin",
       "type" : [ "null", "string" ],
       "default" : null
    }, {
       "name" : "dest",
       "type" : [ "null", "string" ],
       "default" : null
    }, {
       "name" : "depdelay",
       "type" : [ "null", "int" ],
       "default" : null
    }, {
       "name" : "carrierdelay",
       "type" : [ "null", "int" ],
       "default" : null
    }, {
       "name" : "weatherdelay",
       "type" : [ "null", "int" ],
       "default" : null
     } ]
}
')
STORED AS AVRO
LOCATION 's3://athena-examples-myregion/flight/avro/';
```

Run the `MSCK REPAIR TABLE` statement on the table to refresh partition metadata.

```
MSCK REPAIR TABLE flights_avro_example;
```

Query the top 10 departure cities by number of total departures.

```
SELECT origin, count(*) AS total_departures
FROM flights_avro_example
WHERE year >= '2000'
GROUP BY origin
ORDER BY total_departures DESC
LIMIT 10;
```

> **Note**
>
> The flight table data comes from Flights provided by US Department of Transportation, Bureau of Transportation Statistics. Desaturated from original.

Grok SerDe

The Logstash Grok SerDe is a library with a set of specialized patterns for deserialization of unstructured text data, usually logs. Each Grok pattern is a named regular expression. You can identify and re-use these deserialization patterns as needed. This makes it easier to use Grok compared with using regular expressions. Grok provides a set of pre-defined patterns. You can also create custom patterns.

Serialization library name

The serialization library name for the Grok SerDe is `com.amazonaws.glue.serde.GrokSerDe`.

How to use the Grok SerDe

To specify the Grok SerDe when creating a table in Athena, use the `ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'` clause, followed by the `WITH SERDEPROPERTIES` clause that specifies the patterns to match in your data, where:

- The `input.format` expression defines the patterns to match in the data. This is required.
- The `input.grokCustomPatterns` expression defines a named custom pattern, which you can subsequently use within the `input.format` expression. This is optional. To include multiple pattern entries into the `input.grokCustomPatterns` expression, use the newline escape character (\n) to separate them, as follows: `'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)')`.
- The `STORED AS INPUTFORMAT` and `OUTPUTFORMAT` clauses are required.

- The `LOCATION` clause specifies an Amazon S3 bucket, which can contain multiple data objects. All data objects in the bucket are deserialized to create the table.

Examples

The examples in this section rely on the list of predefined Grok patterns. For more information, see grok-patterns on GitHub.com.

Example 1

This example uses source data from Postfix maillog entries saved in `s3://amzn-s3-demo-bucket/groksample/`.

```
Feb  9 07:15:00 m4eastmail postfix/smtpd[19305]: B88C4120838: connect from
 unknown[192.168.55.4]
Feb  9 07:15:00 m4eastmail postfix/smtpd[20444]: B58C4330038:
 client=unknown[192.168.55.4]
Feb  9 07:15:03 m4eastmail postfix/cleanup[22835]: BDC22A77854: message-
id=<31221401257553.5004389LCBF@m4eastmail.example.com>
```

The following statement creates a table in Athena called `mygroktable` from the source data, using a custom pattern and the predefined patterns that you specify:

```
CREATE EXTERNAL TABLE `mygroktable`(
   syslogbase string,
   queue_id string,
   syslog_message string
   )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.grokCustomPatterns' = 'POSTFIX_QUEUEID [0-9A-F]{7,12}',
   'input.format'='%{SYSLOGBASE} %{POSTFIX_QUEUEID:queue_id}:
 %{GREEDYDATA:syslog_message}'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://amzn-s3-demo-bucket/groksample/';
```

Start with a pattern like %{NOTSPACE:column} to get the columns mapped first, and then specialize the columns if needed.

Example 2

In the following example, you create a query for Log4j logs. The example logs have the entries in this format:

```
2017-09-12 12:10:34,972 INFO  - processType=AZ, processId=ABCDEFG614B6F5E49,
 status=RUN,
threadId=123:amqListenerContainerPool23P:AJ|ABCDE9614B6F5E49||
2017-09-12T12:10:11.172-0700],
executionTime=7290, tenantId=12456, userId=123123f8535f8d76015374e7a1d87c3c,
 shard=testapp1,
jobId=12312345e5e7df0015e777fb2e03f3c, messageType=REAL_TIME_SYNC,
action=receive, hostname=1.abc.def.com
```

To query this log data:

- Add the Grok pattern to the input.format for each column. For example, for timestamp, add %{TIMESTAMP_ISO8601:timestamp}. For loglevel, add %{LOGLEVEL:loglevel}.
- Make sure the pattern in input.format matches the format of the log exactly, by mapping the dashes (-) and the commas that separate the entries in the log format.

```
CREATE EXTERNAL TABLE bltest (
 timestamp STRING,
 loglevel STRING,
 processtype STRING,
 processid STRING,
 status STRING,
 threadid STRING,
 executiontime INT,
 tenantid INT,
 userid STRING,
 shard STRING,
 jobid STRING,
 messagetype STRING,
 action STRING,
 hostname STRING
 )
ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
```

```
"input.grokCustomPatterns" = 'C_ACTION receive|send',
"input.format" = "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:loglevel} - processType=
%{NOTSPACE:processtype}, processId=%{NOTSPACE:processid}, status=%{NOTSPACE:status},
 threadId=%{NOTSPACE:threadid}, executionTime=%{POSINT:executiontime}, tenantId=
%{POSINT:tenantid}, userId=%{NOTSPACE:userid}, shard=%{NOTSPACE:shard}, jobId=
%{NOTSPACE:jobid}, messageType=%{NOTSPACE:messagetype}, action=%{C_ACTION:action},
 hostname=%{HOST:hostname}"
) STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/samples/';
```

Example 3

The following example Amazon S3 server access logs `CREATE TABLE` statement shows the `'input.grokCustomPatterns'` expression that contains two pattern entries, separated by the newline escape character (\n), as shown in this snippet from the example query: `'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\]]*)'`.

```
CREATE EXTERNAL TABLE `s3_access_auto_raw_02`(
  `bucket_owner` string COMMENT 'from deserializer',
  `bucket` string COMMENT 'from deserializer',
  `time` string COMMENT 'from deserializer',
  `remote_ip` string COMMENT 'from deserializer',
  `requester` string COMMENT 'from deserializer',
  `request_id` string COMMENT 'from deserializer',
  `operation` string COMMENT 'from deserializer',
  `key` string COMMENT 'from deserializer',
  `request_uri` string COMMENT 'from deserializer',
  `http_status` string COMMENT 'from deserializer',
  `error_code` string COMMENT 'from deserializer',
  `bytes_sent` string COMMENT 'from deserializer',
  `object_size` string COMMENT 'from deserializer',
  `total_time` string COMMENT 'from deserializer',
  `turnaround_time` string COMMENT 'from deserializer',
  `referrer` string COMMENT 'from deserializer',
  `user_agent` string COMMENT 'from deserializer',
  `version_id` string COMMENT 'from deserializer')
ROW FORMAT SERDE
  'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
```

```
  'input.format'='%{NOTSPACE:bucket_owner} %{NOTSPACE:bucket} \
\[%{INSIDE_BRACKETS:time}\\] %{NOTSPACE:remote_ip} %{NOTSPACE:requester}
 %{NOTSPACE:request_id} %{NOTSPACE:operation} %{NOTSPACE:key} \"?
%{INSIDE_QS:request_uri}\"? %{NOTSPACE:http_status} %{NOTSPACE:error_code}
 %{NOTSPACE:bytes_sent} %{NOTSPACE:object_size} %{NOTSPACE:total_time}
 %{NOTSPACE:turnaround_time} \"?%{INSIDE_QS:referrer}\"? \"?%{INSIDE_QS:user_agent}\"?
 %{NOTSPACE:version_id}',
  'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket'
```

See also

- Understanding Grok Patterns (external website)
- Built-in patterns (*AWS Glue User Guide*)

JSON SerDe libraries

In Athena, you can use SerDe libraries to deserialize JSON data. Deserialization converts the JSON data so that it can be serialized (written out) into a different format like Parquet or ORC.

- Hive JSON SerDe
- OpenX JSON SerDe
- Amazon Ion Hive SerDe

> **Note**
>
> The Hive and OpenX libraries expect JSON data to be on a single line (not formatted), with records separated by a new line character.

Because Amazon Ion is a superset of JSON, you can use the Amazon Ion Hive SerDe to query non-Amazon Ion JSON datasets. Unlike the Hive and OpenX JSON SerDe libraries, the Amazon Ion SerDe does not expect each row of data to be on a single line. This feature is useful if you want to query JSON datasets that are in "pretty print" format or otherwise break up the fields in a row with newline characters.

Library names

Use one of the following:

org.apache.hive.hcatalog.data.JsonSerDe

org.openx.data.jsonserde.JsonSerDe

com.amazon.ionhiveserde.IonHiveSerDe

Hive JSON SerDe

The Hive JSON SerDe is commonly used to process JSON data like events. These events are represented as single-line strings of JSON-encoded text separated by a new line. The Hive JSON SerDe does not allow duplicate keys in `map` or `struct` key names.

> **Note**
>
> The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

The following example DDL statement uses the Hive JSON SerDe to create a table based on sample online advertising data. In the `LOCATION` clause, replace the *`myregion`* in `s3://amzn-s3-demo-bucket.elasticmapreduce/samples/hive-ads/tables/impressions` with the region identifier where you run Athena (for example, `s3://us-west-2.elasticmapreduce/samples/hive-ads/tables/impressions`).

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionid string,
    referrer string,
```

```
    useragent string,
    usercookie string,
    ip string,
    number string,
    processid string,
    browsercookie string,
    requestendtime string,
    timers struct
                <
                 modellookup:string,
                 requesttime:string
                >,
    threadid string,
    hostname string,
    sessionid string
)
PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

Specify timestamp formats with the Hive JSON SerDe

To parse timestamp values from string, you can add the `WITH SERDEPROPERTIES` subfield to the `ROW FORMAT SERDE` clause and use it to specify the `timestamp.formats` parameter. In the parameter, specify a comma-separated list of one or more timestamp patterns, as in the following example:

```
...
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
WITH SERDEPROPERTIES ("timestamp.formats"="yyyy-MM-dd'T'HH:mm:ss.SSS'Z',yyyy-MM-
dd'T'HH:mm:ss")
...
```

For more information, see Timestamps in the Apache Hive documentation.

Load the table for querying

After you create the table, run MSCK REPAIR TABLE to load the table and make it queryable from Athena:

```
MSCK REPAIR TABLE impressions
```

Query CloudTrail logs

You can use the Hive JSON SerDe to query CloudTrail logs. For more information and example `CREATE TABLE` statements, see Query AWS CloudTrail logs.

OpenX JSON SerDe

Like the Hive JSON SerDe, you can use the OpenX JSON to process JSON data. The data are also represented as single-line strings of JSON-encoded text separated by a new line. Like the Hive JSON SerDe, the OpenX JSON SerDe does not allow duplicate keys in `map` or `struct` key names.

Considerations and limitations

- When using the OpenX JSON SerDe, the number of results and their values can be non-deterministic. The results can contain more rows than expected, fewer rows than expected, or unexpected null values when none are present in the underlying data. To work around this issue, use the Hive JSON SerDe, or rewrite the data to another file format type.
- The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it.

  For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

Optional properties

Unlike the Hive JSON SerDe, the OpenX JSON SerDe also has the following optional SerDe properties that can be useful for addressing inconsistencies in data.

ignore.malformed.json

Optional. When set to `TRUE`, lets you skip malformed JSON syntax. The default is `FALSE`.

dots.in.keys

Optional. The default is `FALSE`. When set to `TRUE`, allows the SerDe to replace the dots in key names with underscores. For example, if the JSON dataset contains a key with the name `"a.b"`, you can use this property to define the column name to be `"a_b"` in Athena. By default (without this SerDe), Athena does not allow dots in column names.

case.insensitive

Optional. The default is TRUE. When set to TRUE, the SerDe converts all uppercase columns to lowercase.

To use case-sensitive key names in your data, use WITH SERDEPROPERTIES ("case.insensitive"= FALSE;). Then, for every key that is not already all lowercase, provide a mapping from the column name to the property name using the following syntax:

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.userid" = "userId")
```

If you have two keys like URL and Url that are the same when they are in lowercase, an error like the following can occur:

HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Duplicate key "url"

To resolve this, set the case.insensitive property to FALSE and map the keys to different names, as in the following example:

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.url1" = "URL",
 "mapping.url2" = "Url")
```

mapping

Optional. Maps column names to JSON keys that aren't identical to the column names. The mapping parameter is useful when the JSON data contains keys that are keywords. For example, if you have a JSON key named timestamp, use the following syntax to map the key to a column named ts:

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.ts" = "timestamp")
```

Mapping nested field names with colons to Hive-compatible names

If you have a field name with colons inside a struct, you can use the mapping property to map the field to a Hive-compatible name. For example, if your column type definitions contain my:struct:field:string, you can map the definition to my_struct_field:string by including the following entry in WITH SERDEPROPERTIES:

```
("mapping.my_struct_field" = "my:struct:field")
```

The following example shows the corresponding `CREATE TABLE` statement.

```
CREATE EXTERNAL TABLE colon_nested_field (
item struct<my_struct_field:string>)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.my_struct_field" = "my:struct:field")
```

Example: advertising data

The following example DDL statement uses the OpenX JSON SerDe to create a table based on the same sample online advertising data used in the example for the Hive JSON SerDe. In the `LOCATION` clause, replace *`myregion`* with the region identifier where you run Athena.

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionId string,
    referrer string,
    useragent string,
    usercookie string,
    ip string,
    number string,
    processid string,
    browsercokie string,
    requestendtime string,
    timers struct<
       modellookup:string,
       requesttime:string>,
    threadid string,
    hostname string,
    sessionid string
)   PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

Example: deserializing nested JSON

You can use the JSON SerDes to parse more complex JSON-encoded data. This requires using `CREATE TABLE` statements that use `struct` and `array` elements to represent nested structures.

The following example creates an Athena table from JSON data that has nested structures. The example has the following structure:

```
{
"DocId": "AWS",
"User": {
        "Id": 1234,
        "Username": "carlos_salazar",
        "Name": "Carlos",
"ShippingAddress": {
"Address1": "123 Main St.",
"Address2": null,
"City": "Anytown",
"State": "CA"
   },
"Orders": [
   {
     "ItemId": 6789,
     "OrderDate": "11/11/2022"
   },
   {
     "ItemId": 4352,
     "OrderDate": "12/12/2022"
   }
  ]
 }
}
```

Remember that the OpenX SerDe expects each JSON record to be on a single line of text. When stored in Amazon S3, all of the data in the preceding example should be on a single line, like this:

```
{"DocId":"AWS","User":
{"Id":1234,"Username":"carlos_salazar","Name":"Carlos","ShippingAddress" ...
```

The following `CREATE TABLE` statement uses the Openx-JsonSerDe with the `struct` and `array` collection data types to establish groups of objects for the example data.

```
CREATE external TABLE complex_json (
   docid string,
   `user` struct<
               id:INT,
               username:string,
               name:string,
               shippingaddress:struct<
                                      address1:string,
                                      address2:string,
                                      city:string,
                                      state:string
                                      >,
               orders:array<
                            struct<
                                 itemid:INT,
                                  orderdate:string
                                  >
                              >
               >
   )
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket/myjsondata/';
```

To query the table, use a `SELECT` statement like the following.

```
SELECT
 user.name as Name,
 user.shippingaddress.address1 as Address,
 user.shippingaddress.city as City,
 o.itemid as Item_ID, o.orderdate as Order_date
FROM complex_json, UNNEST(user.orders) as temp_table (o)
```

To access the data fields inside structs, the sample query uses dot notation (for example, `user.name`). To access data inside an array of structs (as with the `orders` field), you can use the `UNNEST` function. The `UNNEST` function flattens the array into a temporary table (in this case called `o`). This lets you use the dot notation as you do with structs to access the unnested array elements (for example, `o.itemid`). The name `temp_table`, used in the example for illustrative purposes, is often abbreviated as `t`.

The following table shows the query results.

#	Name	Address	City	Item_ID	Order_date
1	Carlos	123 Main St.	Anytown	6789	11/11/2022
2	Carlos	123 Main St.	Anytown	4352	12/12/2022

Additional resources

For more information about working with JSON and nested JSON in Athena, see the following resources:

- Create tables in Amazon Athena from nested JSON and mappings using JSONSerDe (AWS Big Data Blog)
- I get errors when I try to read JSON data in Amazon Athena (AWS Knowledge Center article)
- hive-json-schema (GitHub) – Tool written in Java that generates `CREATE TABLE` statements from example JSON documents. The `CREATE TABLE` statements that are generated use the OpenX JSON Serde.

CSV SerDe libraries

When you create a table for CSV data in Athena, you can use either the Open CSV SerDe or the Lazy Simple SerDe library. To help you decide which to use, consider the following guidelines.

- If your data contains values enclosed in double quotes ("), you can use the Open CSV SerDe library to deserialize the values in Athena. If your data does not contain values enclosed in double quotes ("), you can omit specifying any SerDe. In this case, Athena uses the default Lazy Simple SerDe. For information, see Lazy Simple SerDe for CSV, TSV, and custom-delimited files.
- If your data has UNIX numeric `TIMESTAMP` values (for example, `1579059880000`), use the Open CSV SerDe. If your data uses the `java.sql.Timestamp` format, use the Lazy Simple SerDe.

Topics

Lazy Simple SerDe for CSV, TSV, and custom-delimited files

Because this is the default SerDe in Athena for data in CSV, TSV, and custom-delimited formats, specifying it is optional. In your `CREATE TABLE` statement, if you don't specify a SerDe and specify only `ROW FORMAT DELIMITED`, Athena uses this SerDe. Use this SerDe if your data does not have values enclosed in quotes.

For reference documentation about the Lazy Simple SerDe, see the Hive SerDe section of the Apache Hive Developer Guide.

Serialization library name

The serialization library name for the Lazy Simple SerDe is `org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`. For source code information, see LazySimpleSerDe.java on GitHub.com.

Ignoring headers

To ignore headers in your data when you define a table, you can use the `skip.header.line.count` table property, as in the following example.

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

For examples that ignore headers, see the `CREATE TABLE` statements in Query Amazon VPC flow logs and Query Amazon CloudFront logs.

CSV example

The following example shows how to use the `LazySimpleSerDe` library to create a table in Athena from CSV data. To deserialize custom-delimited files using this SerDe, follow the pattern in the examples but use the `FIELDS TERMINATED BY` clause to specify a different single-character delimiter. Lazy Simple SerDe does not support multi-character delimiters.

> **Note**
>
> Replace *`myregion`* in `s3://athena-examples-`*`myregion`*`/path/to/data/` with the region identifier where you run Athena, for example, `s3://athena-examples-us-west-1/path/to/data/`.

Use the CREATE TABLE statement to create an Athena table from the underlying data in CSV stored in Amazon S3.

```
CREATE EXTERNAL TABLE flight_delays_csv (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
```

```
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
```

```
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
    PARTITIONED BY (year STRING)
    ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      ESCAPED BY '\\'
      LINES TERMINATED BY '\n'
    LOCATION 's3://athena-examples-myregion/flight/csv/';
```

Run `MSCK REPAIR TABLE` to refresh partition metadata each time a new partition is added to this table:

```
MSCK REPAIR TABLE flight_delays_csv;
```

Query the top 10 routes delayed by more than 1 hour:

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_csv
```

```
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Note

The flight table data comes from Flights provided by US Department of Transportation, Bureau of Transportation Statistics. Desaturated from original.

TSV example

To create an Athena table from TSV data stored in Amazon S3, use `ROW FORMAT DELIMITED` and specify the `\t` as the tab field delimiter, `\n` as the line separator, and `\` as the escape character. The following excerpt shows this syntax. No sample TSV flight data is available in the `athena-examples` location, but as with the CSV table, you would run `MSCK REPAIR TABLE` to refresh partition metadata each time a new partition is added.

```
...
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
...
```

Open CSV SerDe for processing CSV

Use the Open CSV SerDe to create Athena tables from comma-separated data (CSV) data.

Serialization library name

The serialization library name for the Open CSV SerDe is `org.apache.hadoop.hive.serde2.OpenCSVSerde`. For source code information, see CSV SerDe in the Apache documentation.

Using the Open CSV SerDe

To use this SerDe, specify its fully qualified class name after `ROW FORMAT SERDE`. Also specify the delimiters inside `SERDEPROPERTIES`, as in the following example.

```
...
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "`",
  "escapeChar"    = "\\"
)
```

Ignore headers

To ignore headers in your data when you define a table, you can use the `skip.header.line.count` table property, as in the following example.

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

For examples, see the `CREATE TABLE` statements in Query Amazon VPC flow logs and Query Amazon CloudFront logs.

Considerations for string data

The Open CSV SerDe has the following characteristics for string data:

- Uses double quotes (") as the default quote character, and allows you to specify separator, quote, and escape characters, such as:

  ```
  WITH SERDEPROPERTIES ("separatorChar" = ",", "quoteChar" = "`", "escapeChar" = "\\" )
  ```
- You cannot escape `\t` or `\n` directly. To escape them, use `"escapeChar" = "\\"`. For an example, see Example: Escaping \t or \n.
- The Open CSV SerDe does not support embedded line breaks in CSV files.

Considerations for non-string data

For data types other than `STRING`, the Open CSV SerDe behaves as follows:

- Recognizes `BOOLEAN`, `BIGINT`, `INT`, and `DOUBLE` data types.
- Does not recognize empty or null values in columns defined as a numeric data type, leaving them as `string`. One workaround is to create the column with the null values as `string` and then use `CAST` to convert the field in a query to a numeric data type, supplying a default

value of `0` for nulls. For more information, see [When I query CSV data in Athena, I get the error HIVE_BAD_DATA: Error parsing field value] in the AWS Knowledge Center.

- For columns specified with the `timestamp` data type in your `CREATE TABLE` statement, recognizes `TIMESTAMP` data if it is specified in the UNIX numeric format in milliseconds, such as `1579059880000`. For an example, see [Example: Using the TIMESTAMP type and DATE type specified in the UNIX numeric format].
  - The Open CSV SerDe does not support `TIMESTAMP` in the JDBC-compliant `java.sql.Timestamp` format, such as `"YYYY-MM-DD HH:MM:SS.fffffffff"` (9 decimal place precision).
- For columns specified with the `DATE` data type in your `CREATE TABLE` statement, recognizes values as dates if the values represent the number of days that elapsed since January 1, 1970. For example, the value `18276` in a column with the `date` data type renders as `2020-01-15` when queried. In this UNIX format, each day is considered to have 86,400 seconds.
  - The Open CSV SerDe does not support `DATE` in any other format directly. To process timestamp data in other formats, you can define the column as `string` and then use time conversion functions to return the desired results in your `SELECT` query. For more information, see the article [When I query a table in Amazon Athena, the TIMESTAMP result is empty] in the [AWS knowledge center].
- To further convert columns to the desired type in a table, you can [create a view] over the table and use `CAST` to convert to the desired type.

Examples

Example Example: Querying simple CSV data

The following example assumes you have CSV data saved in the location `s3://amzn-s3-demo-bucket/mycsv/` with the following contents:

```
"a1","a2","a3","a4"
"1","2","abc","def"
"a","a1","abc3","ab4"
```

Use a `CREATE TABLE` statement to create an Athena table based on the data. Reference `OpenCSVSerde` (note the "d" in lower case) after `ROW FORMAT SERDE` and specify the character separator, quote character, and escape character in `WITH SERDEPROPERTIES`, as in the following example.

```
CREATE EXTERNAL TABLE myopencsvtable (
   col1 string,
   col2 string,
   col3 string,
   col4 string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   'separatorChar' = ',',
   'quoteChar' = '"',
   'escapeChar' = '\\'
   )
STORED AS TEXTFILE
LOCATION 's3://amzn-s3-demo-bucket/mycsv/';
```

Query all values in the table:

```
SELECT * FROM myopencsvtable;
```

The query returns the following values:

```
col1     col2    col3    col4
-----------------------------
a1       a2      a3      a4
1        2       abc     def
a        a1      abc3    ab4
```

Example Example: Using the TIMESTAMP type and DATE type specified in the UNIX numeric format

Consider the following three columns of comma-separated data. The values in each column are enclosed in double quotes.

```
"unixvalue creationdate 18276 creationdatetime 1579059880000","18276","1579059880000"
```

The following statement creates a table in Athena from the specified Amazon S3 bucket location.

```
CREATE EXTERNAL TABLE IF NOT EXISTS testtimestamp1(
 `profile_id` string,
 `creationdate` date,
 `creationdatetime` timestamp
```

```
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION 's3://amzn-s3-demo-bucket'
```

Next, run the following query:

```
SELECT * FROM testtimestamp1
```

The query returns the following result, showing the date and time data:

```
profile_id                                                     creationdate
 creationdatetime
unixvalue creationdate 18276 creationdatetime 1579146280000       2020-01-15
 2020-01-15 03:44:40.000
```

Example Example: Escaping \t or \n

Consider the following test data:

```
" \\t\\t\\n 123 \\t\\t\\n ",abc
" 456 ",xyz
```

The following statement creates a table in Athena, specifying that `"escapeChar" = "\\"`.

```
CREATE EXTERNAL TABLE test1 (
f1 string,
s2 string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("separatorChar" = ",", "escapeChar" = "\\")
LOCATION 's3://amzn-s3-demo-bucket/dataset/test1/'
```

Next, run the following query:

```
SELECT * FROM test1;
```

It returns this result, correctly escaping `\t` or `\n`:

```
f1            s2
\t\t\n 123 \t\t\n            abc
456                          xyz
```

ORC SerDe

Use the ORC SerDe to create Athena tables from ORC data.

Serialization library name

The serialization library for the ORC SerDe is `org.apache.hadoop.hive.ql.io.orc.OrcSerde`, but in your `CREATE TABLE` statements, you specify this with the clause `STORED AS ORC`. For source code information, see [OrcSerde.java](#) on GitHub.com.

Example: create a table for ORC flight data

> **Note**
>
> Replace *`myregion`* in `s3://athena-examples-`*`myregion`*`/path/to/data/` with the region identifier where you run Athena, for example, `s3://athena-examples-us-west-1/path/to/data/`.

The following example creates a table for the flight delays data in ORC. The table includes partitions:

```
DROP TABLE flight_delays_orc;
CREATE EXTERNAL TABLE flight_delays_orc (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
```

```
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
```

```
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
```

```
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year String)
STORED AS ORC
LOCATION 's3://athena-examples-myregion/flight/orc/'
tblproperties ("orc.compress"="ZLIB");
```

Run the `MSCK REPAIR TABLE` statement on the table to refresh partition metadata:

```
MSCK REPAIR TABLE flight_delays_orc;
```

Use this query to obtain the top 10 routes delayed by more than 1 hour:

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_orc
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Parquet SerDe

Use the Parquet SerDe to create Athena tables from Parquet data.

The Parquet SerDe is used for data stored in the Parquet format. To convert data into Parquet format, you can use CREATE TABLE AS SELECT (CTAS) queries. For more information, see Create a table from query results (CTAS), Examples of CTAS queries and Use CTAS and INSERT INTO for ETL and data analysis.

Serialization library name

The serialization library name for the Parquet SerDe is `org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe`. For source code information, see Class ParquetHiveSerDe in the Apache documentation.

Example: Query a file stored in parquet

> **Note**
>
> Replace *myregion* in `s3://athena-examples-myregion/path/to/data/` with the region identifier where you run Athena, for example, `s3://athena-examples-us-west-1/path/to/data/`.

Use the following `CREATE TABLE` statement to create an Athena table from the underlying data stored in Parquet format in Amazon S3:

```
CREATE EXTERNAL TABLE flight_delays_pq (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
```

```
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
```

```
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year STRING)
STORED AS PARQUET
LOCATION 's3://athena-examples-myregion/flight/parquet/'
tblproperties ("parquet.compression"="SNAPPY");
```

Run the `MSCK REPAIR TABLE` statement on the table to refresh partition metadata:

```
MSCK REPAIR TABLE flight_delays_pq;
```

Query the top 10 routes delayed by more than 1 hour:

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_pq
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Note

The flight table data comes from Flights provided by US Department of Transportation, Bureau of Transportation Statistics. Desaturated from original.

Ignore Parquet statistics

When you read Parquet data, you might receive error messages like the following:

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

To workaround this issue, use the CREATE TABLE or ALTER TABLE SET TBLPROPERTIES statement to set the Parquet SerDe `parquet.ignore.statistics` property to `true`, as in the following examples.

CREATE TABLE example

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES (
'parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE example

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

Regex SerDe

The Regex SerDe uses a regular expression (regex) to deserialize data by extracting regex groups into table columns.

If a row in the data does not match the regex, then all columns in the row are returned as `NULL`. If a row matches the regex but has fewer groups than expected, the missing groups are `NULL`. If a row in the data matches the regex but has more columns than groups in the regex, the additional columns are ignored.

For more information, see [Class RegexSerDe] in the Apache Hive documentation.

Serialization library name

The serialization library name for the Regex SerDe is `org.apache.hadoop.hive.serde2.RegexSerDe`. For source code information, see [Class RegexSerDe] In the Apache documentation.

Example

The following example creates a table from CloudFront logs using the RegExSerDe. Replace *`myregion`* in `s3://athena-examples-`*`myregion`*`/cloudfront/plaintext/` with the region identifier where you run Athena (for example, `s3://athena-examples-us-west-1/cloudfront/plaintext/`).

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
```

```
  BrowserVersion STRING
 )
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
 WITH SERDEPROPERTIES (
 "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\
\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)[\/]
(.*)$"
 )
LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

Run SQL queries in Amazon Athena

You can run SQL queries using Amazon Athena on data sources that are registered with the AWS Glue Data Catalog and data sources such as Hive metastores and Amazon DocumentDB instances that you connect to using the Athena Federated Query feature. For more information about working with data sources, see Connect to data sources. When you run a Data Definition Language (DDL) query that modifies schema, Athena writes the metadata to the metastore associated with the data source. In addition, some queries, such as `CREATE TABLE AS` and `INSERT INTO` can write records to the dataset—for example, adding a CSV record to an Amazon S3 location.

This section provides guidance for running Athena queries on common data sources and data types using a variety of SQL statements. General guidance is provided for working with common structures and operators—for example, working with arrays, concatenating, filtering, flattening, and sorting. Other examples include queries for data in tables with nested structures and maps, tables based on JSON-encoded datasets, and datasets associated with AWS services such as AWS CloudTrail logs and Amazon EMR logs. Comprehensive coverage of standard SQL usage is beyond the scope of this documentation. For more information about SQL, refer to the Trino and Presto language references.

Topics

- Use the cost-based optimizer
- Query S3 Express One Zone data
- Query restored Amazon S3 Glacier objects
- Handle schema updates
- Query arrays
- Query geospatial data
- Query JSON data
- Use Machine Learning (ML) with Amazon Athena
- Query with user defined functions
- Query across regions
- Query the AWS Glue Data Catalog
- Query AWS service logs
- Query web server logs stored in Amazon S3

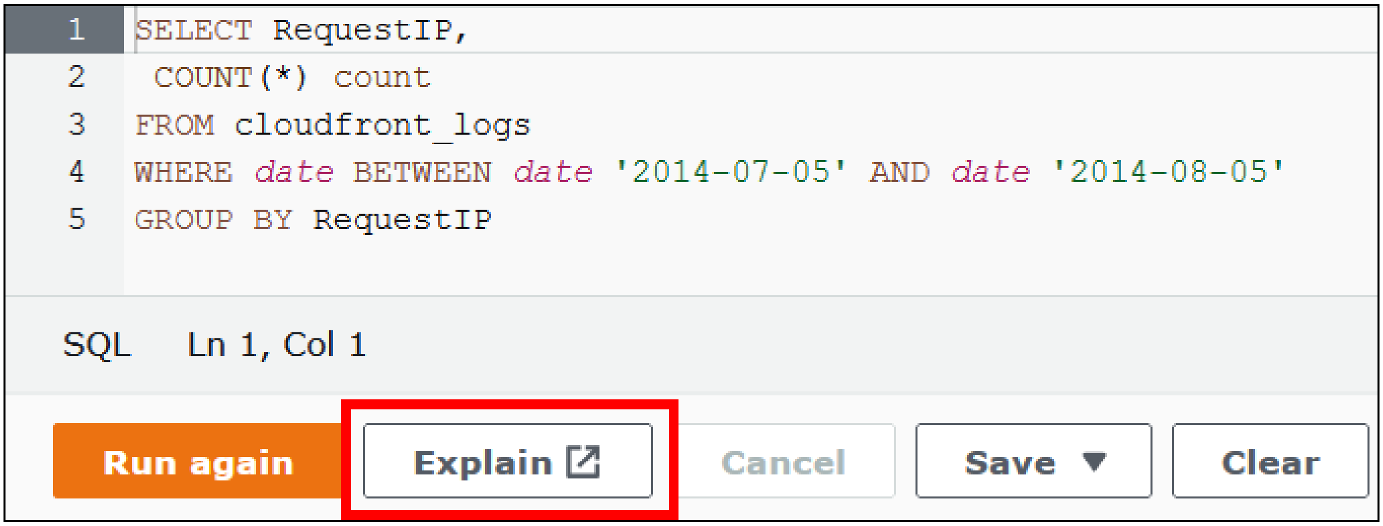
For considerations and limitations, see Considerations and limitations for SQL queries in Amazon Athena.

View execution plans for SQL queries

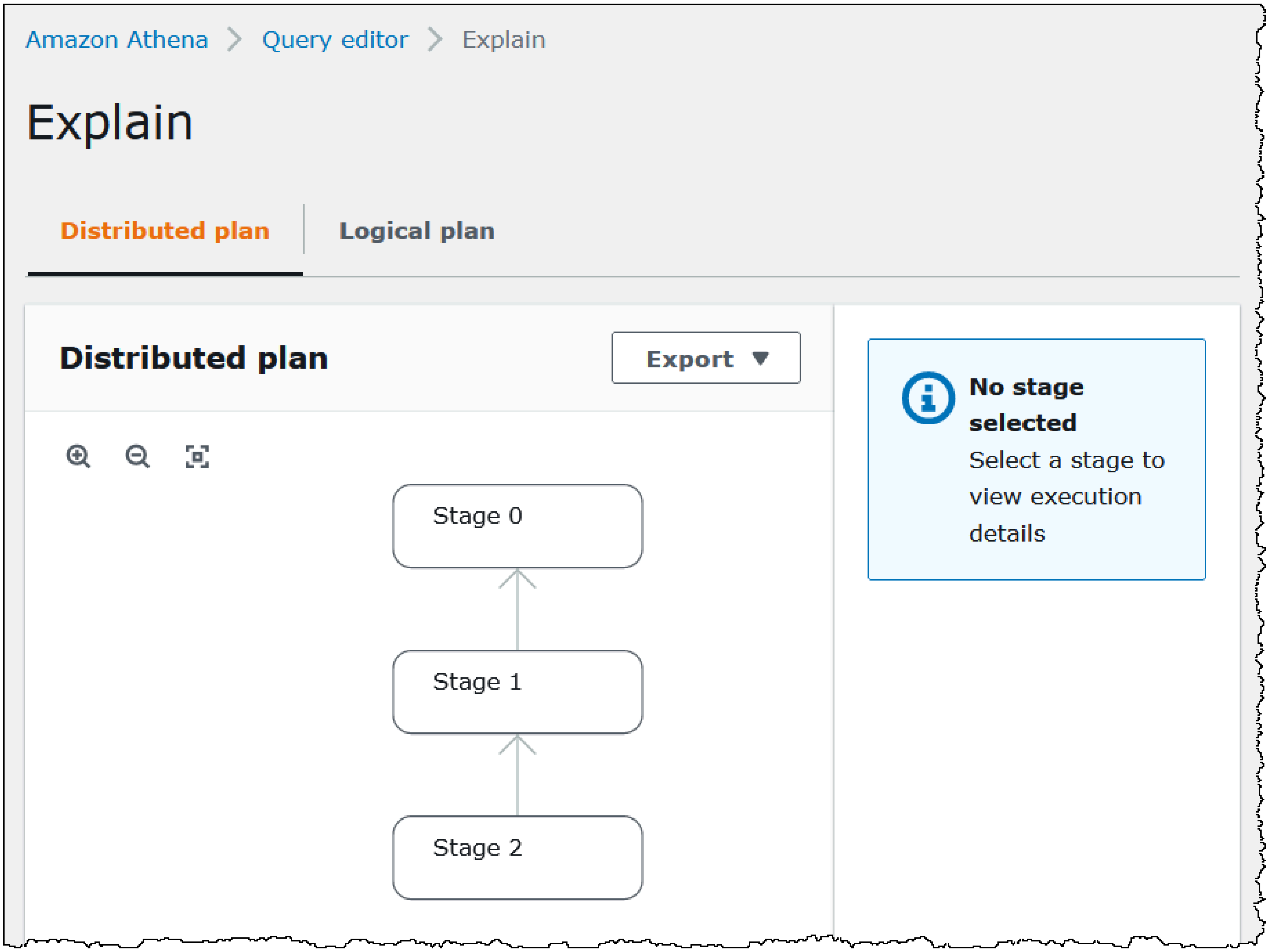
You can use the Athena query editor to see graphical representations of how your query will be run. When you enter a query in the editor and choose the **Explain** option, Athena uses an EXPLAIN SQL statement on your query to create two corresponding graphs: a distributed execution plan and a logical execution plan. You can use these graphs to analyze, troubleshoot, and improve the efficiency of your queries.

To view execution plans for a query

1. Enter your query in the Athena query editor, and then choose **Explain**.



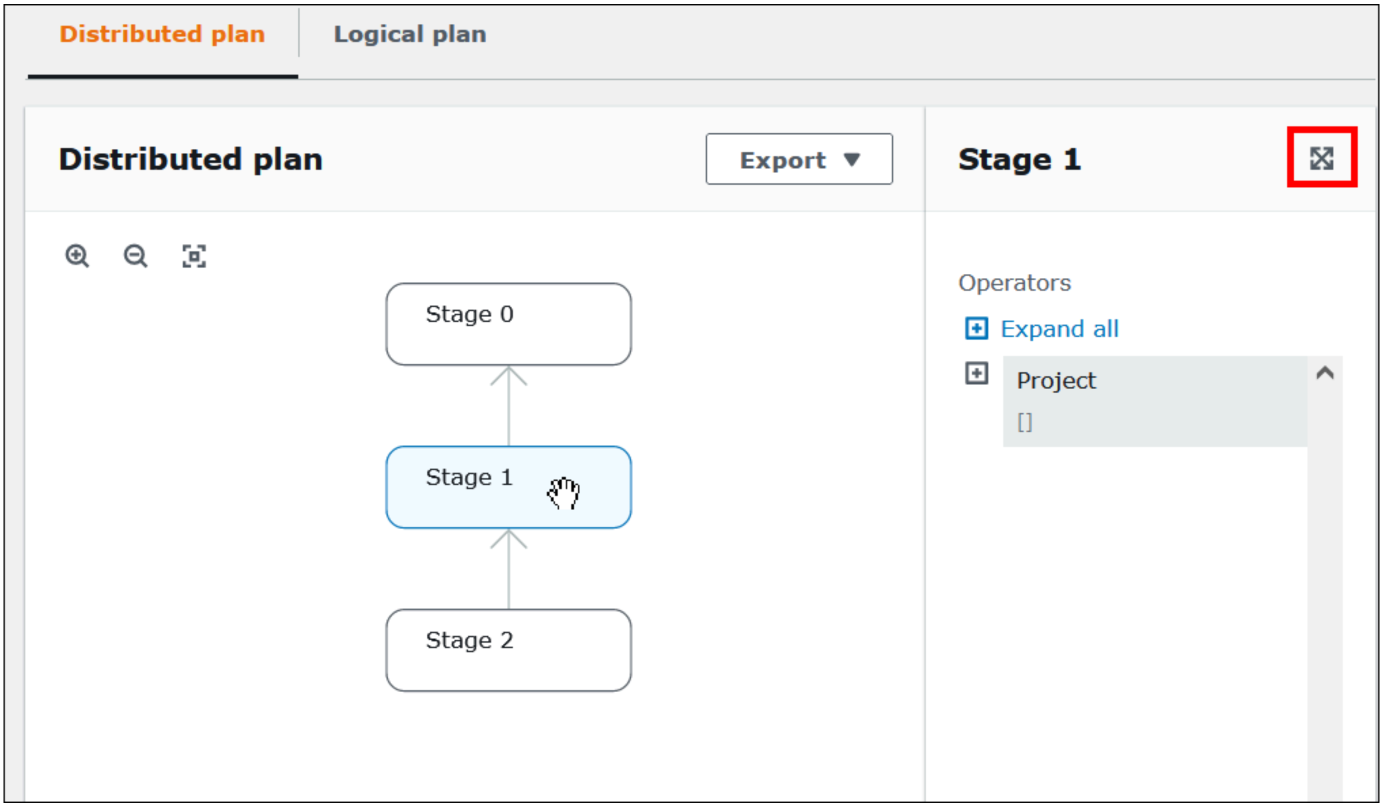
The **Distributed plan** tab shows you the execution plan for your query in a distributed environment. A distributed plan has processing fragments or *stages*. Each stage has a zero-based index number and is processed by one or more nodes. Data can be exchanged between nodes.



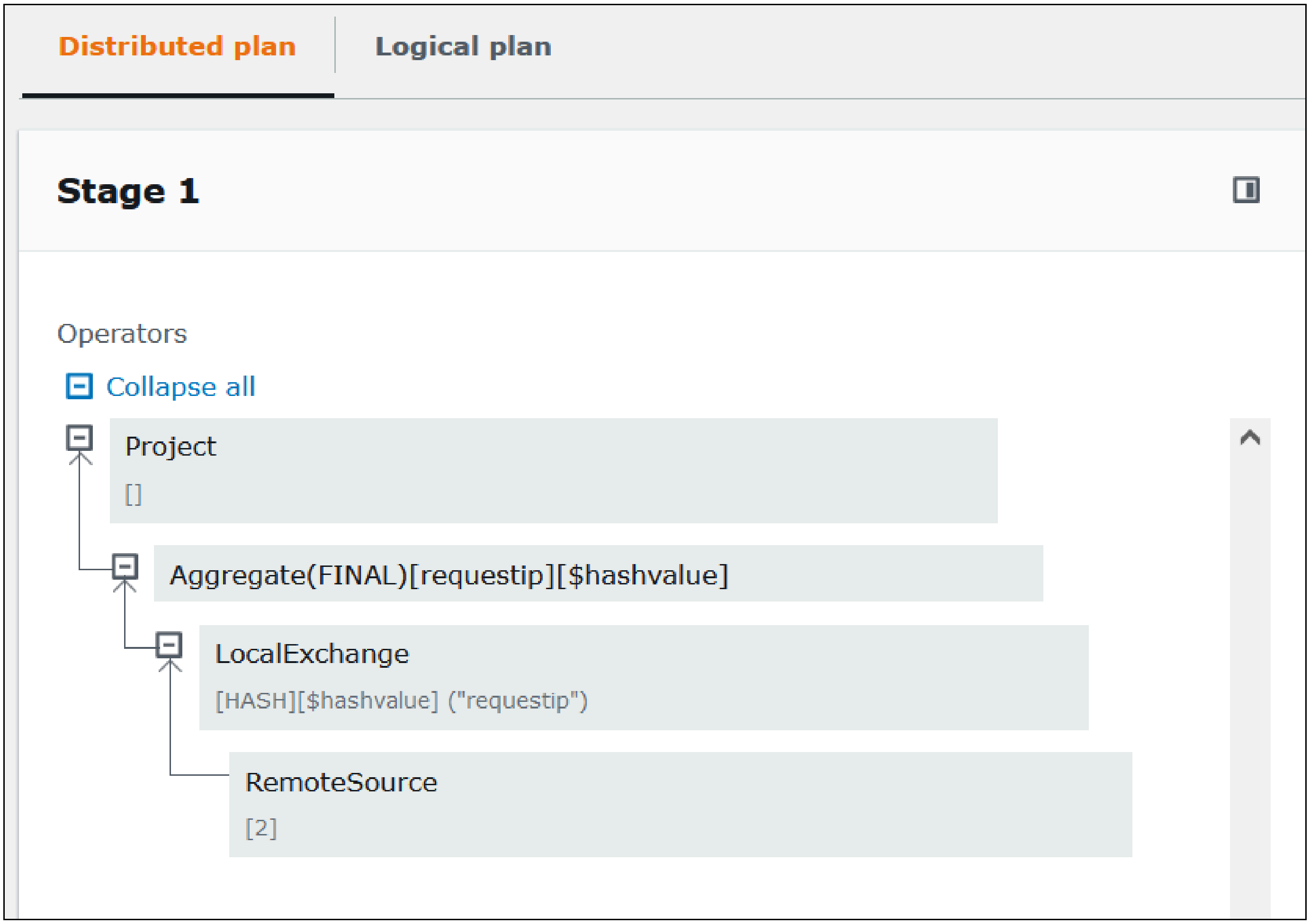
2. To navigate the graph, use the following options:

   - To zoom in or out, scroll the mouse, or use the magnifying icons.
   - To adjust the graph to fit the screen, choose the **Zoom to fit** icon.
   - To move the graph around, drag the mouse pointer.

3. To see details for a stage, choose the stage.



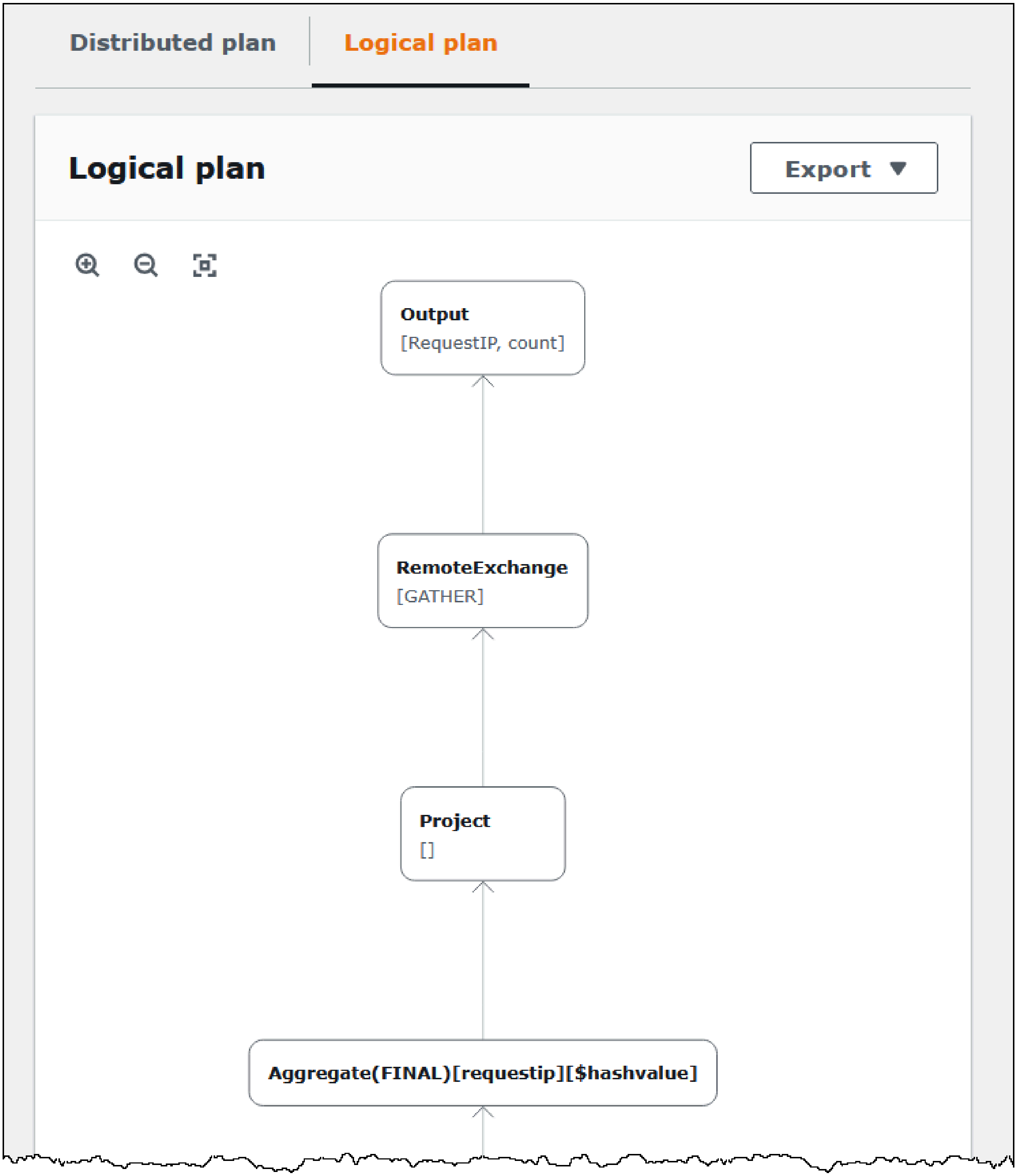
4. To see the stage details full width, choose the expand icon at the top right of the details pane.
5. To see more detail, expand one or more items in the operator tree. For information about distributed plan fragments, see EXPLAIN statement output types.



Important

Currently, some partition filters may not be visible in the nested operator tree graph even though Athena does apply them to your query. To verify the effect of such filters, run EXPLAIN or EXPLAIN ANALYZE on your query and view the results.

6. Choose the **Logical plan** tab. The graph shows the logical plan for running your query. For information about operational terms, see Understand Athena EXPLAIN statement results.



7. To export a plan as an SVG or PNG image, or as JSON text, choose **Export**.

Additional resources

For more information, see the following resources.

Using EXPLAIN and EXPLAIN ANALYZE in Athena

Understand Athena EXPLAIN statement results

View statistics and execution details for completed queries

Work with query results and recent queries

Amazon Athena automatically stores query results and query execution result metadata for each query that runs in a *query result location* that you can specify in Amazon S3. If necessary, you can access the files in this location to work with them. You can also download query result files directly from the Athena console.

Athena now offers you two options for managing query results; you can either use a customer-owned S3 bucket or opt for the managed query results feature. With your own bucket, you maintain complete control over storage, permissions, lifecycle policies, and retention, providing maximum flexibility but require more management. Alternatively, when you choose the managed query results option, the service automatically handles storage and lifecycle management, eliminating your need to configure a separate results bucket and automatically cleaning up results after a predetermined retention period. For more information, see Managed query results.

To set up an Amazon S3 query result location for the first time, see Specify a query result location using the Athena console.

Output files are saved automatically for every query that runs. To access and view query output files using the Athena console, IAM principals (users and roles) need permission to the Amazon S3 GetObject action for the query result location, as well as permission for the Athena GetQueryResults action. The query result location can be encrypted. If the location is encrypted, users must have the appropriate key permissions to encrypt and decrypt the query result location.

> **Important**
>
> IAM principals with permission to the Amazon S3 `GetObject` action for the query result location are able to retrieve query results from Amazon S3 even if permission to the Athena `GetQueryResults` action is denied.

Topics

Managed query results

With managed query results, you can run SQL queries without providing an Amazon S3 bucket for query result storage. This saves you from having to provision, manage, control access to, and clean up your own S3 buckets. To get started, create a new workgroup or edit an existing workgroup. Under **Query result configuration**, select **Athena managed**.

Key features

- Simplifies your workflow by removing the requirement to choose an S3 bucket location before you run queries.
- No additional cost to use managed query results and automatic deletion of query results reduces administration overhead and the need for separate S3 bucket clean up processes.
- Simple to get started: new and pre-existing workgroups can be easily configured to use managed query results. You can have a mix of Athena managed and customer managed query results in your AWS account.
- Streamlined IAM permissions with access to read results through `GetQueryResults` and `GetQueryResultsStream` tied to individual workgroups.
- Query results are automatically encrypted with your choice of AWS owned keys or customer owned keys.

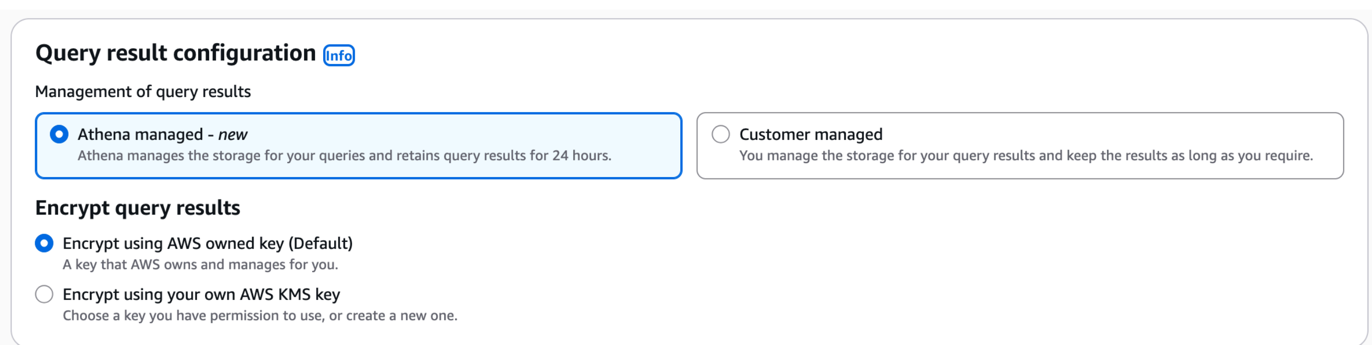
Considerations and limitations

- Access to query results is managed at the workgroup level in Athena. For this, you need explicit permissions to `GetQueryResults` and `GetQueryResultsStream` IAM actions on the specific workgroup. The `GetQueryResults` action determines who can retrieve the results of a completed query in a paginated format, while the `GetQueryResultsStream` action determines who can stream the results of a completed query (commonly used by Athena drivers).
- You cannot download query result files larger than 200 MB from the console. Use the `UNLOAD` statement to write results larger than 200 MB to a location that you can download separately.
- Managed query results feature does not support Query result reuse.
- Query results are available for 24 hours. Query results are stored at no cost to you during this period. After this period, query results are automatically deleted.

Create or edit a workgroup with managed query results

To create a workgroup or update an existing workgroup with managed query results from the console:

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. From the left navigation, choose **Workgroups**.
3. Choose **Create Workgroup** to create a new workgroup or edit an existing workgroup from the list.
4. Under **Query result configuration**, choose **Athena managed**.



5. For **Encrypt query results**, choose the encryption option that you want. For more information, see Choose query result encryption.
6. Fill in all the other required details and choose **Save changes**.

Choose query result encryption

There are two options for encryption configuration:

- **Encrypt using an AWS owned key** – This is the default option when you use managed query results. Choose this option if you want query results encrypted by an AWS owned key.
- **Encrypt using customer managed key** – Choose this option if you want to encrypt and decrypt query results with a customer managed key. To use a customer managed key, add the Athena service in the Principal element of the key policy section. For more information, see Set up an AWS KMS key policy for managed query results. To run queries successfully, the user running queries needs permission to access the AWS KMS key.

Set up an AWS KMS key policy for managed query results

The `Principal` section on the key policy specifies who can use this key. The managed query results feature introduces the principal `encryption.athena.amazonaws.com` that you must specify in the `Principal` section. This service principal is specifically for accessing keys not owned by Athena. You must also add the `kms:Decrypt`, `kms:GenerateDataKey`, and `kms:DescribeKey` actions to the key policy that you use to access managed results. These three actions are the minimum allowed actions.

Managed query results uses your workgroup ARN for encryption context. Because the `Principal` section is an AWS service, you also need to add `aws:sourceArn` and `aws:sourceAccount` to the key policy conditions. The following example shows an AWS KMS key policy that has minimum permissions on a single workgroup.

```
{
   "Sid": "Allow athena service principal to use the key",
   "Effect": "Allow",
   "Principal": {
       "Service": "encryption.athena.amazonaws.com"
   },
   "Action": [
       "kms:Decrypt",
       "kms:GenerateDataKey",
       "kms:DescribeKey"
     ],
   "Resource": "arn:aws:kms:us-east-1:{account-id}:key/{key-id}",
   "Condition": {
   "ArnLike": {
```

```
        "kms:EncryptionContext:aws:athena:arn": "arn:aws:athena:us-east-1:{account-
id}:workgroup/{workgroup-name}",
        "aws:SourceArn": "arn:aws:athena:us-east-1:{account-id}:workgroup/{workgroup-
name}"
    },
    "StringEquals": {
        "aws:SourceAccount": "{account-id}"
    }
}
```

The following example AWS KMS key policy allows all workgroups within the same account *account-id* to use the same AWS KMS key.

```
{
    "Sid": "Allow athena service principal to use the key",
    "Effect": "Allow",
    "Principal": {
        "Service": "encryption.athena.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:DescribeKey"
    ],
    "Resource": "arn:aws:kms:us-east-1:account-id:key/{key-id}",
    "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:athena:arn": "arn:aws:athena:us-east-1:account-
id:workgroup/*",
          "aws:SourceArn": "arn:aws:athena:us-east-1:account-id:workgroup/*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
    }
}
```

In addition to the Athena and Amazon S3 permissions, you must also get permissions to perform `kms:GenerateDataKey` and `kms:Decrypt` actions. For more information, see the section called "Permissions to encrypted data in Amazon S3".

For more information on managed query results encryption, see the section called "Encrypt managed query results".

Encrypt managed query results

Athena offers the following options for encrypting the section called "Managed query results".

Encrypt using an AWS owned key

This is the default option when you use managed query results. This option indicates that you want to encrypt query results using an AWS owned key. AWS owned keys are not stored in your AWS account and are part of a collection of KMS keys that AWS owns. You are not charged a fee when you use AWS owned keys, and they do not count against AWS KMS quotas for your account.

Encrypt using AWS KMS customer managed key

Customer managed keys are the KMS keys in your AWS account that you create, own, and manage. You have full control over these KMS keys, which includes establishing and maintaining their key policies, IAM policies and grants, enabling and disabling them, rotating their cryptographic material, adding tags, creating aliases that refer to them, and scheduling them for deletion. For more information, see Customer managed keys.

How Athena uses customer managed key to encrypt results

When you specify a customer managed key, Athena uses it to encrypt the query results when stored in managed query results. The same key is used to decrypt the results when you call `GetQueryResults`. When you set the state of the customer managed key to disabled or schedule it for deletion, it prevents Athena and all users from encrypting or decrypting results with that key.

Athena uses envelope encryption and key hierarchy to encrypt data. Your AWS KMS encryption key is used to generate and decrypt the root key of this key hierarchy.

Each result is encrypted using the customer managed key configured in the workgroup at the time of encryption. Switching the key to a different customer managed key or to an AWS owned key does not re-encrypt existing results with the new key. Deleting and disabling a particular customer managed key only affects decryption of the results that the key encrypted.

Athena needs access to your encryption key to perform `kms:Decrypt`, `kms:GenerateDataKey`, and `kms:DescribeKey` operations for encrypting and decrypting results. For more information, see the section called "Permissions to encrypted data in Amazon S3".

The principal that submits the query using the `StartQueryExecution` API and reads results using `GetQueryResults` must also have permission to the customer managed key for `kms:Decrypt`, `kms:GenerateDataKey`, and `kms:DescribeKey` operations in addition to Athena and Amazon S3 permissions. For more information, see Key policies in AWS KMS.

Specify a query result location

The query result location that Athena uses is determined by a combination of workgroup settings and *client-side settings*. Client-side settings are based on how you run the query.

- If you run the query using the Athena console, the **Query result location** entered under **Settings** in the navigation bar determines the client-side setting.
- If you run the query using the Athena API, the `OutputLocation` parameter of the StartQueryExecution action determines the client-side setting.
- If you use the ODBC or JDBC drivers to run queries, the `S3OutputLocation` property specified in the connection URL determines the client-side setting.

> **Important**
>
> When you run a query using the API or using the ODBC or JDBC driver, the console setting does not apply.

Each workgroup configuration has an Override client-side settings option that can be enabled. When this option is enabled, the workgroup settings take precedence over the applicable client-side settings when an IAM principal associated with that workgroup runs the query.

About previously created default locations

Previously in Athena, if you ran a query without specifying a value for **Query result location**, and the query result location setting was not overridden by a workgroup, Athena created a default location for you. The default location was `aws-athena-query-results-MyAcctID-MyRegion`, where *`MyAcctID`* was the Amazon Web Services account ID of the IAM principal that ran the query, and *`MyRegion`* was the region where the query ran (for example, `us-west-1`.)

Now, before you can run an Athena query in a region in which your account hasn't used Athena previously, you must specify a query result location, or use a workgroup that overrides the query result location setting. While Athena no longer creates a default query results location for you, previously created default `aws-athena-query-results-MyAcctID-MyRegion` locations remain valid and you can continue to use them.

Topics

- Specify a query result location using the Athena console
- Specify a query result location using a workgroup

Specify a query result location using the Athena console

Before you can run a query, a query result bucket location in Amazon S3 must be specified, or you must use a workgroup that has specified a bucket and whose configuration overrides client settings.

To specify a client-side setting query result location using the Athena console

1. Switch to the workgroup for which you want to specify a query results location. The name of the default workgroup is **primary**.
2. From the navigation bar, choose **Settings**.
3. From the navigation bar, choose **Manage**.
4. For **Manage settings**, do one of the following:
   - In the **Location of query result** box, enter the path to the bucket that you created in Amazon S3 for your query results. Prefix the path with `s3://`.
   - Choose **Browse S3**, choose the Amazon S3 bucket that you created for your current Region, and then choose **Choose**.

> **Note**
>
> If you are using a workgroup that specifies a query result location for all users of the workgroup, the option to change the query result location is unavailable.

5. (Optional) Choose **View lifecycle configuration** to view and configure the Amazon S3 lifecycle rules on your query results bucket. The Amazon S3 lifecycle rules that you create can be either expiration rules or transition rules. Expiration rules automatically delete query results after a certain amount of time. Transition rules move them to another Amazon S3 storage tier. For more information, see Setting lifecycle configuration on a bucket in the Amazon Simple Storage Service User Guide.

6. (Optional) For **Expected bucket owner**, enter the ID of the AWS account that you expect to be the owner of the output location bucket. This is an added security measure. If the account ID of the bucket owner does not match the ID that you specify here, attempts to output to the bucket will fail. For in-depth information, see Verifying bucket ownership with bucket owner condition in the *Amazon S3 User Guide*.

> **Note**
>
> The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. It does not apply to other Amazon S3 locations like data source locations in external Amazon S3 buckets, `CTAS` and `INSERT INTO` destination table locations, `UNLOAD` statement output locations, operations to spill buckets for federated queries, or `SELECT` queries run against a table in another account.

7. (Optional) Choose **Encrypt query results** if you want to encrypt the query results stored in Amazon S3. For more information about encryption in Athena, see Encryption at rest.
8. (Optional) Choose **Assign bucket owner full control over query results** to grant full control access over query results to the bucket owner when ACLs are enabled for the query result bucket. For example, if your query result location is owned by another account, you can grant ownership and full control over your query results to the other account. For more information, see Controlling ownership of objects and disabling ACLs for your bucket in the *Amazon S3 User Guide*.
9. Choose **Save**.

Specify a query result location using a workgroup

You specify the query result location in a workgroup configuration using the AWS Management Console, the AWS CLI, or the Athena API.

When using the AWS CLI, specify the query result location using the `OutputLocation` parameter of the `--configuration` option when you run the `aws athena create-work-group` or `aws athena update-work-group` command.

To specify the query result location for a workgroup using the Athena console

1. If the console navigation pane is not visible, choose the expansion menu on the left.

2. In the navigation pane, choose **Workgroups**.
3. In the list of workgroups, choose the link of the workgroup that you want to edit.
4. Choose **Edit**.
5. For **Query result location and encryption**, do one of the following:
   - In the **Location of query result** box, enter the path to a bucket in Amazon S3 for your query results. Prefix the path with `s3://`.
   - Choose **Browse S3**, choose the Amazon S3 bucket for your current Region that you want to use, and then choose **Choose**.
6. (Optional) For **Expected bucket owner**, enter the ID of the AWS account that you expect to be the owner of the output location bucket. This is an added security measure. If the account ID of the bucket owner does not match the ID that you specify here, attempts to output to the bucket will fail. For in-depth information, see [Verifying bucket ownership with bucket owner condition](#) in the *Amazon S3 User Guide*.

> **Note**
>
> The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. It does not apply to other Amazon S3 locations like data source locations in external Amazon S3 buckets, `CTAS` and `INSERT INTO` destination table locations, `UNLOAD` statement output locations, operations to spill buckets for federated queries, or `SELECT` queries run against a table in another account.

7. (Optional) Choose **Encrypt query results** if you want to encrypt the query results stored in Amazon S3. For more information about encryption in Athena, see Encryption at rest.
8. (Optional) Choose **Assign bucket owner full control over query results** to grant full control access over query results to the bucket owner when ACLs are enabled for the query result bucket. For example, if your query result location is owned by another account, you can grant ownership and full control over your query results to the other account.

   If the bucket's S3 Object Ownership setting is **Bucket owner preferred**, the bucket owner also owns all query result objects written from this workgroup. For example, if an external account's workgroup enables this option and sets its query result location to your account's Amazon S3 bucket which has an S3 Object Ownership setting of **Bucket owner preferred**, you own and have full control access over the external workgroup's query results.

   Selecting this option when the query result bucket's S3 Object Ownership setting is **Bucket owner enforced** has no effect. For more information, see Controlling ownership of objects and disabling ACLs for your bucket in the *Amazon S3 User Guide*.
9. If you want to require all users of the workgroup to use the query results location that you specified, scroll down to the **Settings** section and select **Override client-side settings**.
10. Choose **Save changes**.

Download query results files using the Athena console

You can download the query results CSV file from the query pane immediately after you run a query. You can also download query results from recent queries from the **Recent queries** tab.

> **Note**
>
> Athena query result files are data files that contain information that can be configured by individual users. Some programs that read and analyze this data can potentially interpret some of the data as commands (CSV injection). For this reason, when you import query results CSV data to a spreadsheet program, that program might warn you about security concerns. To keep your system secure, you should always choose to disable links or macros from downloaded query results.

To run a query and download the query results

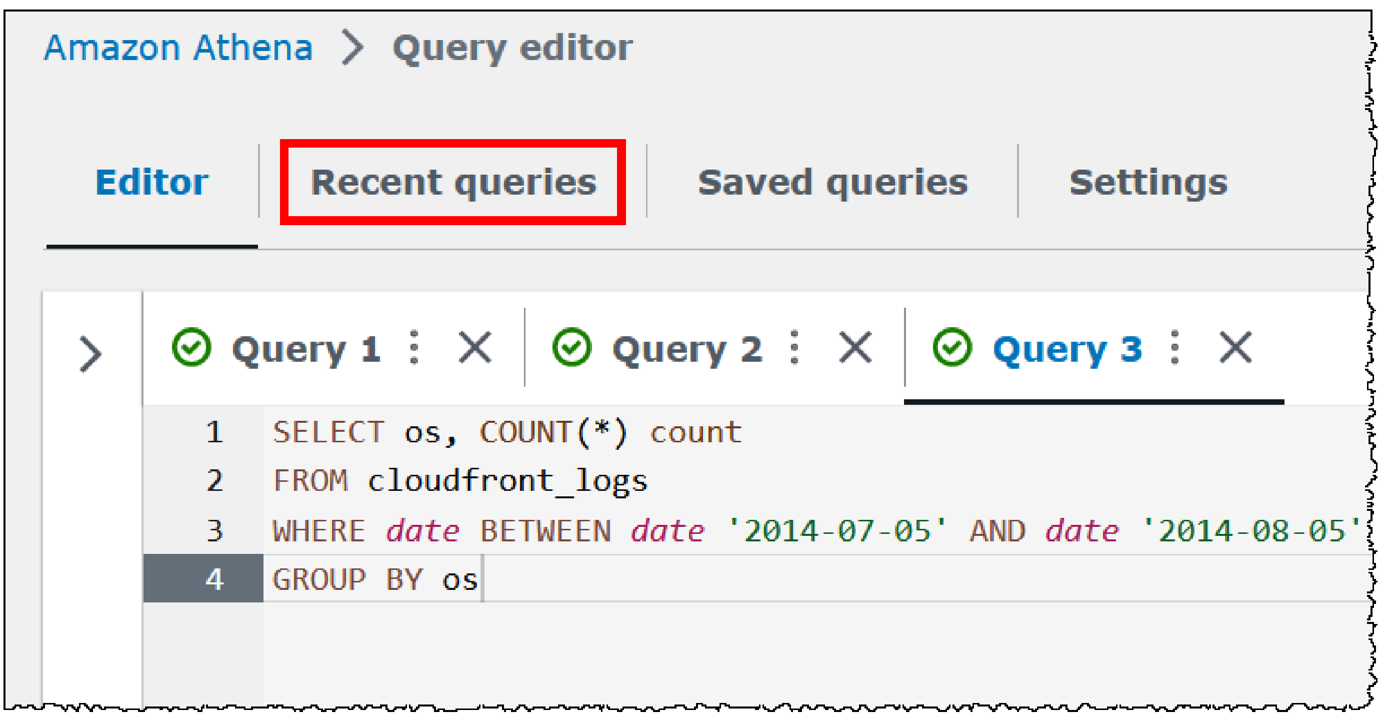
1. Enter your query in the query editor and then choose **Run**.

   When the query finishes running, the **Results** pane shows the query results.

2. To download a CSV file of the query results, choose **Download results** above the query results pane. Depending on your browser and browser configuration, you may need to confirm the download.

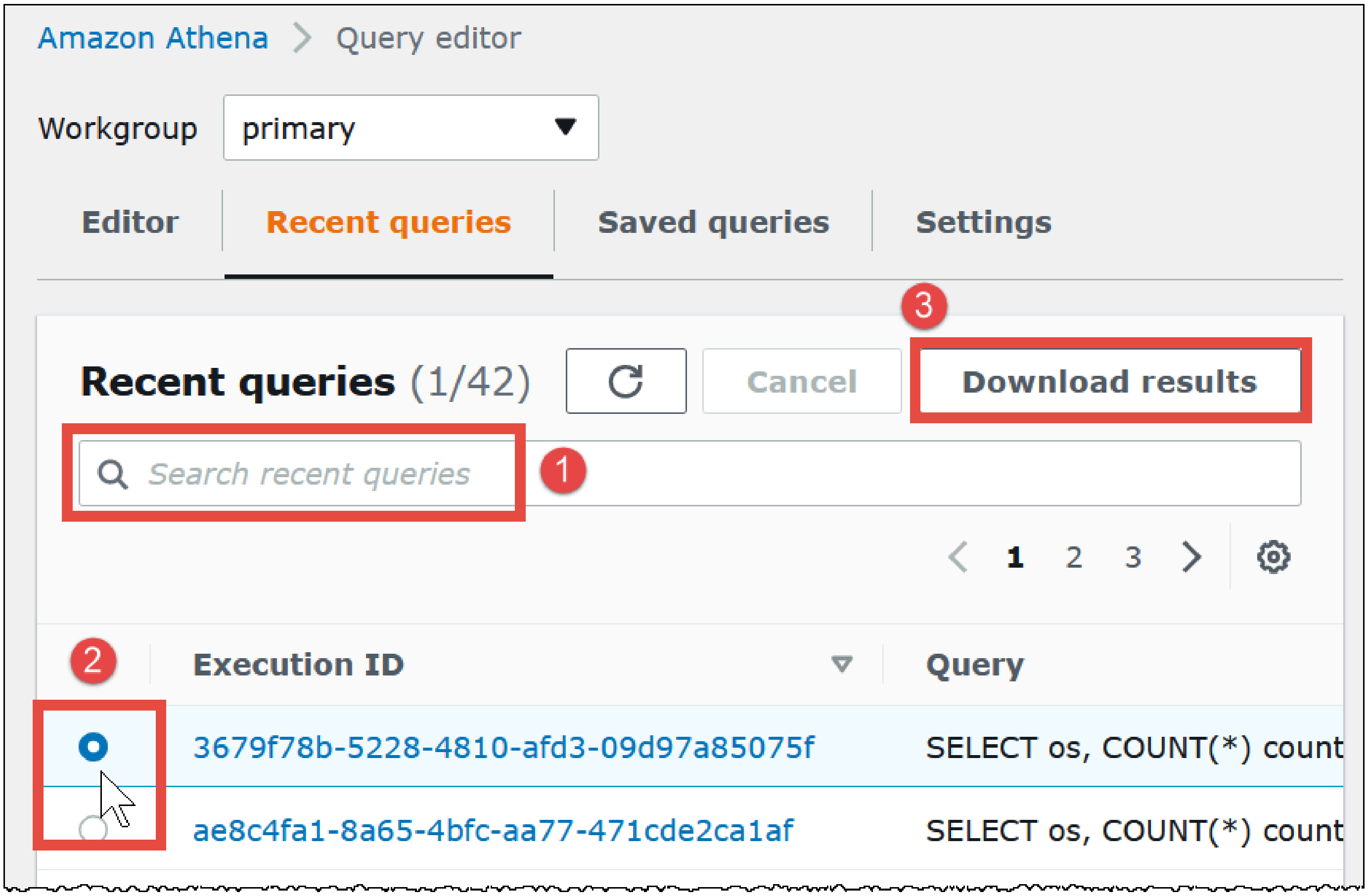
To download a query results file for an earlier query

1. Choose **Recent queries**.



2. Use the search box to find the query, select the query, and then choose **Download results**.

> **Note**
>
> You cannot use the **Download results** option to retrieve query results that have been deleted manually, or retrieve query results that have been deleted or moved to another location by Amazon S3 lifecycle rules.



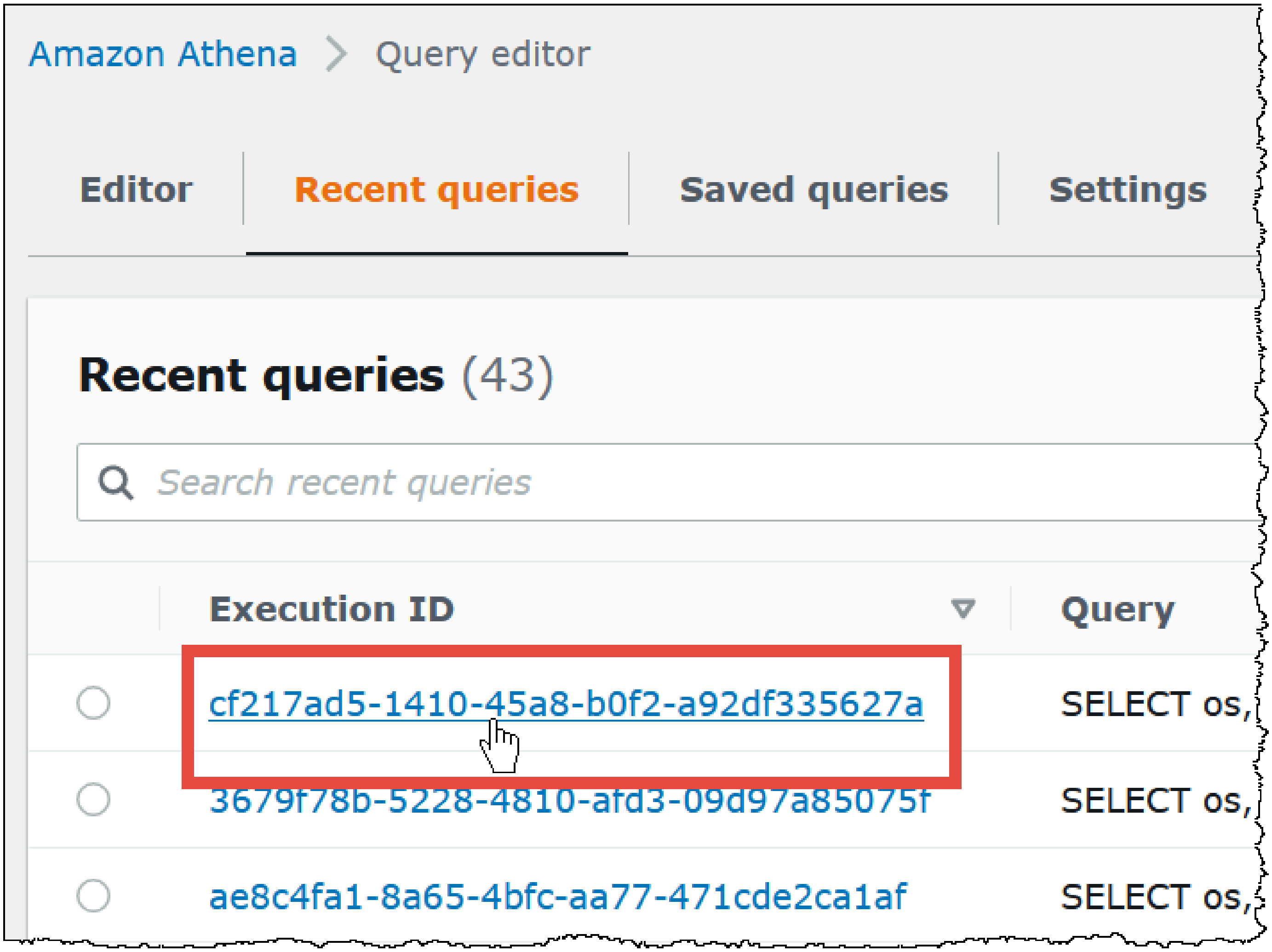
View recent queries in the Athena console

You can use the Athena console to see which queries succeeded or failed, and view error details for the queries that failed. Athena keeps a query history for 45 days.

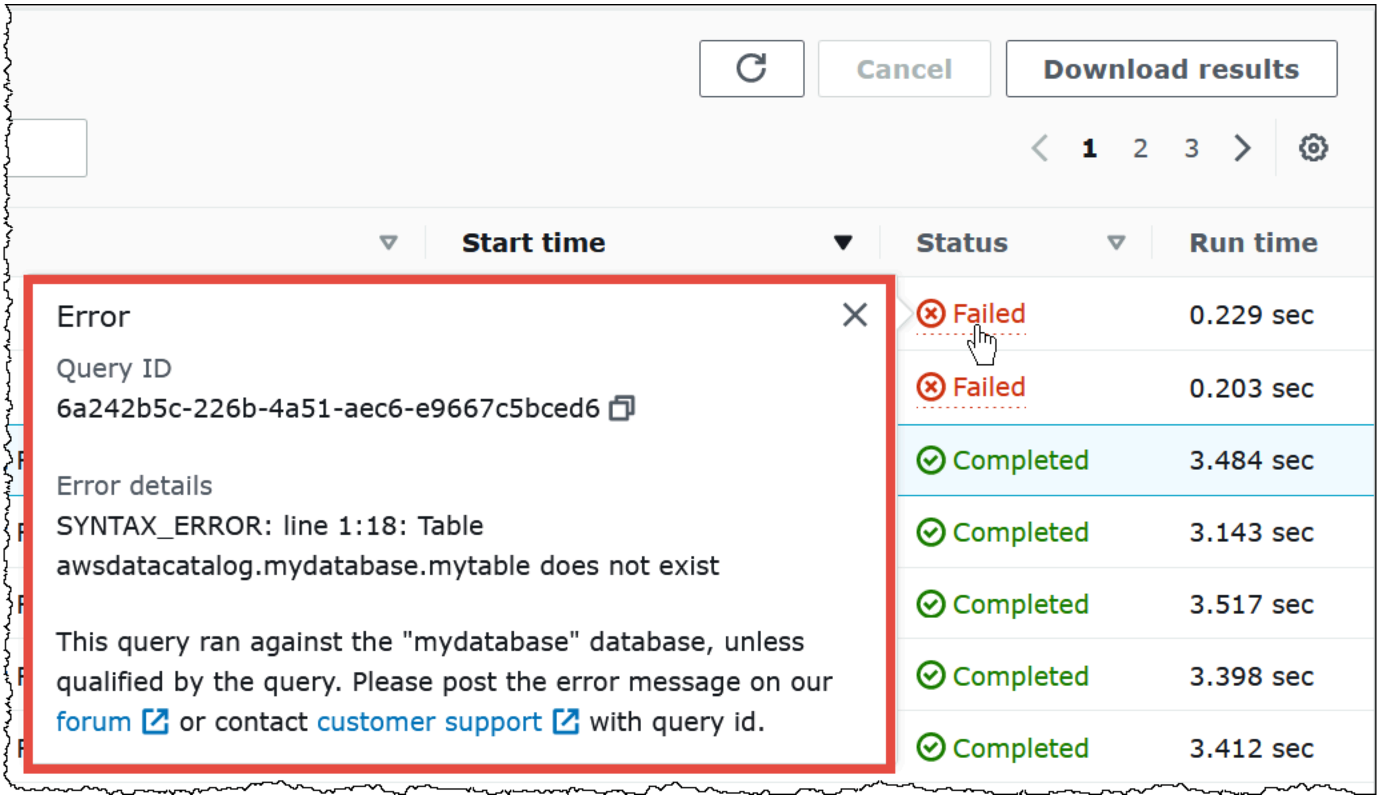
To view recent queries in the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Choose **Recent queries**. The **Recent queries** tab shows information about each query that ran.

3. To open a query statement in the query editor, choose the query's execution ID.



4. To see the details for a query that failed, choose the **Failed** link for the query.

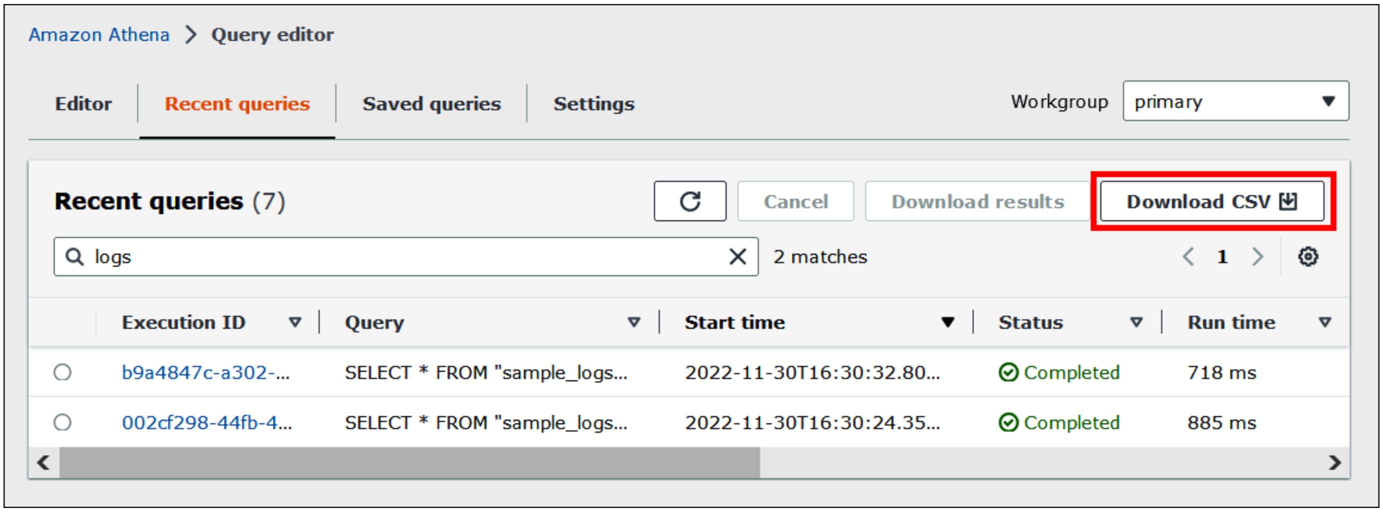


Download multiple recent queries to a CSV file

You can use the **Recent queries** tab of the Athena console to export one or more recent queries to a CSV file in order to view them in tabular format. The downloaded file contains not the query results, but the SQL query string itself and other information about the query. Exported fields include the execution ID, query string contents, query start time, status, run time, amount of data scanned, query engine version used, and encryption method. You can export a maximum of 500 recent queries, or a filtered maximum of 500 queries using criteria that you enter in the search box.

To export one or more recent queries to a CSV file

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Choose **Recent queries**.
3. (Optional) Use the search box to filter for the recent queries that you want to download.
4. Choose **Download CSV**.



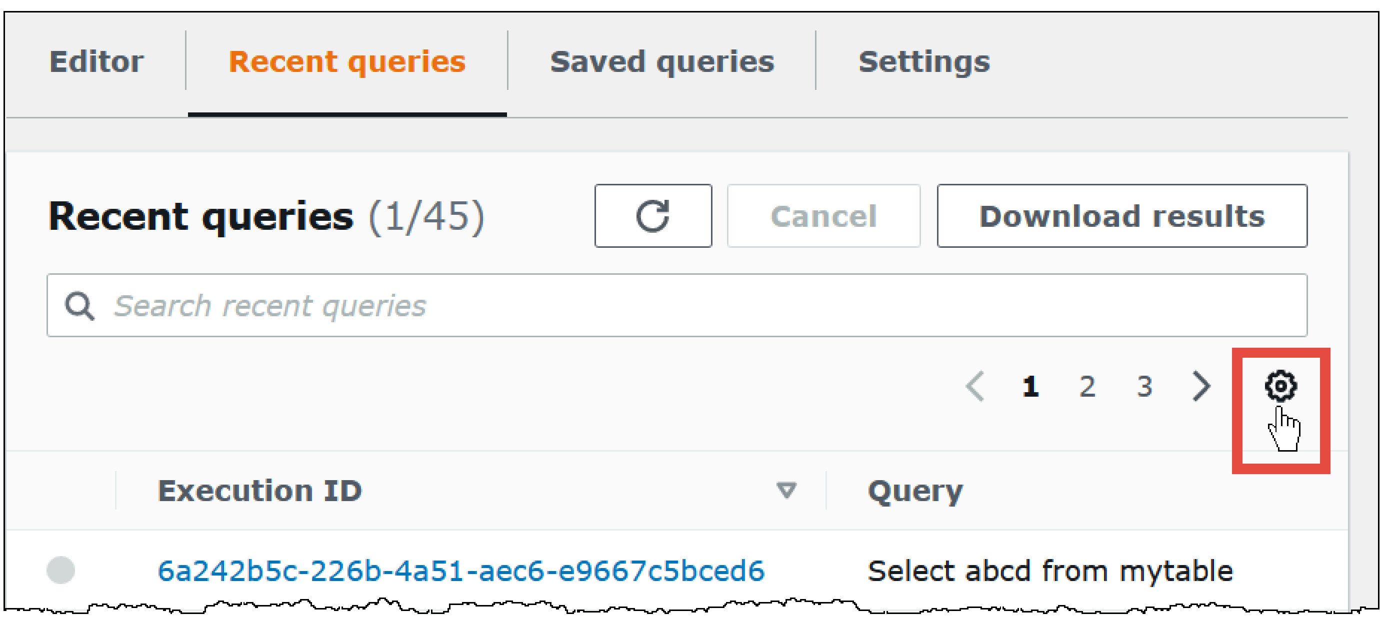
5. At the file save prompt, choose **Save**. The default file name is `Recent Queries` followed by a timestamp (for example, `Recent Queries 2022-12-05T16 04 27.352-08 00.csv`)

Configure recent query display options

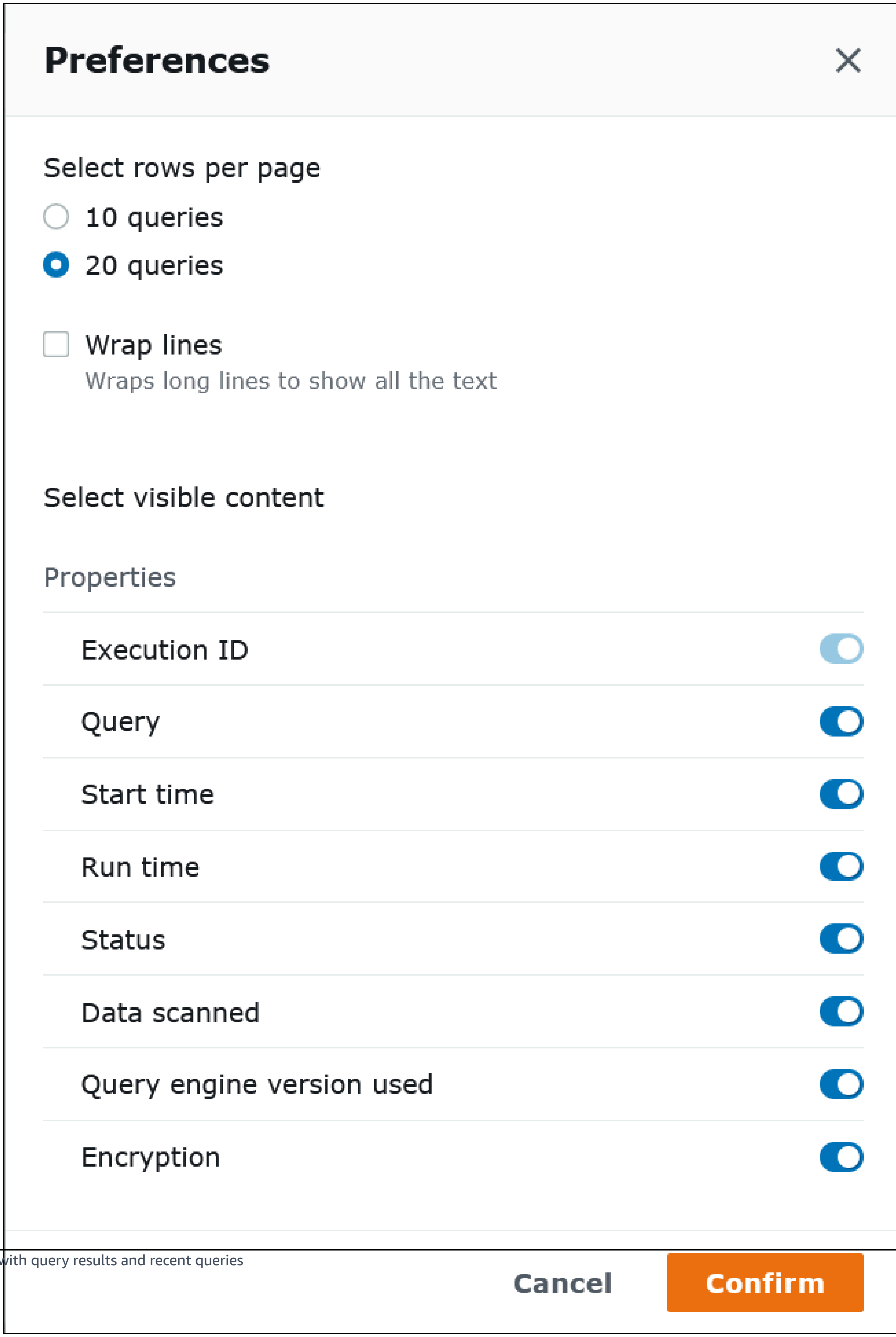
You can configure options for the **Recent queries** tab like columns to display and text wrapping.

To configure options for the Recent queries tab

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Choose **Recent queries**.
3. Choose the options button (gear icon).



4. In the **Preferences** dialog box, choose the number of rows per page, line wrapping behavior, and columns to display.


Preferences
Select rows per page
10 queries
20 queries
Wrap lines
Wraps long lines to show all the text
Select visible content
Properties
Execution ID
Query
Start time
Run time
Status
Data scanned
Query engine version used
Encryption
Cancel
Confirm

5. Choose **Confirm**.

Keep your query history longer than 45 days

If you want to keep the query history longer than 45 days, you can retrieve the query history and save it to a data store such as Amazon S3. To automate this process, you can use Athena and Amazon S3 API actions and CLI commands. The following procedure summarizes these steps.

To retrieve and save query history programmatically

1. Use Athena ListQueryExecutions API action or the list-query-executions CLI command to retrieve the query IDs.
2. Use the Athena GetQueryExecution API action or the get-query-execution CLI command to retrieve information about each query based on its ID.
3. Use the Amazon S3 PutObject API action or the put-object CLI command to save the information in Amazon S3.

Find query output files in Amazon S3

Query output files are stored in sub-folders on Amazon S3 in the following path pattern unless the query occurs in a workgroup whose configuration overrides client-side settings. When workgroup configuration overrides client-side settings, the query uses the results path specified by the workgroup.

```
QueryResultsLocationInS3/[QueryName|Unsaved/yyyy/mm/dd/]
```

- *QueryResultsLocationInS3* is the query result location specified either by workgroup settings or client-side settings. For more information, see the section called "Specify a query result location" later in this document.
- The following sub-folders are created only for queries run from the console whose results path has not been overriden by workgroup configuration. Queries that run from the AWS CLI or using the Athena API are saved directly to the *QueryResultsLocationInS3*.
  - *QueryName* is the name of the query for which the results are saved. If the query ran but wasn't saved, `Unsaved` is used.
  - *yyyy/mm/dd* is the date that the query ran.

Files associated with a `CREATE TABLE AS SELECT` query are stored in a `tables` sub-folder of the above pattern.

Identify query output files

Files are saved to the query result location in Amazon S3 based on the name of the query, the query ID, and the date that the query ran. Files for each query are named using the *QueryID*, which is a unique identifier that Athena assigns to each query when it runs.

The following file types are saved:

File type	File naming patterns	Description
Query results files	*QueryID*.csv *QueryID*.txt	DML query results files are saved in comma-separated values (CSV) format. DDL query results are saved as plain text files. You can download results files from the console from the **Results** pane when using the console or from the query **History**. For more informati on, see Download query results files using the Athena console.
Query metadata files	*QueryID*.csv.metadata *QueryID*.txt.metadata	DML and DDL query metadata files are saved in binary format and are not human readable. The file extension corresponds to the related query results file. Athena uses the metadata when reading query results using the `GetQueryResults` action. Although these

File type	File naming patterns	Description
		files can be deleted, we do not recommend it because important information about the query is lost.
Data manifest files	*QueryID*-manifest.csv	Data manifest files are generated to track files that Athena creates in Amazon S3 data source locations when an INSERT INTO query runs. If a query fails, the manifest also tracks files that the query intended to write. The manifest is useful for identifyi ng orphaned files resulting from a failed query.

Use the AWS CLI to identify query output location and files

To use the AWS CLI to identify the query output location and result files, run the `aws athena get-query-execution` command, as in the following example. Replace *abc1234d-5efg-67hi-jklm-89n0op12qr34* with the query ID.

```
aws athena get-query-execution --query-execution-id abc1234d-5efg-67hi-
jklm-89n0op12qr34
```

The command returns output similar to the following. For descriptions of each output parameter, see get-query-execution in the *AWS CLI Command Reference*.

```
{
    "QueryExecution": {
        "Status": {
            "SubmissionDateTime": 1565649050.175,
            "State": "SUCCEEDED",
            "CompletionDateTime": 1565649056.6229999
        },
        "Statistics": {
```

```
            "DataScannedInBytes": 5944497,
            "DataManifestLocation": "s3://amzn-s3-demo-bucket/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34-manifest.csv",
            "EngineExecutionTimeInMillis": 5209
        },
        "ResultConfiguration": {
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            },
            "OutputLocation": "s3://amzn-s3-demo-bucket/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34"
        },
        "QueryExecutionId": "abc1234d-5efg-67hi-jklm-89n0op12qr34",
        "QueryExecutionContext": {},
        "Query": "INSERT INTO mydb.elb_log_backup SELECT * FROM mydb.elb_logs LIMIT
 100",
        "StatementType": "DML",
        "WorkGroup": "primary"
    }
}
```

Reuse query results in Athena

When you re-run a query in Athena, you can optionally choose to reuse the last stored query result. This option can increase performance and reduce costs in terms of the number of bytes scanned. Reusing query results is useful if, for example, you know that the results will not change within a given time frame. You can specify a maximum age for reusing query results. Athena uses the stored result as long as it is not older than the age that you specify. For more information, see [Reduce cost and improve query performance with Amazon Athena](#) in the *AWS Big Data Blog*.

> **Note**
>
> The query result reuse feature requires Athena engine version 3. For information about changing engine versions, see [Change Athena engine versions](#).

Key features

- Reusing query results is a per-query, opt-in feature. You can enable query result reuse on a per query basis.
- The maximum age for reusing query results can be specified in minutes, hours, or days. The maximum age specifiable is the equivalent of 7 days regardless of the time unit used. The default is 60 minutes.
- When you enable result reuse for a query, Athena looks for a previous execution of the query within the same workgroup. If Athena finds corresponding stored query results, it does not rerun the query, but points to the previous result location or fetches data from it.

- For any query that enables the results reuse option, Athena reuses the last query result saved to the workgroup folder only when all the following conditions are true:
  - The query string is an exact match.
  - The database and the catalog name match.
  - The previous result is not older than the maximum age specified, or not older than 60 minutes if a maximum age has not been specified.
  - Athena only reuses an execution that has the exact same result configuration as the current execution.
  - You have access to all the tables referenced in the query.
  - You have access to the S3 file location where the previous result is stored.

If any of these conditions are not met, Athena runs the query without using the cached results.

Considerations and limitations

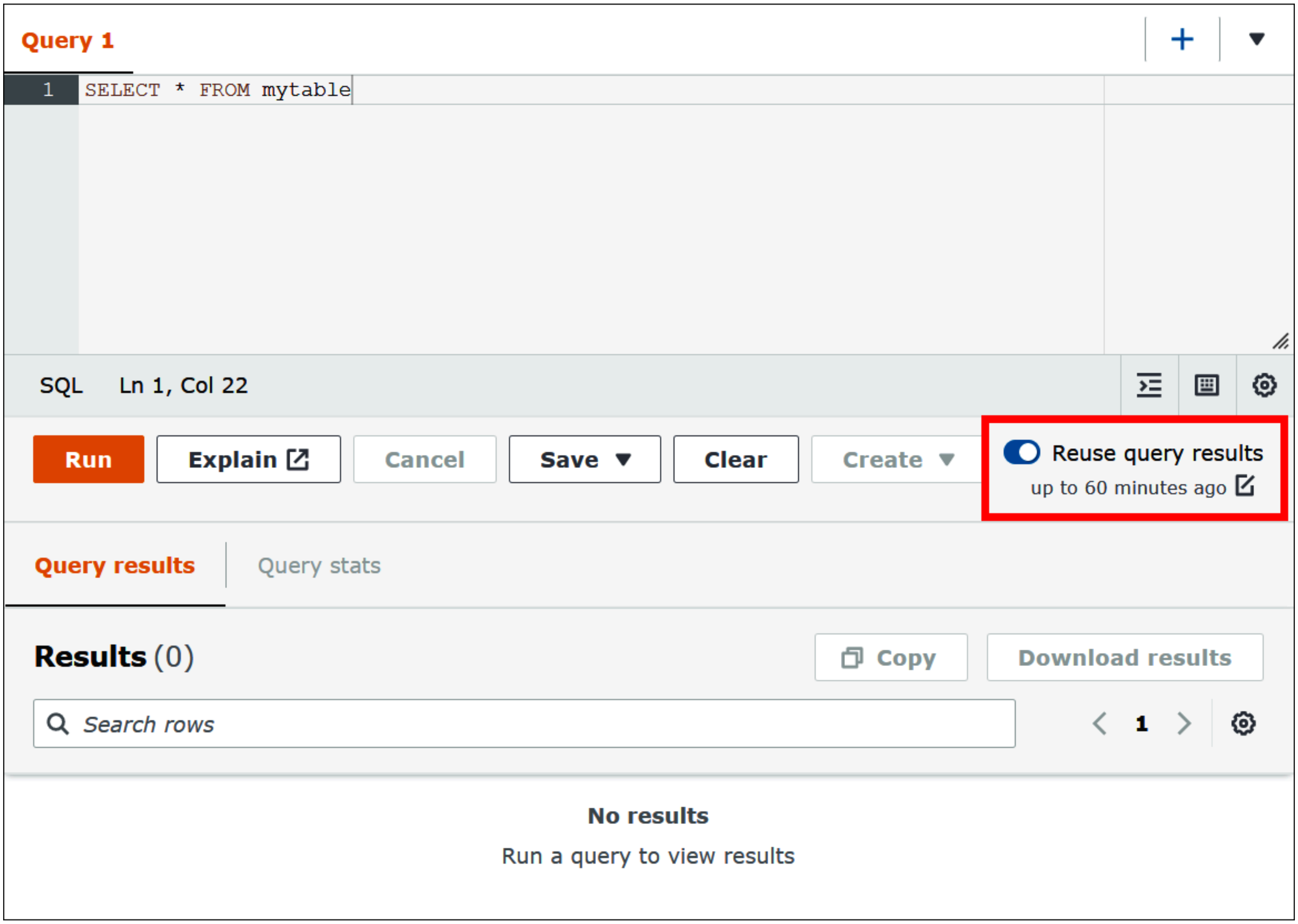
When using the query result reuse feature, keep in mind the following points:

- Athena reuses query results only within the same workgroup.
- The reuse query results feature respects workgroup configurations. If you override the result configuration for a query, the feature is disabled.
- Only queries that produce query result sets on Amazon S3 can reuse query results. This means that, for example, CTAS, `INSERT INTO`, `MERGE`, `UNLOAD`, and DDL queries are not supported.
- Apache Hive, Apache Hudi, Apache Iceberg, and Linux Foundation Delta Lake tables registered with AWS Glue are supported. External Hive metastores are not supported.

- Queries that reference federated catalogs or an external Hive metastore are not supported.
- Query result reuse is not supported for Lake Formation governed tables.
- Query result reuse is not supported when the Amazon S3 location of the table source is registered as a data location in Lake Formation.
- Tables with row and column permissions are not supported.
- Tables that have fine grained access control (for example, column or row filtering) are not supported.
- Any query that references an unsupported table is not eligible for query result reuse.
- Athena requires that you have Amazon S3 read permissions for the previously generated output file to be reused.
- The reuse query results feature assumes that content of previous result has not been modified. Athena does not check the integrity of a previous result before using it.
- If the query results from the previous execution have been deleted or moved to different location in Amazon S3, the subsequent execution of the same query will not reuse the query results.
- Potentially stale results can be returned. Athena does not check for changes in source data until the maximum reuse age that you specify has been reached.
- If multiple results are available for reuse, Athena uses the latest result.
- Queries that use non-deterministic operators or functions like `rand()` or `shuffle()` do not use cached results. For example, `LIMIT` without `ORDER BY` is non-deterministic and not cached, but `LIMIT` with `ORDER BY` is deterministic and is cached.
- Query result reuse is supported in the Athena console, in the Athena API, and in the JDBC driver. Currently, ODBC driver support for query result reuse is available only for Windows.
- To use the query result reuse feature with JDBC, the minimum required driver version is 2.0.34.1000. For ODBC, the minimum required driver version is 1.1.19.1002. For driver download information, see Connect to Amazon Athena with ODBC and JDBC drivers.
- Query result reuse is not supported for queries that use more than one data catalog.
- Query result reuse is not supported for queries that include more than 20 tables.

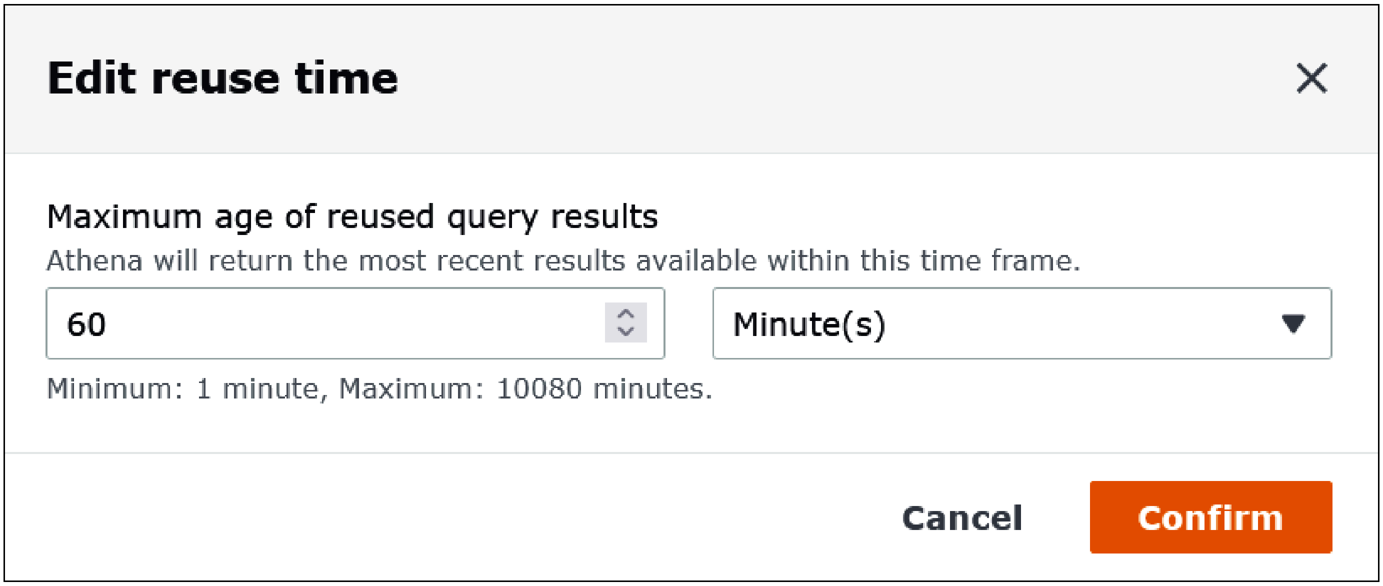
How to reuse query results in the Athena console

To use the feature, enable the **Reuse query results** option in the Athena query editor.

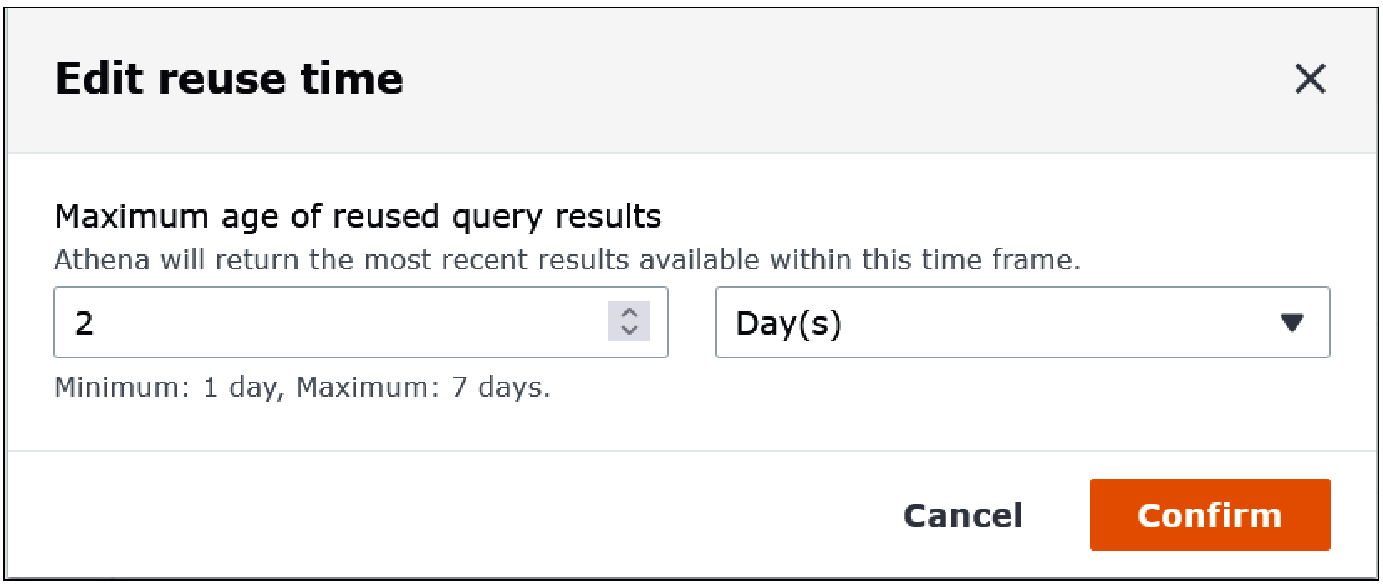


To configure the reuse query results feature

1. In the Athena query editor, under the **Reuse query results** option, choose the edit icon next to **up to 60 minutes ago**.
2. In the **Edit reuse time** dialog box, from the box on the right, choose a time unit (minutes, hours, or days).
3. In the box on the left, enter or choose the number of time units that you want to specify. The maximum time you can enter is the equivalent of seven days regardless of the time unit chosen.

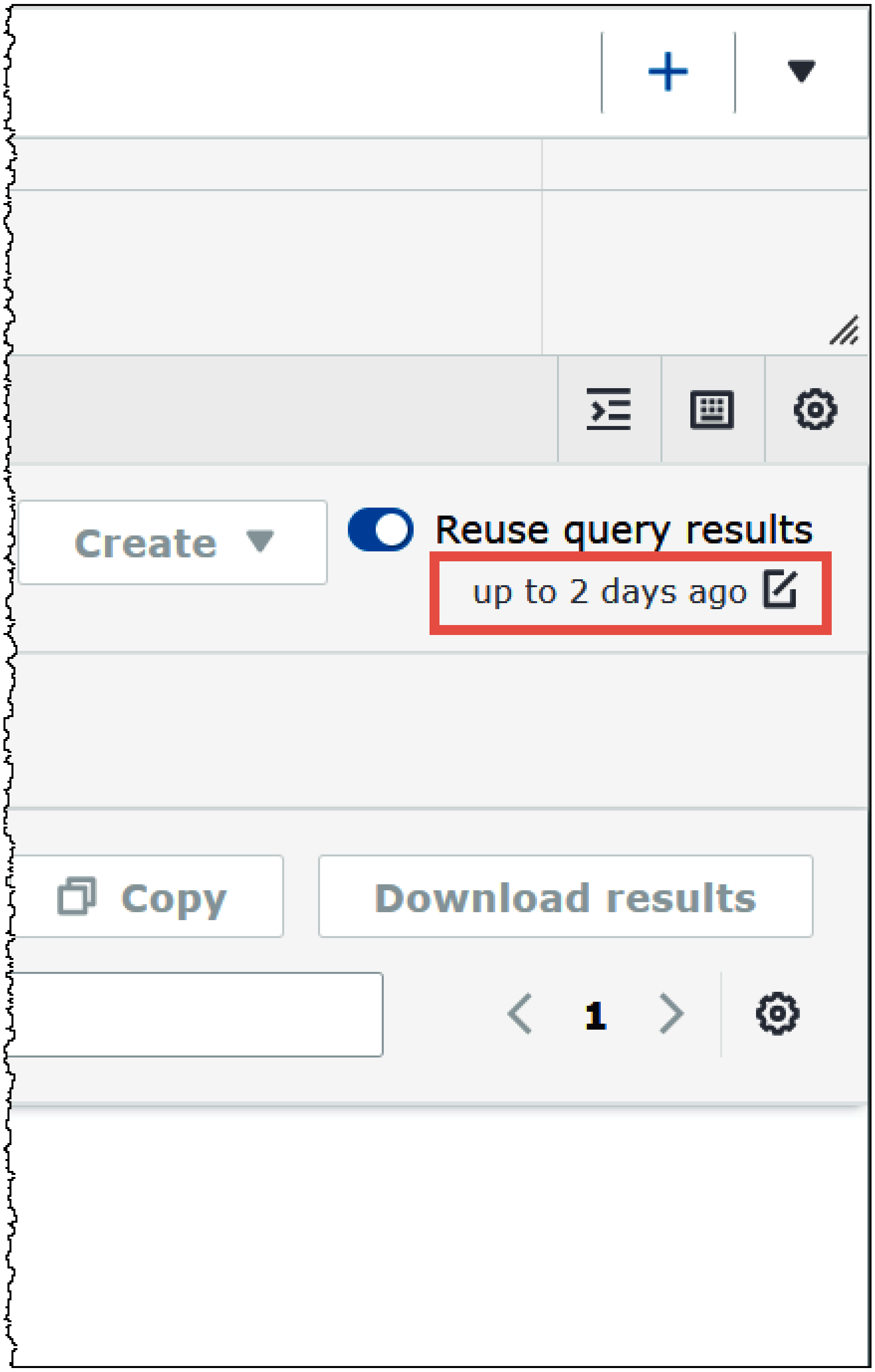


The following example specifies a maximum reuse time of two days.



4. Choose **Confirm**.

   A banner confirms your configuration change, and the **Reuse query results** option displays your new setting.

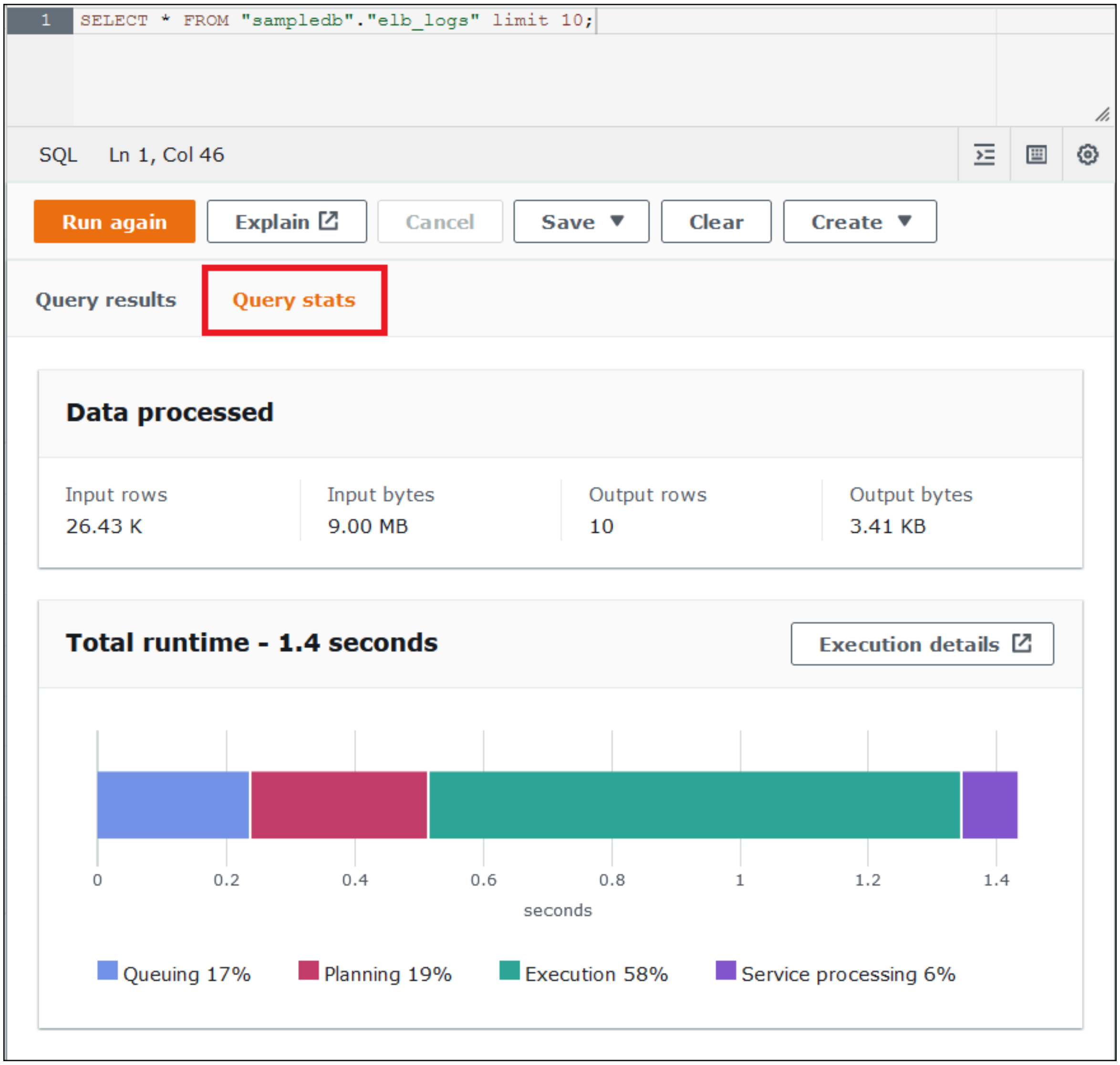


View statistics and execution details for completed queries

After you run a query, you can get statistics on the input and output data processed, see a graphical representation of the time taken for each phase of the query, and interactively explore execution details.

To view query statistics for a completed query

1. After you run a query in the Athena query editor, choose the **Query stats** tab.



The **Query stats** tab provides the following information:

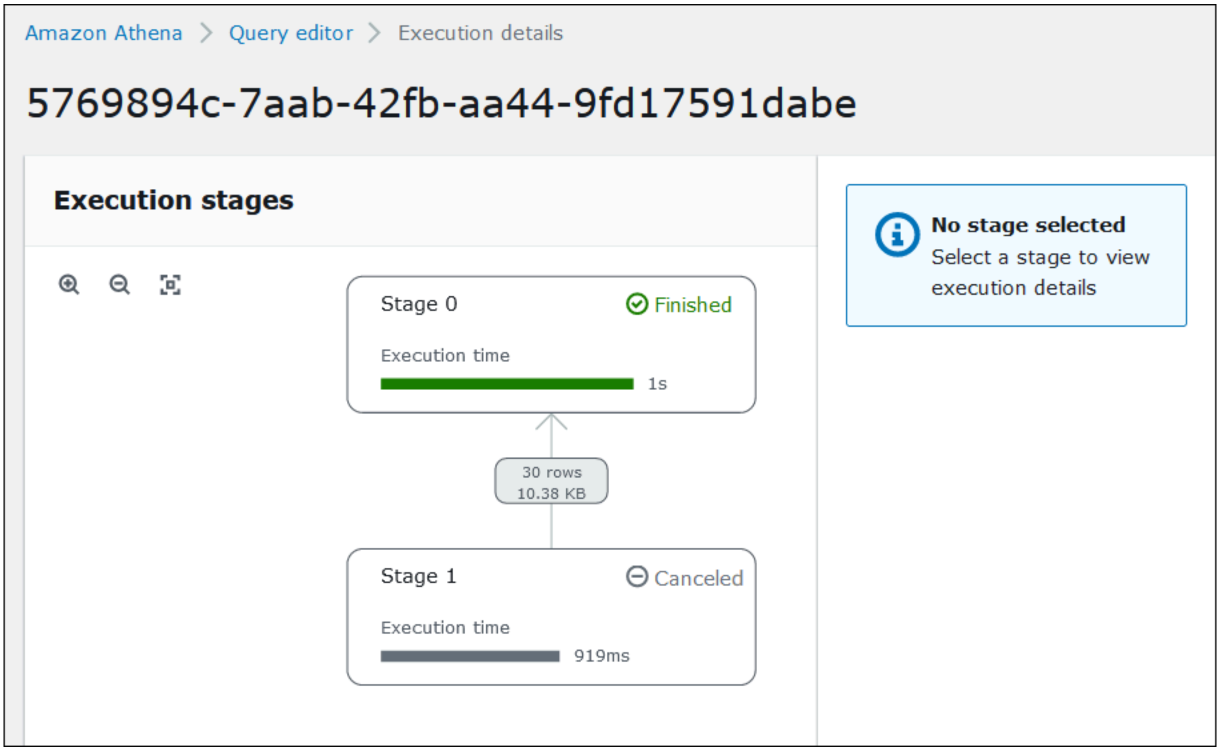
- **Data processed** – Shows you the number of input rows and bytes processed, and the number of rows and bytes output.
- **The Total runtime** – Shows the total amount of time the query took to run and a graphical representation of how much of that time was spent on queuing, planning, execution, and service processing.

> **Note**
>
> Stage-level input and output row count and data size information are not shown when a query has row-level filters defined in Lake Formation.

2. To interactively explore information about how the query ran, choose **Execution details**.

The **Execution details** page shows the execution ID for the query and a graph of the zero-based stages in the query. The stages are ordered start to finish from bottom to top. Each stage's label shows the amount of time the stage took to run.

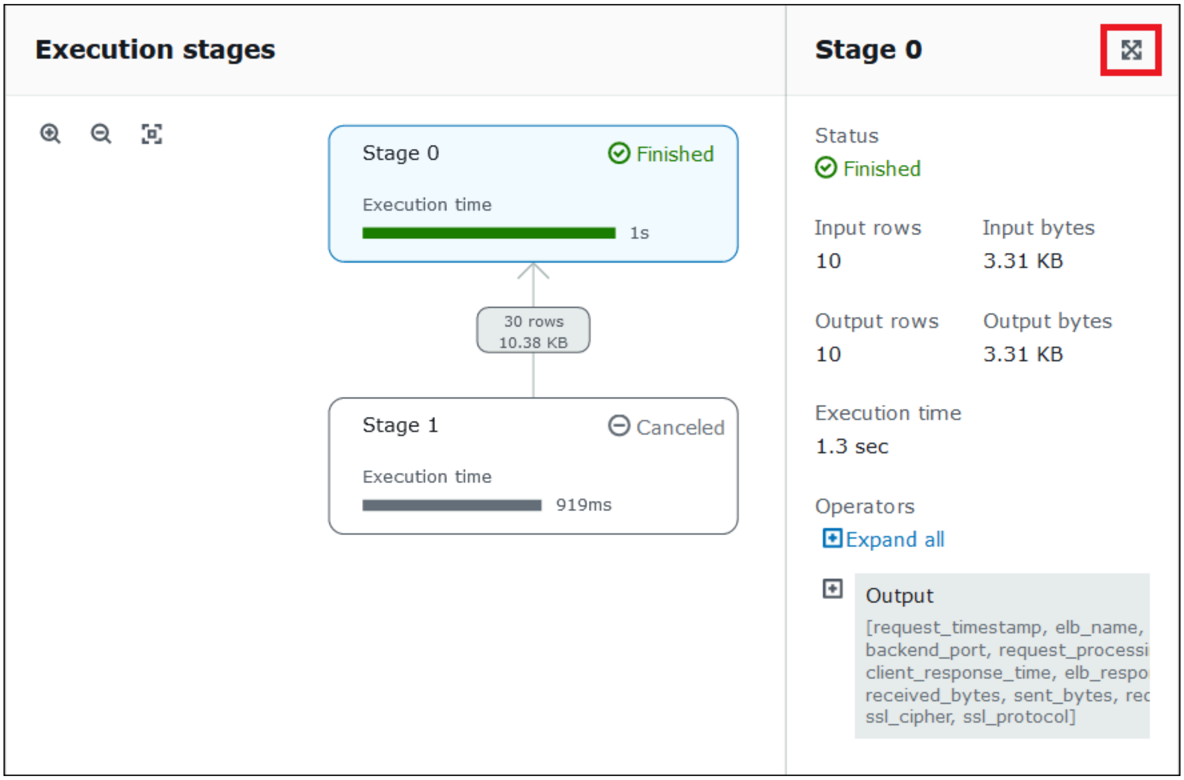
> **Note**
>
> The total runtime and execution stage time of a query often differ significantly. For example, a query with a total runtime in minutes can show an execution time for a stage in hours. Because a stage is a logical unit of computation executed in parallel across many tasks, the execution time of a stage is the aggregate execution time of all of its tasks. Despite this discrepancy, stage execution time can be useful as a relative indicator of which stage was most computationally intensive in a query.



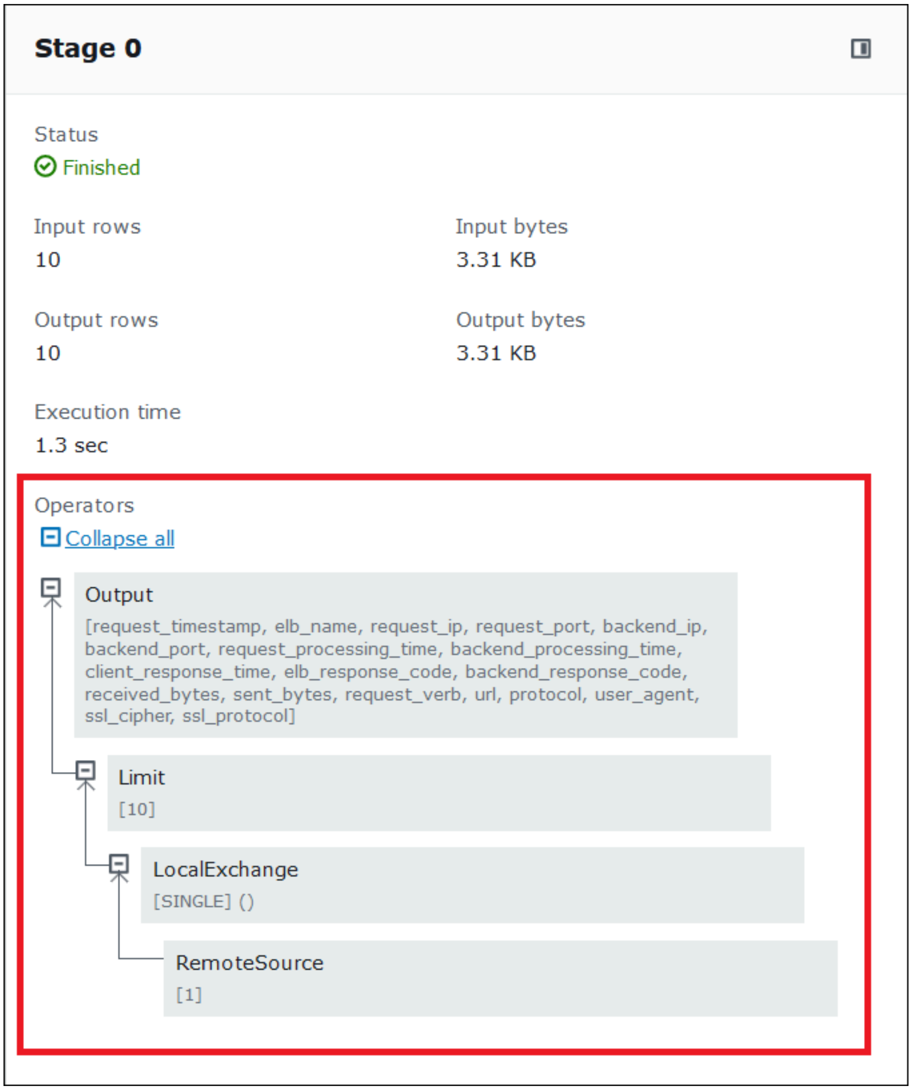
To navigate the graph, use the following options:

- To zoom in or out, scroll the mouse, or use the magnifying icons.
- To adjust the graph to fit the screen, choose the **Zoom to fit** icon.
- To move the graph around, drag the mouse pointer.

3. To see more details for a stage, choose the stage. The stage details pane on the right shows the number of rows and bytes input and output, and an operator tree.



4. To see the stage details full width, choose the expand icon at the top right of the details pane.
5. To get information about the parts of the stage, expand one or more items in the operator tree.



For more information about execution details, see Understand Athena EXPLAIN statement results.

Additional resources

For more information, see the following resources.

View execution plans for SQL queries

Using EXPLAIN and EXPLAIN ANALYZE in Athena

Work with views

A view in Amazon Athena is a logical table, not a physical table. The query that defines a view runs each time the view is referenced in a query. You can create a view from a `SELECT` query and then reference this view in future queries.

You can use two different kinds of views in Athena: Athena views and AWS Glue Data Catalog views.

When to use Athena views?

You may want to create Athena views to:

- **Query a subset of data** – For example, you can create a view with a subset of columns from the original table to simplify querying data.
- **Combine tables** – You can use views to combine multiple tables into one query. When you have multiple tables and want to combine them with `UNION ALL`, you can create a view with that expression to simplify queries against the combined tables.
- **Hide complexity** – Use views to hide the complexity of existing base queries and simplify queries run by users. Base queries often include joins between tables, expressions in the column list, and other SQL syntax that make it difficult to understand and debug them. You might create a view that hides the complexity and simplifies queries.
- **Optimize queries** – You can use views to experiment with optimization techniques to create optimized queries. For example, if you find a combination of `WHERE` conditions, `JOIN` order, or other expressions that demonstrate the best performance, you can create a view with these clauses and expressions. Applications can then make relatively simple queries against this view. If you later find a better way to optimize the original query, when you recreate the view, all the applications immediately take advantage of the optimized base query.
- **Hide underlying names** – You can use views to hide underlying table and column names, and minimize maintenance problems if the names change. If the names change, you can simply recreate the view using the new names. Queries that use the view rather than the tables directly keep running with no changes.

  For more information, see [Work with Athena views](#).

When to use AWS Glue Data Catalog views?

Use AWS Glue Data Catalog views when you want a single common view across AWS services like Amazon Athena and Amazon Redshift. In Data Catalog views, access permissions are defined by the user who created the view instead of the user who queries the view. This method of granting permissions is called *definer* semantics.

The following use cases show how you can use Data Catalog views.

- **Greater access control –** You create a view that restricts data access based on the level of permissions the user requires. For example, you can use Data Catalog views to prevent employees who don't work in the human resources (HR) department from seeing personally identifiable information.
- **Ensure complete records –** By applying certain filters onto your Data Catalog view, you make sure that the data records in a Data Catalog view are always complete.
- **Enhanced security –** In Data Catalog views, the query definition that creates the view must be intact in order for the view to be created. This makes Data Catalog views less susceptible to SQL commands from malicious actors.
- **Prevent access to underlying tables –** Definer semantics allow users to access a view without making the underlying table available to them. Only the user who defines the view requires access to the tables.

Data Catalog view definitions are stored in the AWS Glue Data Catalog. This means that you can use AWS Lake Formation to grant access through resource grants, column grants, or tag-based access controls. For more information about granting and revoking access in Lake Formation, see Granting and revoking permissions on Data Catalog resources in the *AWS Lake Formation Developer Guide*.

For more information, see Use Data Catalog views in Athena.

Work with Athena views

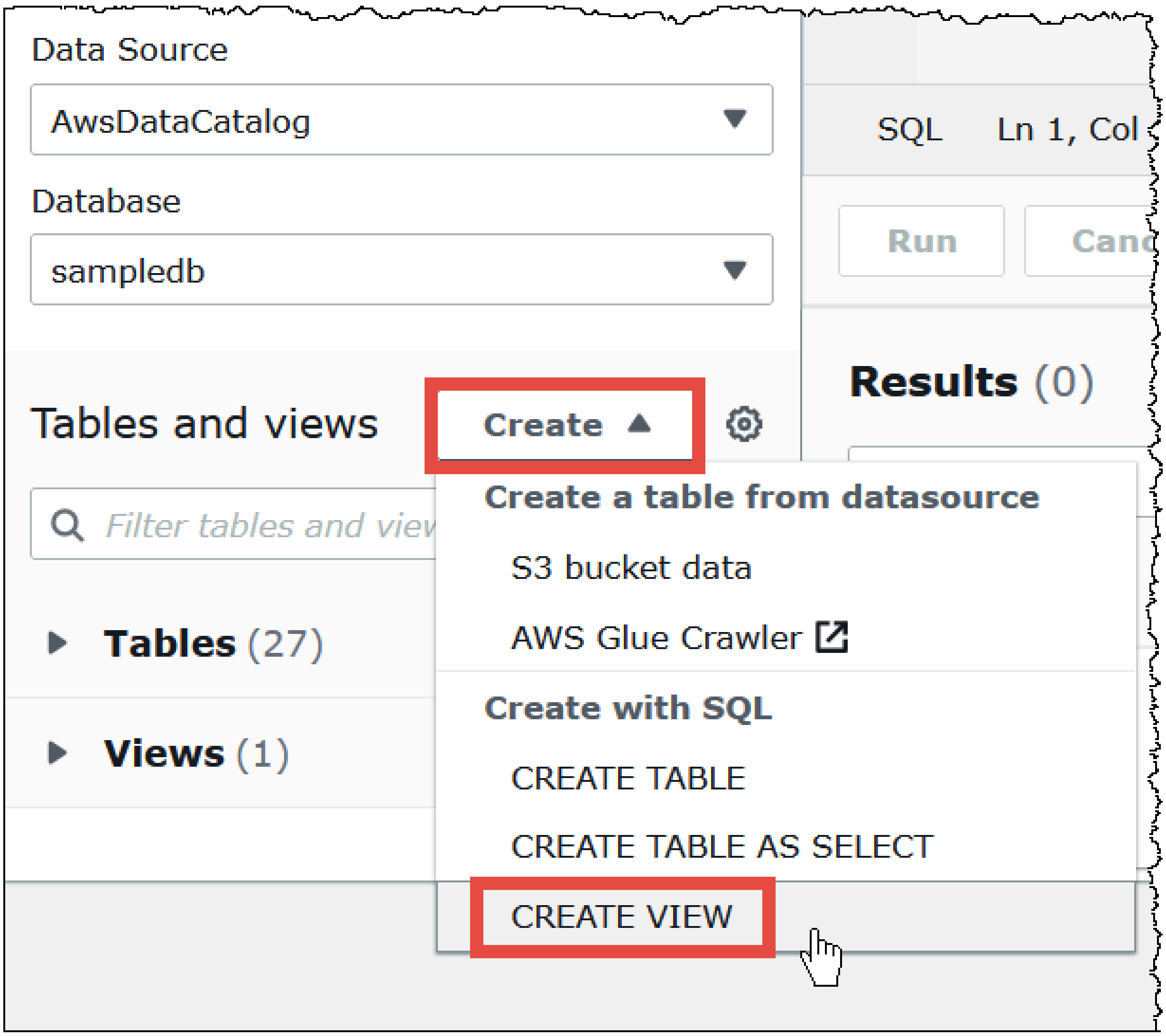
Athena views can be easily created, updated, and managed in the Athena console.

Create views

You can create a view in the Athena console by using a template or by running an existing query.

To use a template to create a view

1. In the Athena console, next to **Tables and views**, choose **Create**, and then choose **Create view**.



   This action places an editable view template into the query editor.

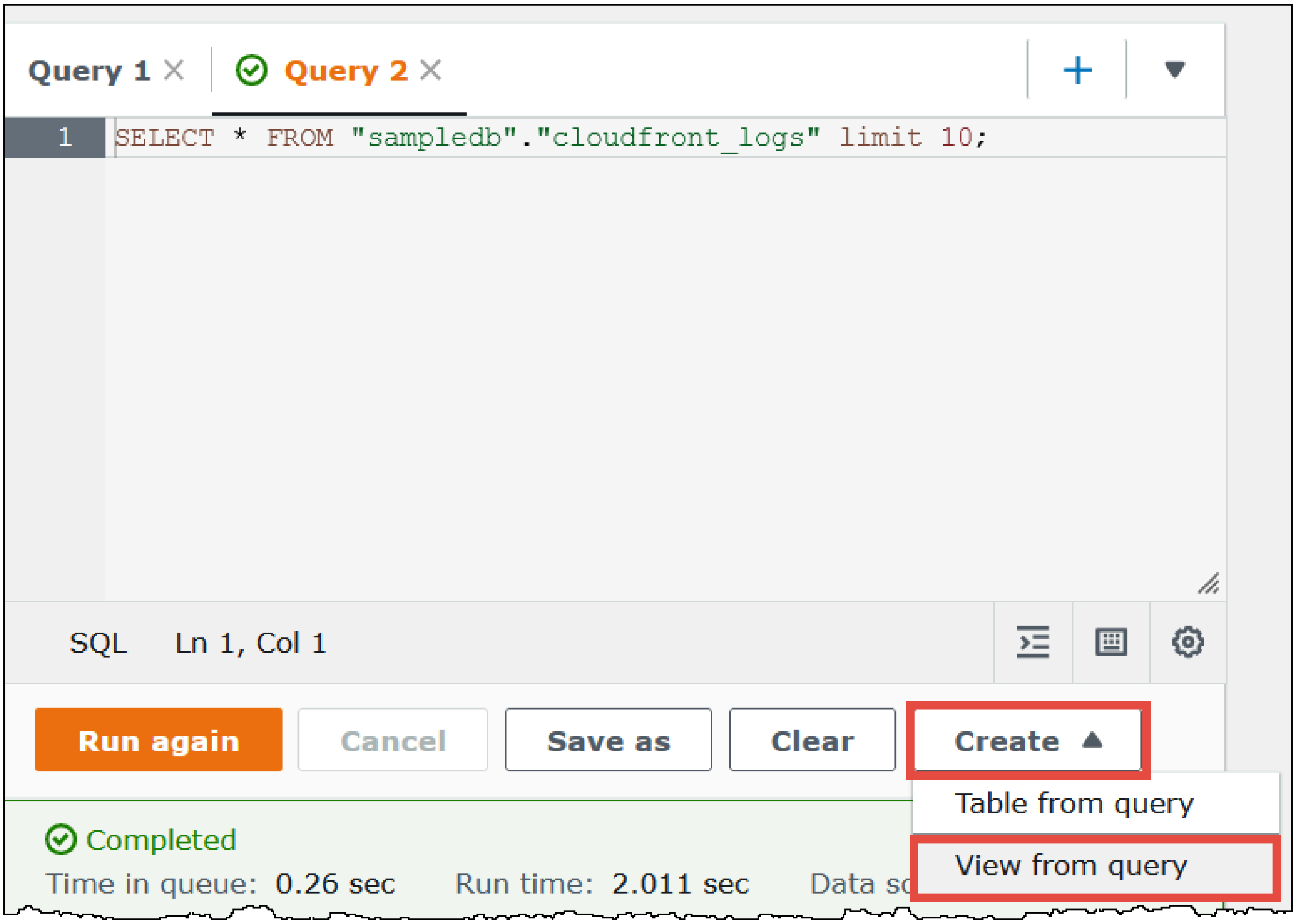
2. Edit the view template according to your requirements. When you enter a name for the view in the statement, remember that view names cannot contain special characters other than underscore (_). See Name databases, tables, and columns. Avoid using Escape reserved keywords in queries for naming views.

   For more information about creating views, see CREATE VIEW and CREATE DIALECT VIEW and Examples of Athena views.

3. Choose **Run** to create the view. The view appears in the list of views in the Athena console.

To create a view from an existing query

1. Use the Athena query editor to run an existing query.
2. Under the query editor window, choose **Create**, and then choose **View from query**.



3. In the **Create View** dialog box, enter a name for the view, and then choose **Create**. View names cannot contain special characters other than underscore (_). See Name databases, tables, and columns. Avoid using Escape reserved keywords in queries for naming views.

   Athena adds the view to the list of views in the console and displays the `CREATE VIEW` statement for the view in the query editor.

Notes

- If you delete a table on which a table is based and then attempt to run the view, Athena displays an error message.

- You can create a nested view, which is a view on top of an existing view. Athena prevents you from running a recursive view that references itself.

Examples of Athena views

To show the syntax of the view query, use SHOW CREATE VIEW.

Example Example 1

Consider the following two tables: a table `employees` with two columns, `id` and `name`, and a table `salaries`, with two columns, `id` and `salary`.

In this example, we create a view named `name_salary` as a `SELECT` query that obtains a list of IDs mapped to salaries from the tables `employees` and `salaries`:

```
CREATE VIEW name_salary AS
SELECT
 employees.name,
 salaries.salary
FROM employees, salaries
WHERE employees.id = salaries.id
```

Example Example 2

In the following example, we create a view named `view1` that enables you to hide more complex query syntax.

This view runs on top of two tables, `table1` and `table2`, where each table is a different `SELECT` query. The view selects columns from `table1` and joins the results with `table2`. The join is based on column `a` that is present in both tables.

```
CREATE VIEW view1 AS
WITH
  table1 AS (
         SELECT a,
         MAX(b) AS the_max
         FROM x
         GROUP BY a
         ),
  table2 AS (
```

```
          SELECT a,
          AVG(d) AS the_avg
          FROM y
          GROUP BY a)
SELECT table1.a, table1.the_max, table2.the_avg
FROM table1
JOIN table2
ON table1.a = table2.a;
```

For information about querying federated views, see Query federated views.

Manage Athena views

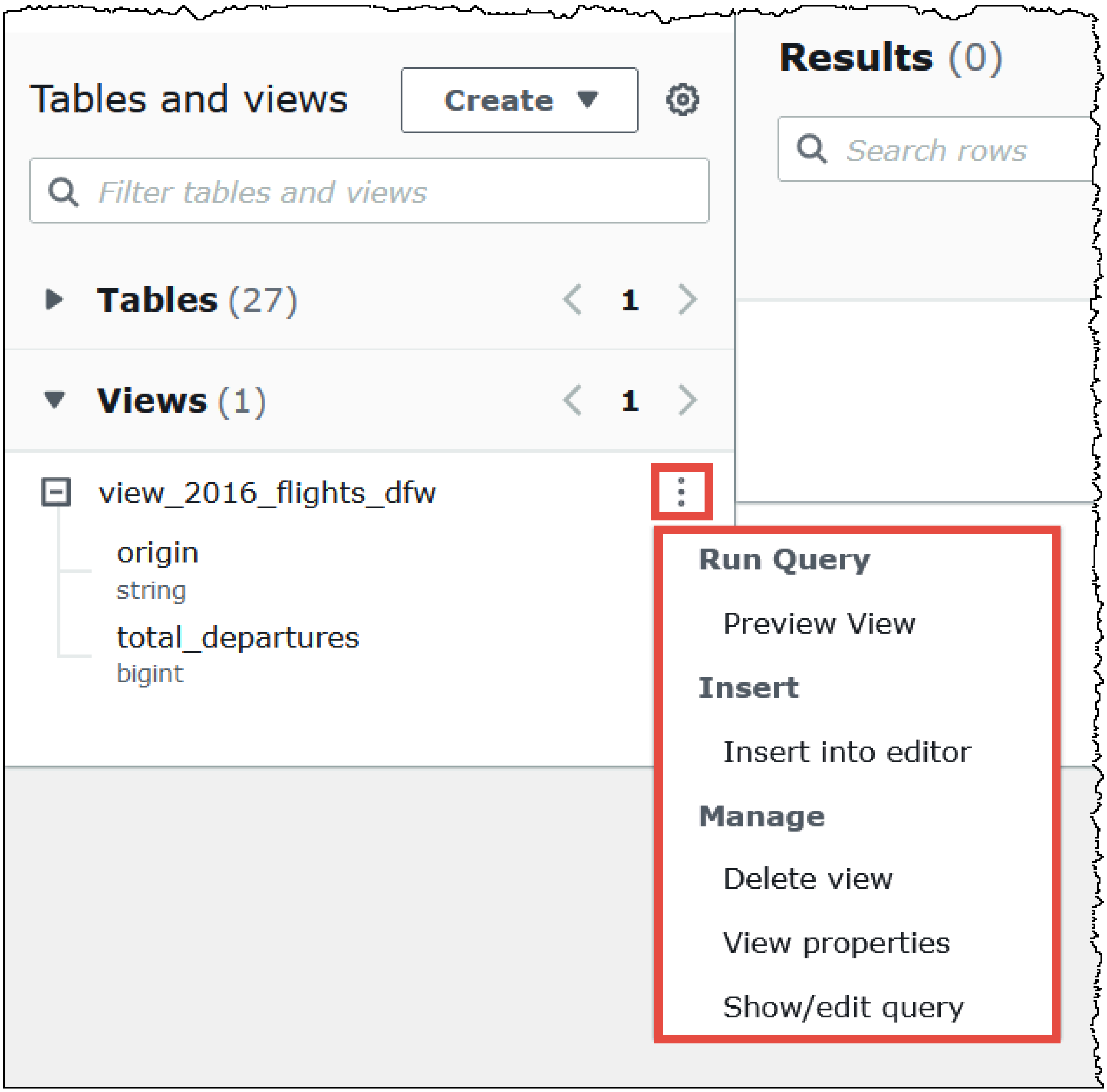
In the Athena console, you can:

- Locate all views in the left pane, where tables are listed.
- Filter views.
- Preview a view, show its properties, edit it, or delete it.

To show the actions for a view

A view shows in the console only if you have already created it.

1. In the Athena console, choose **Views**, and then choose a view to expand it and show the columns in the view.
2. Choose the three vertical dots next to the view to show a list of actions for the view.



3. Choose actions to preview the view, insert the view name into the query editor, delete the view, see the view's properties, or display and edit the view in the query editor.

Supported DDL actions for Athena views

Athena supports the following management actions for views.

Statement	Description
CREATE VIEW and CREATE DIALECT VIEW	Creates a new view from a specified `SELECT` query. For more information, see Create views. The optional `OR REPLACE` clause lets you update the existing view by replacing it.
DESCRIBE VIEW	Shows the list of columns for the named view. This allows you to examine the attributes of a complex view.
DROP VIEW	Deletes an existing view. The optional `IF EXISTS` clause suppresses the error if the view does not exist.
SHOW CREATE VIEW	Shows the SQL statement that creates the specified view.
SHOW VIEWS	Lists the views in the specified database, or in the current database if you omit the database name. Use the optional `LIKE` clause with a regular expression to restrict the list of view names. You can also see the list of views in the left pane in the console.
SHOW COLUMNS	Lists the columns in the schema for a view.

Considerations and limitations for Athena views

Athena views have the following considerations and limitations.

Considerations

The following considerations apply to creating and using views in Athena:

- In Athena, you can preview and work with views created in the Athena Console, in the AWS Glue Data Catalog or with Presto running on the Amazon EMR cluster connected to the same catalog.
- If you have created Athena views in the Data Catalog, then Data Catalog treats views as tables. You can use table level fine-grained access control in Data Catalog to restrict access to these views.
- Athena prevents you from running recursive views and displays an error message in such cases. A recursive view is a view query that references itself.

- Athena displays an error message when it detects stale views. A stale view is reported when one of the following occurs:
  - The view references tables or databases that do not exist.
  - A schema or metadata change is made in a referenced table.
  - A referenced table is dropped and recreated with a different schema or configuration.
- You can create and run nested views as long as the query behind the nested view is valid and the tables and databases exist.

Limitations

- Athena view names cannot contain special characters, other than underscore `(_)`. For more information, see Name databases, tables, and columns.
- Avoid using reserved keywords for naming views. If you use reserved keywords, use double quotes to enclose reserved keywords in your queries on views. See Escape reserved keywords in queries.
- You cannot use views created in Athena with external Hive metastores or UDFs. For information about working with views created externally in Hive, see Work with Hive views.
- You cannot use views with geospatial functions.
- You cannot use views to manage access control on data in Amazon S3. To query a view, you need permissions to access the data stored in Amazon S3. For more information, see Control access to Amazon S3 from Athena.
- While querying views across accounts is supported in Athena engine version 3, you cannot create a view that includes a cross-account AWS Glue Data Catalog. For information about cross-account data catalog access, see Configure cross-account access to AWS Glue data catalogs.
- The Hive or Iceberg hidden metadata columns `$bucket`, `$file_modified_time`, `$file_size`, and `$partition` are not supported for views in Athena. For information about using the `$path` metadata column in Athena, see Getting the file locations for source data in Amazon S3 .

Use Data Catalog views in Athena

Creating Data Catalog views in Amazon Athena requires a special `CREATE VIEW` statement. Querying them uses conventional SQL `SELECT` syntax. Data Catalog views are also referred to as *multi dialect* views, or MDVs.

Create a Data Catalog view

To create a Data Catalog view in Athena, use the following syntax.

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
SECURITY DEFINER
[ SHOW VIEW JSON ]
AS athena-sql-statement
```

> **Note**
>
> The `SHOW VIEW JSON` option applies to Data Catalog views only and not to Athena views. Using the `SHOW VIEW JSON` option performs a "dry run" that validates the input and, if the validation succeeds, returns the JSON of the AWS Glue table object that will represent the view. The actual view is not created. If the `SHOW VIEW JSON` option is not specified, validations are done and the view is created as usual in the Data Catalog.

The following example shows how a user of the `Definer` role creates the `orders_by_date` Data Catalog view. The example assumes that the `Definer` role has full `SELECT` permissions on the `orders` table in the `default` database.

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

For syntax information, see CREATE PROTECTED MULTI DIALECT VIEW.

Query a Data Catalog view

After the view is created, the `Lake Formation` admin can grant `SELECT` permissions on the Data Catalog view to the `Invoker` principals. The `Invoker` principals can then query the view without having access to the underlying base tables referenced by the view. The following is an example `Invoker` query.

```
SELECT * from orders_by_date where price > 5000
```

Considerations and limitations

Most of the following Data Catalog view limitations are specific to Athena. For additional limitations on Data Catalog views that also apply to other services, see the Lake Formation documentation.

- Data Catalog views cannot reference other views, database resource links, or table resource links.
- You can reference up to 10 tables in the view definition.
- Tables must not have the `IAMAllowedPrincipals` data lake permission in Lake Formation. If present, the error Multi Dialect views may only reference tables without IAMAllowedPrincipals permissions occurs.
- The table's Amazon S3 location must be registered as a Lake Formation data lake location. If the table is not so registered, the error Multi Dialect views may only reference Lake Formation managed tables occurs. For information about how to register Amazon S3 locations in Lake Formation, see Registering an Amazon S3 location in the *AWS Lake Formation Developer Guide*.
- The AWS Glue GetTables and SearchTables API calls do not update the `IsRegisteredWithLakeFormation` parameter. To view the correct value for the parameter, use the AWS Glue GetTable API. For more information, see GetTables and SearchTables APIs do not update the value for the IsRegisteredWithLakeFormation parameter in the *AWS Lake Formation Developer Guide*.
- The `DEFINER` principal can be only an IAM role.
- The `DEFINER` role must have full `SELECT` (grantable) permissions on the underlying tables.
- `UNPROTECTED` Data Catalog views are not supported.
- User-defined functions (UDFs) are not supported in the view definition.
- Athena federated data sources cannot be used in Data Catalog views.
- Data Catalog views are not supported for external Hive metastores.
- Athena displays an error message when it detects stale views. A stale view is reported when one of the following occurs:
  - The view references tables or databases that do not exist.
  - A schema or metadata change is made in a referenced table.
  - A referenced table is dropped and recreated with a different schema or configuration.

Permissions

Data Catalog views require three roles: `Lake Formation Admin`, `Definer`, and `Invoker`.

- **`Lake Formation Admin`** – Has access to configure all Lake Formation permissions.
- **`Definer`** – Creates the Data Catalog view. The `Definer` role must have full grantable `SELECT` permissions on all underlying tables that the view definition references.
- **`Invoker`** – Can query the Data Catalog view or check its metadata. To show the invoker of a query, you can use the `invoker_principal()` DML function. For more information, see invoker_principal().

The `Definer` role's trust relationships must allow the `sts:AssumeRole` action for the AWS Glue and Lake Formation service principals. For more information, see Prerequisites for creating views in the *AWS Lake Formation Developer Guide*.

IAM permissions for Athena access are also required. For more information, see AWS managed policies for Amazon Athena.

Manage Data Catalog views

You can use DDL commands to update and manage your Data Catalog views.

Update a Data Catalog view

The `Lake Formation` admin or definer can use the `ALTER VIEW UPDATE DIALECT` syntax to update the view definition. The following example modifies the view definition to select columns from the `returns` table instead of the `orders` table.

```
ALTER VIEW orders_by_date UPDATE DIALECT
AS
SELECT return_date, sum(totalprice) AS price
FROM returns
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

Supported DDL actions for AWS Glue Data Catalog views

Athena supports the following actions for AWS Glue Data Catalog views.

Statement	Description
ALTER VIEW DIALECT	Updates a Data Catalog view by either adding an engine dialect or by updating or dropping an existing engine dialect.

Statement	Description
CREATE PROTECTED MULTI DIALECT VIEW	Creates a Data Catalog view from a specified `SELECT` query. For more information, see CREATE PROTECTED MULTI DIALECT VIEW. The optional `OR REPLACE` clause lets you update the existing view by replacing it.
DESCRIBE VIEW	Shows the list of columns for the named view. This allows you to examine the attributes of a complex view.
DROP VIEW	Deletes an existing view. The optional `IF EXISTS` clause suppresses the error if the view does not exist.
SHOW CREATE VIEW	Shows the SQL statement that creates the specified view.
SHOW VIEWS	Lists the views in the specified database, or in the current database if you omit the database name. Use the optional `LIKE` clause with a regular expression to restrict the list of view names. You can also see the list of views in the left pane in the console.
SHOW COLUMNS	Lists the columns in the schema for a view.

Use saved queries

You can use the Athena console to save, edit, run, rename, and delete the queries that you create in the query editor.

Considerations and limitations

- You can update the name, description, and query text of saved queries.
- You can only update the queries in your own account.
- You cannot change the workgroup or database to which the query belongs.
- Athena does not keep a history of query modifications. If you want to keep a particular version of a query, save it with a different name.

> **Note**
>
> Amazon Athena resources can now be accessed within Amazon SageMaker Unified Studio (Preview), which helps you access your organization's data and act on it with the best tools. You can migrate saved queries from an Athena workgroup to a SageMaker Unified Studio project, configure projects with existing Athena workgroups, and maintain necessary permissions through IAM role updates. For more information, see Migrating Amazon Athena resources to Amazon SageMaker Unified Studio (Preview).

Topics

Save a query with a name

To save a query and give it a name

1. In the Athena console query editor, enter or run a query.
2. Above the query editor window, on the tab for the query, choose the three vertical dots, and then choose **Save as**.
3. In the **Save query** dialog box, enter a name for the query and an optional description. You can use the expandable **Preview SQL query** window to verify the contents of the query before you save it.
4. Choose **Save query**.

   In the query editor, the tab for the query shows the name that you specified.

Run a saved query

To run a saved query

1. In the Athena console, choose the **Saved queries** tab.
2. In the **Saved queries** list, choose the ID of the query that you want to run.

   The query editor displays the query that you chose.
3. Choose **Run**.

Edit a saved query

To edit a saved query

1. In the Athena console, choose the **Saved queries** tab.
2. In the **Saved queries** list, choose the ID of the query that you want to edit.
3. Edit the query in the query editor.
4. Perform one of the following steps:

   - To run the query, choose **Run**.
   - To save the query, choose the three vertical dots on the tab for the query, and then choose **Save**.
   - To save the query with a different name, choose the three vertical dots on the tab for the query, and then choose **Save as**.

Rename or delete a saved query

To rename or delete a saved query already displayed in the query editor

1. Choose the three vertical dots on the tab for the query, and then choose **Rename** or **Delete**.
2. Follow the prompts to rename or delete the query.

Rename an undisplayed saved query

To rename a saved query not displayed in the query editor

1. In the Athena console, choose the **Saved queries** tab.

2. Select the check box for the query that you want to rename.
3. Choose **Rename**.
4. In the **Rename query** dialog box, edit the query name and query description. You can use the expandable **Preview SQL query** window to verify the contents of the query before you rename it.
5. Choose **Rename query**.

   The renamed query appears in the **Saved queries** list.

Delete an undisplayed saved query

To delete a saved query not displayed in the query editor

1. In the Athena console, choose the **Saved queries** tab.
2. Select one or more check boxes for the queries that you want to delete.
3. Choose **Delete**.
4. At the confirmation prompt, choose **Delete**.

   One or more queries are removed from the **Saved queries** list.

Use the Athena API to update saved queries

For information about using the Athena API to update a saved query, see the [UpdateNamedQuery] action in the Athena API Reference.

Use parameterized queries

You can use Athena parameterized queries to re-run the same query with different parameter values at execution time and help prevent SQL injection attacks. In Athena, parameterized queries can take the form of execution parameters in any DML query or SQL prepared statements.

- Queries with execution parameters can be done in a single step and are not workgroup specific. You place question marks in any DML query for the values that you want to parameterize. When you run the query, you declare the execution parameter values sequentially. The declaration of parameters and the assigning of values for the parameters can be done in the same query, but in a decoupled fashion. Unlike prepared statements, you can select the workgroup when you submit a query with execution parameters.

- Prepared statements require two separate SQL statements: `PREPARE` and `EXECUTE`. First, you define the parameters in the `PREPARE` statement. Then, you run an `EXECUTE` statement that supplies the values for the parameters that you defined. Prepared statements are workgroup specific; you cannot run them outside the context of the workgroup to which they belong.

Considerations and limitations

- Parameterized queries are supported in Athena engine version 2 and later versions. For information about Athena engine versions, see Athena engine versioning.
- Currently, parameterized queries are supported only for `SELECT`, `INSERT INTO`, `CTAS`, and `UNLOAD` statements.
- In parameterized queries, parameters are positional and are denoted by `?`. Parameters are assigned values by their order in the query. Named parameters are not supported.
- Currently, `?` parameters can be placed only in the `WHERE` clause. Syntax like `SELECT ? FROM table` is not supported.
- Question mark parameters cannot be placed in double or single quotes (that is, `'?'` and `"?"` are not valid syntax).
- For SQL execution parameters to be treated as strings, they must be enclosed in single quotes rather than double quotes.
- If necessary, you can use the `CAST` function when you enter a value for a parameterized term. For example, if you have a column of the `date` type that you have parameterized in a query and you want to query for the date `2014-07-05`, entering `CAST('2014-07-05' AS DATE)` for the parameter value will return the result.
- Prepared statements are workgroup specific, and prepared statement names must be unique within the workgroup.
- IAM permissions for prepared statements are required. For more information, see Configure access to prepared statements.
- Queries with execution parameters in the Athena console are limited to a maximum of 25 question marks.

Topics

- Additional resources

Use execution parameters

You can use question mark placeholders in any DML query to create a parameterized query without creating a prepared statement first. To run these queries, you can use the Athena console, or use the AWS CLI or the AWS SDK and declare the variables in the `execution-parameters` argument.

Topics

Run queries with execution parameters in the Athena console

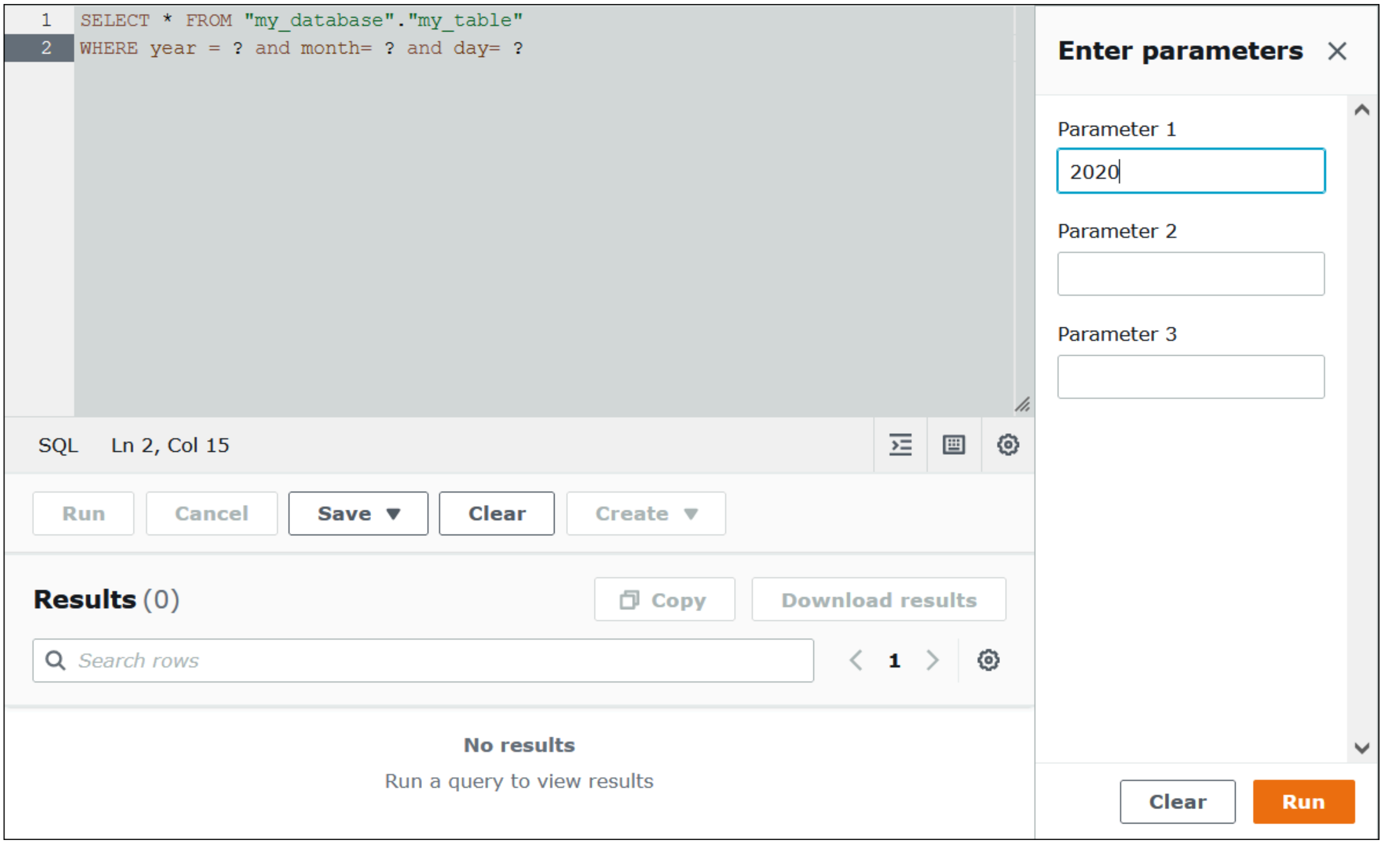
When you run a parameterized query that has execution parameters (question marks) in the Athena console, you are prompted for the values in the order in which the question marks occur in the query.

To run a query that has execution parameters

1. Enter a query with question mark placeholders in the Athena editor, as in the following example.

```
SELECT * FROM "my_database"."my_table"
WHERE year = ? and month= ? and day= ?
```

2. Choose **Run.**
3. In the **Enter parameters** dialog box, enter a value in order for each of the question marks in the query.



4. When you are finished entering the parameters, choose **Run**. The editor shows the query results for the parameter values that you entered.

At this point, you can do one of the following:

- Enter different parameter values for the same query, and then choose **Run again**.
- To clear all of the values that you entered at once, choose **Clear**.
- To edit the query directly (for example, to add or remove question marks), close the **Enter parameters** dialog box first.
- To save the parameterized query for later use, choose **Save** or **Save as**, and then give the query a name. For more information about using saved queries, see Use saved queries.

As a convenience, the **Enter parameters** dialog box remembers the values that you entered previously for the query as long as you use the same tab in the query editor.

Run queries with execution parameters using the AWS CLI

To use the AWS CLI to run queries with execution parameters, use the `start-query-execution` command and provide a parameterized query in the `query-string` argument. Then, in the `execution-parameters` argument, provide the values for the execution parameters. The following example illustrates this technique.

```
aws athena start-query-execution
--query-string "SELECT * FROM table WHERE x = ? AND y = ?"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://amzn-s3-demo-bucket;/..."
--execution-parameters "1" "2"
```

Use prepared statements

You can use a prepared statement for repeated execution of the same query with different query parameters. A prepared statement contains parameter placeholders whose values are supplied at execution time.

> **Note**
>
> The maximum number of prepared statements in a workgroup is 1000.

Topics

SQL syntax for prepared statements

You can use the `PREPARE`, `EXECUTE` and `DEALLOCATE PREPARE` SQL statements to run parameterized queries in the Athena console query editor.

- To specify parameters where you would normally use literal values, use question marks in the `PREPARE` statement.
- To replace the parameters with values when you run the query, use the `USING` clause in the `EXECUTE` statement.

- To remove a prepared statement from the prepared statements in a workgroup, use the DEALLOCATE PREPARE statement.

The following sections provide additional detail about each of these statements.

Topics

- PREPARE
- EXECUTE
- DEALLOCATE PREPARE

PREPARE

Prepares a statement to be run at a later time. Prepared statements are saved in the current workgroup with the name that you specify. The statement can include parameters in place of literals to be replaced when the query is run. Parameters to be replaced by values are denoted by question marks.

Syntax

```
PREPARE statement_name FROM statement
```

The following table describes these parameters.

Parameter	Description
statement_name	The name of the statement to be prepared. The name must be unique within the workgroup.
statement	A SELECT, CTAS, or INSERT INTO query.

PREPARE examples

The following examples show the use of the PREPARE statement. Question marks denote the values to be supplied by the EXECUTE statement when the query is run.

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
```

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

```
PREPARE my_unload FROM
UNLOAD (SELECT * FROM table1 WHERE productid < ?)
TO 's3://amzn-s3-demo-bucket/'
WITH (format='PARQUET')
```

EXECUTE

Runs a prepared statement. Values for parameters are specified in the `USING` clause.

Syntax

```
EXECUTE statement_name [USING value1 [ ,value2, ... ] ]
```

`statement_name` is the name of the prepared statement. *`value1`* and *`value2`* are the values to be specified for the parameters in the statement.

EXECUTE examples

The following example runs the `my_select1` prepared statement, which contains no parameters.

```
EXECUTE my_select1
```

The following example runs the `my_select2` prepared statement, which contains a single parameter.

```
EXECUTE my_select2 USING 2012
```

The following example runs the `my_select3` prepared statement, which has two parameters.

```
EXECUTE my_select3 USING 346078, 12
```

The following example supplies a string value for a parameter in the prepared statement `my_insert`.

```
EXECUTE my_insert USING 'usa'
```

The following example supplies a numerical value for the `productid` parameter in the prepared statement `my_unload`.

```
EXECUTE my_unload USING 12
```

DEALLOCATE PREPARE

Removes the prepared statement with the specified name from the list of prepared statements in the current workgroup.

Syntax

```
DEALLOCATE PREPARE statement_name
```

`statement_name` is the name of the prepared statement to be removed.

Example

The following example removes the `my_select1` prepared statement from the current workgroup.

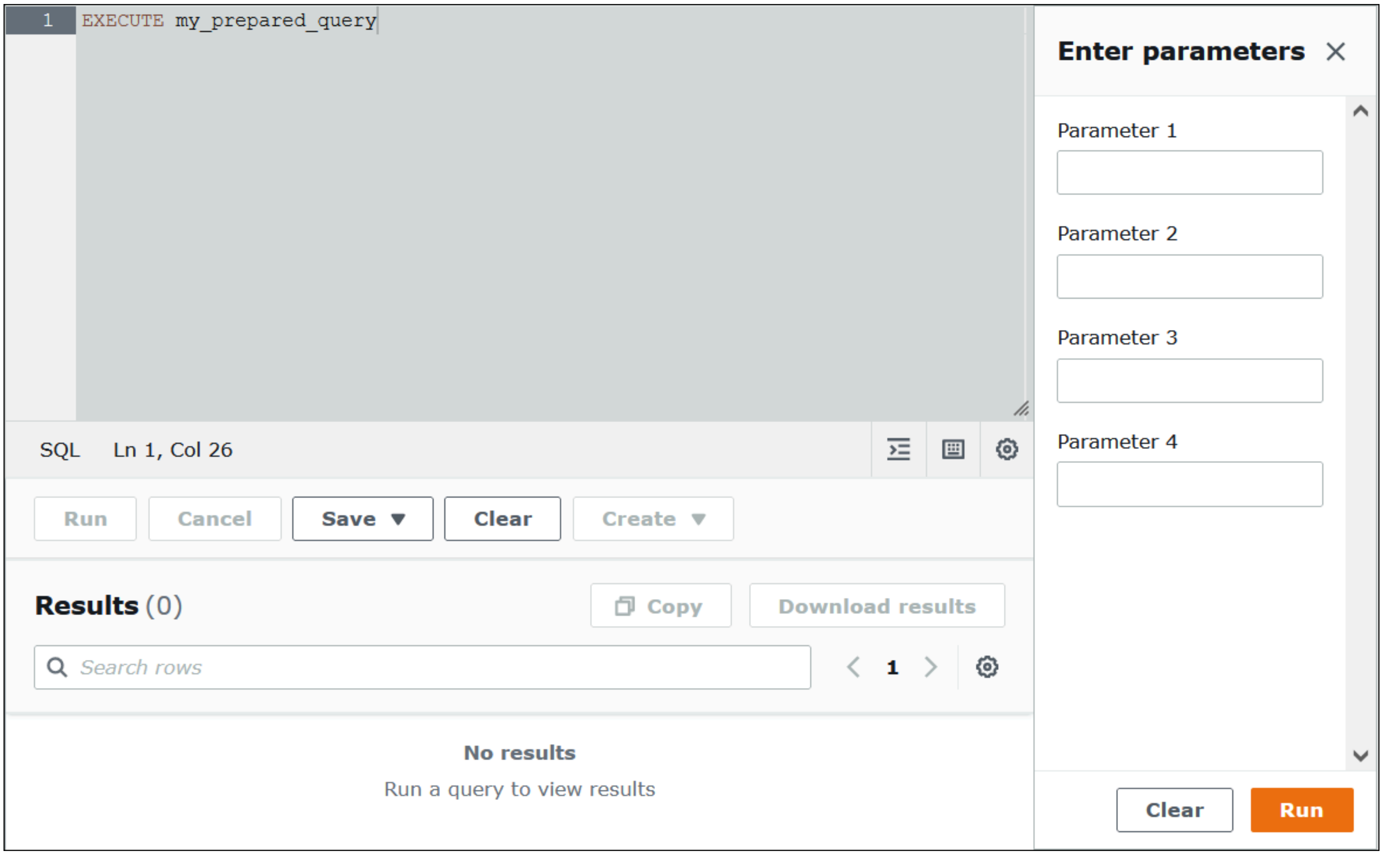
```
DEALLOCATE PREPARE my_select1
```

Run interactive prepared statements in the Athena console

If you run an existing prepared statement with the syntax `EXECUTE` *`prepared_statement`* in the query editor, Athena opens the **Enter parameters** dialog box so that you can enter the values that would normally go in the `USING` clause of the `EXECUTE ... USING` statement.

To run a prepared statement using the Enter parameters dialog box

1. In the query editor, instead of using the syntax `EXECUTE prepared_statement USING` *`value1`*, *`value2`* `...`, use the syntax `EXECUTE` *`prepared_statement`*.
2. Choose **Run**. The **Enter parameters** dialog box appears.



3. Enter the values in order in the **Execution parameters** dialog box. Because the original text of the query is not visible, you must remember the meaning of each positional parameter or have the prepared statement available for reference.
4. Choose **Run**.

Use the AWS CLI to create, execute, and list prepared statements

You can use the AWS CLI to create, execute, and list prepared statements.

Topics

- Create prepared statements using the AWS CLI
- Execute prepared statements using the AWS CLI
- List prepared statements using the AWS CLI

Create prepared statements using the AWS CLI

To use the AWS CLI to create a prepared statement, you can use one of the following `athena` commands:

- Use the `create-prepared-statement` command and provide a query statement that has execution parameters.
- Use the `start-query-execution` command and provide a query string that uses the `PREPARE` syntax.

Use create-prepared-statement

In a `create-prepared-statement` command, define the query text in the `query-statement` argument, as in the following example.

```
aws athena create-prepared-statement
--statement-name PreparedStatement1
--query-statement "SELECT * FROM table WHERE x = ?"
--work-group athena-engine-v2
```

Use start-query-execution and the PREPARE syntax

Use the `start-query-execution` command. Put the `PREPARE` statement in the `query-string` argument, as in the following example:

```
aws athena start-query-execution
--query-string "PREPARE PreparedStatement1 FROM SELECT * FROM table WHERE x = ?"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://amzn-s3-demo-bucket/..."}'
```

Execute prepared statements using the AWS CLI

To execute a prepared statement with the AWS CLI, you can supply values for the parameters by using one of the following methods:

- Use the `execution-parameters` argument.
- Use the `EXECUTE ... USING` SQL syntax in the `query-string` argument.

Use the execution-parameters argument

In this approach, you use the `start-query-execution` command and provide the name of an existing prepared statement in the `query-string` argument. Then, in the `execution-parameters` argument, you provide the values for the execution parameters. The following example shows this method.

```
aws athena start-query-execution
--query-string "Execute PreparedStatement1"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://amzn-s3-demo-bucket/..."
--execution-parameters "1" "2"
```

Use the EXECUTE ... USING SQL syntax

To run an existing prepared statement using the `EXECUTE ... USING` syntax, you use the `start-query-execution` command and place the both the name of the prepared statement and the parameter values in the `query-string` argument, as in the following example:

```
aws athena start-query-execution
--query-string "EXECUTE PreparedStatement1 USING 1"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://amzn-s3-demo-bucket/..."}'
```

List prepared statements using the AWS CLI

To list the prepared statements for a specific workgroup, you can use the Athena list-prepared-statements AWS CLI command or the ListPreparedStatements Athena API action. The `--work-group` parameter is required.

```
aws athena list-prepared-statements --work-group primary
```

Additional resources

See the following related posts in the AWS Big Data Blog.

- Improve reusability and security using Amazon Athena parameterized queries
- Use Amazon Athena parameterized queries to provide data as a service

Use the cost-based optimizer

You can use the cost-based optimizer (CBO) feature in Athena SQL to optimize your queries. You can optionally request that Athena gather table or column-level statistics for one of your tables in AWS Glue. If all of the tables in your query have statistics, Athena uses the statistics to create an execution plan that it determines to be the most performant. The query optimizer calculates alternative plans based on a statistical model and then selects the one that will likely be the fastest to run the query.

Statistics on AWS Glue tables are collected and stored in the AWS Glue Data Catalog and made available to Athena for improved query planning and execution. These statistics are column-level statistics such as number of distinct, number of null, max, and min values on file types such as Parquet, ORC, JSON, ION, CSV, and XML. Amazon Athena uses these statistics to optimize queries by applying the most restrictive filters as early as possible in query processing. This filtering limits memory usage and the number of records that must be read to deliver the query results.

In conjunction with CBO, Athena uses a feature called the rule-based optimizer (RBO). RBO mechanically applies rules that are expected to improve query performance. RBO is generally beneficial because its transformations aim to simplify the query plan. However, because RBO does not perform cost calculations or plan comparisons, more complicated queries make it difficult for RBO to create an optimal plan.

For this reason, Athena uses both RBO and CBO to optimize your queries. After Athena identifies opportunities to improve query execution, it creates an optimal plan. For information about execution plan details, see View execution plans for SQL queries. For a detailed discussion of how CBO works, see Speed up queries with the cost-based optimizer in Amazon Athena in the the AWS Big Data Blog.

To generate statistics for AWS Glue Catalog tables, you can use the Athena console, the AWS Glue Console, or AWS Glue APIs. Because Athena is integrated with AWS Glue Catalog, you automatically get the corresponding query performance improvements when you run queries from Amazon Athena.

Considerations and limitations

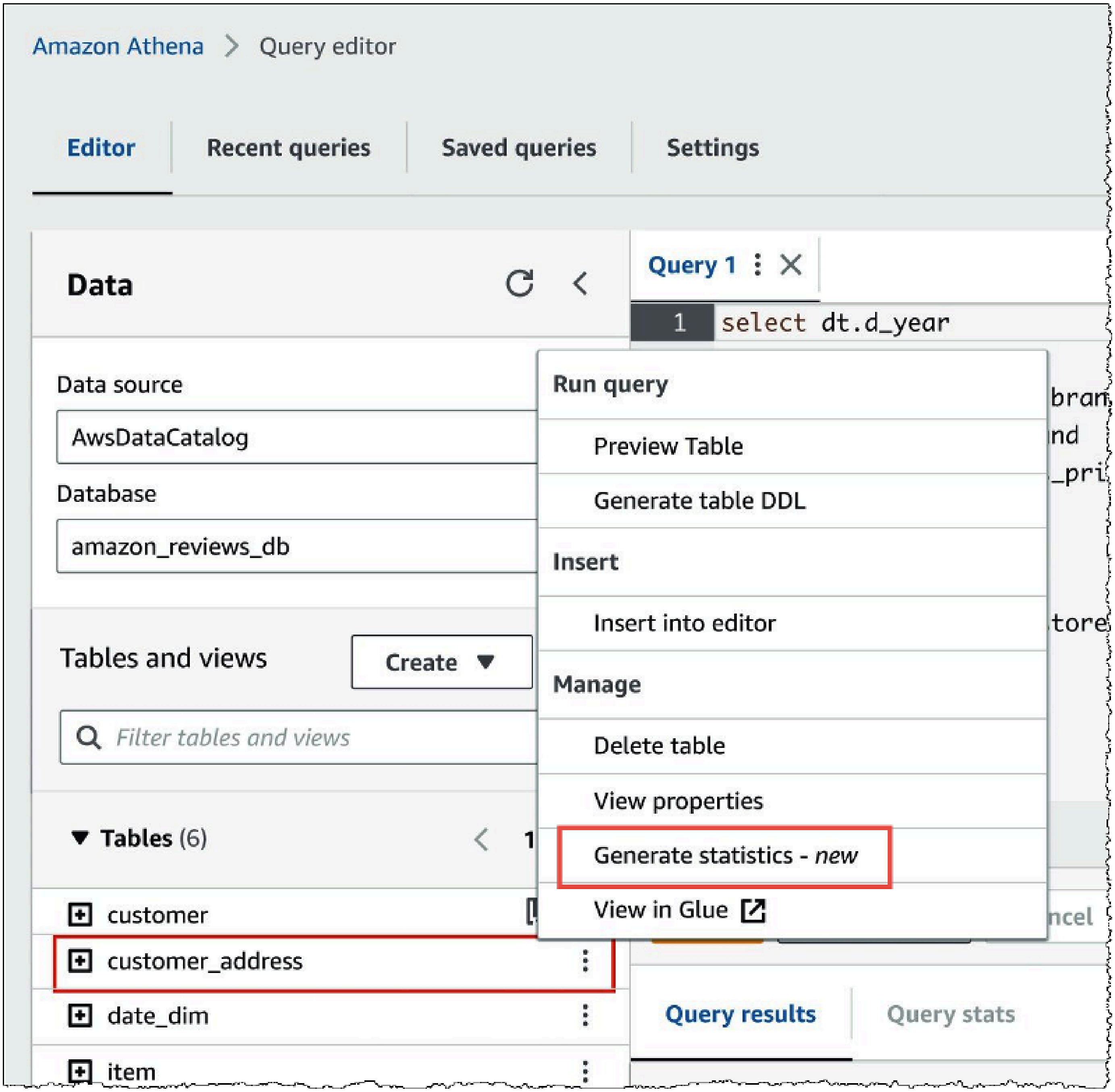
- **Table types –** Currently, the CBO feature in Athena supports only Hive tables that are in the AWS Glue Data Catalog.
- **Athena for Spark –** The CBO feature is not available in Athena for Spark.
- **Pricing –** For pricing information, visit the AWS Glue pricing page.

Generate table statistics using the Athena console

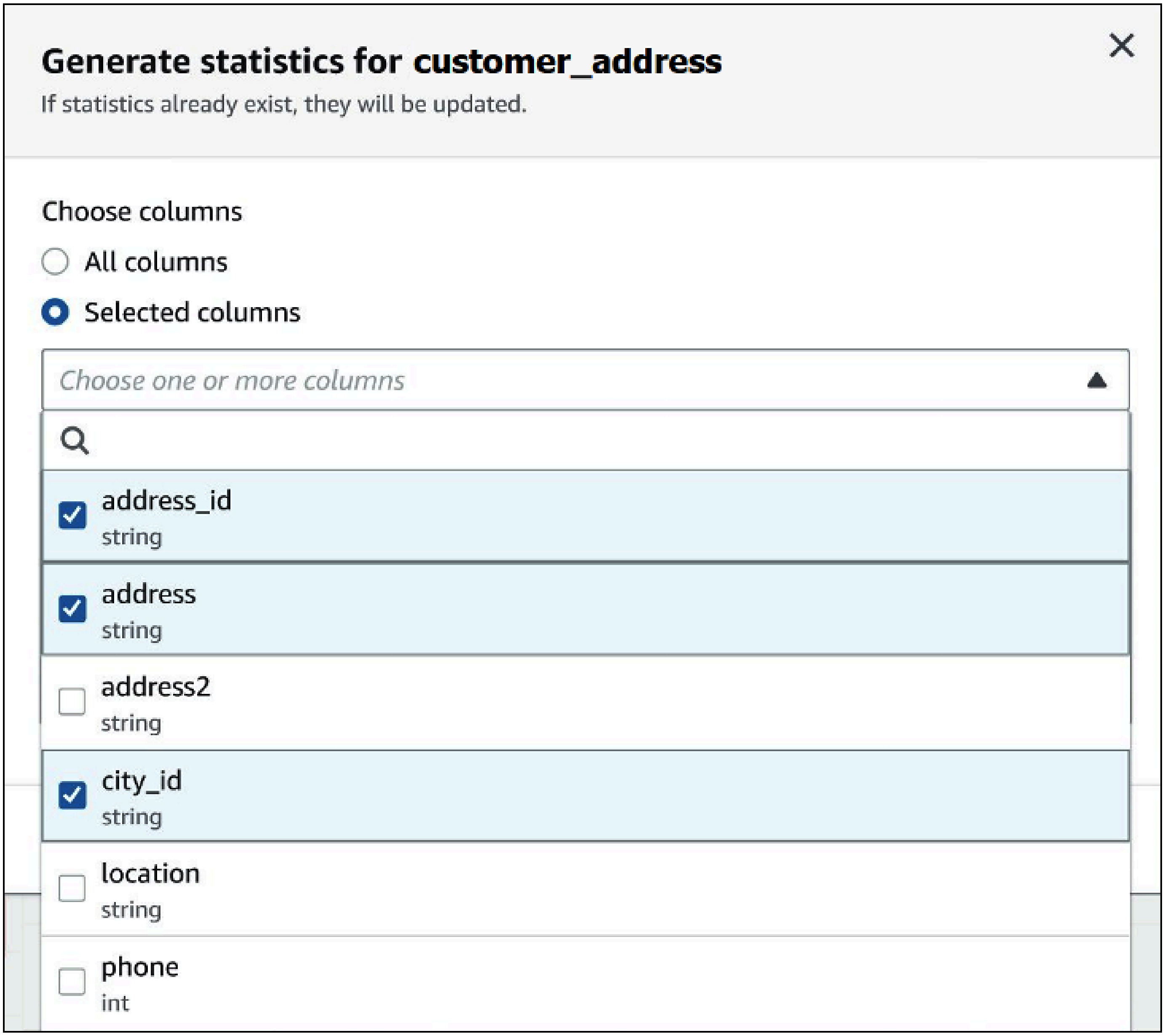
This section describes how to use the Athena console to generate table or column-level statistics for a table in AWS Glue. For information on using AWS Glue to generate table statistics, see Working with column statistics in the *AWS Glue Developer Guide*.

To generate statistics for a table using the Athena console

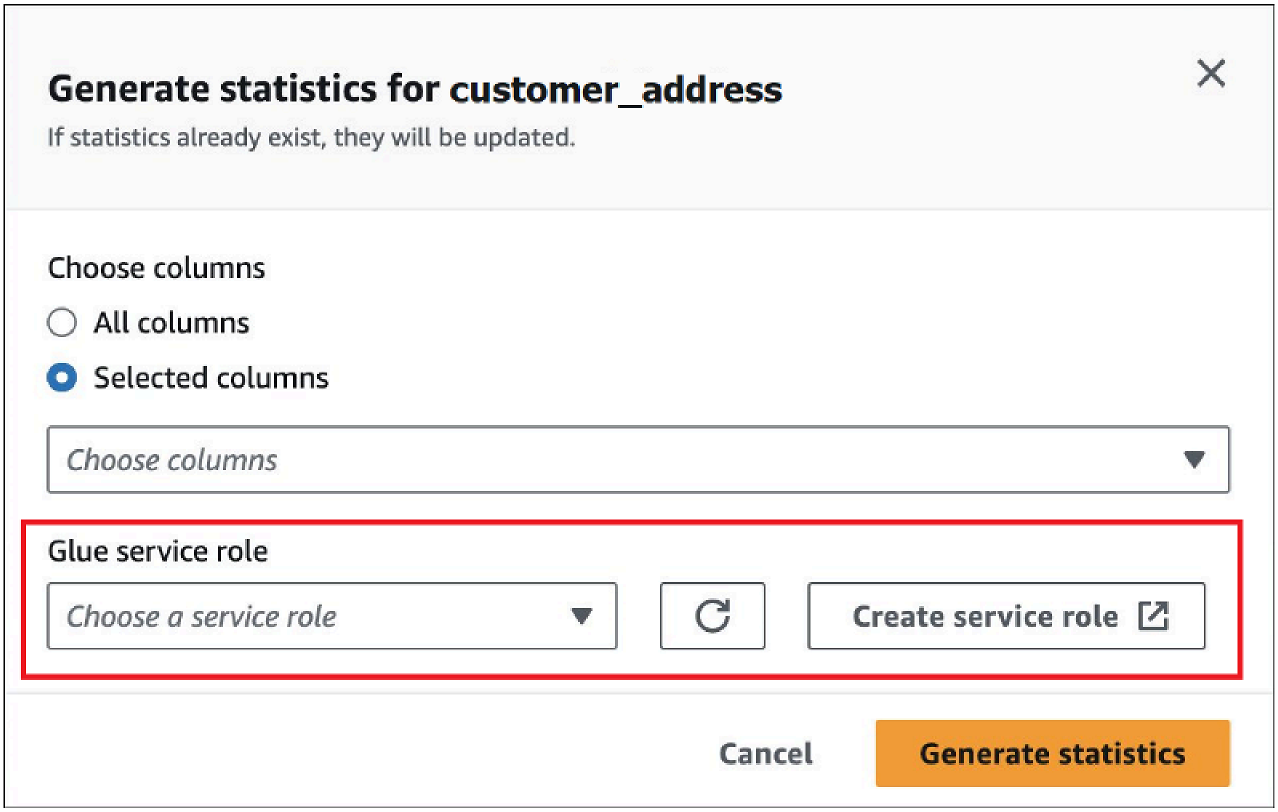
1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena query editor **Tables** list, choose the three vertical dots for the table that you want, and then choose **Generate statistics**.



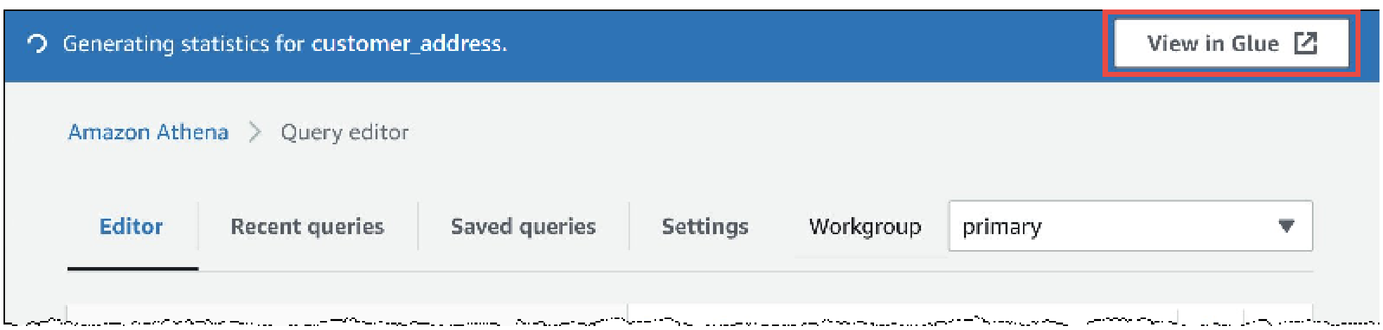
3. In the **Generate statistics** dialog box, choose **All columns** to generate statistics for all columns in the table, or choose **Selected columns** to select specific columns. **All columns** is the default setting.



4. For **AWS Glue service role**, create or select an existing service role to give permission to AWS Glue to generate statistics. The AWS Glue service role also requires [`S3:GetObject`] permissions to the Amazon S3 bucket that contains the table's data.



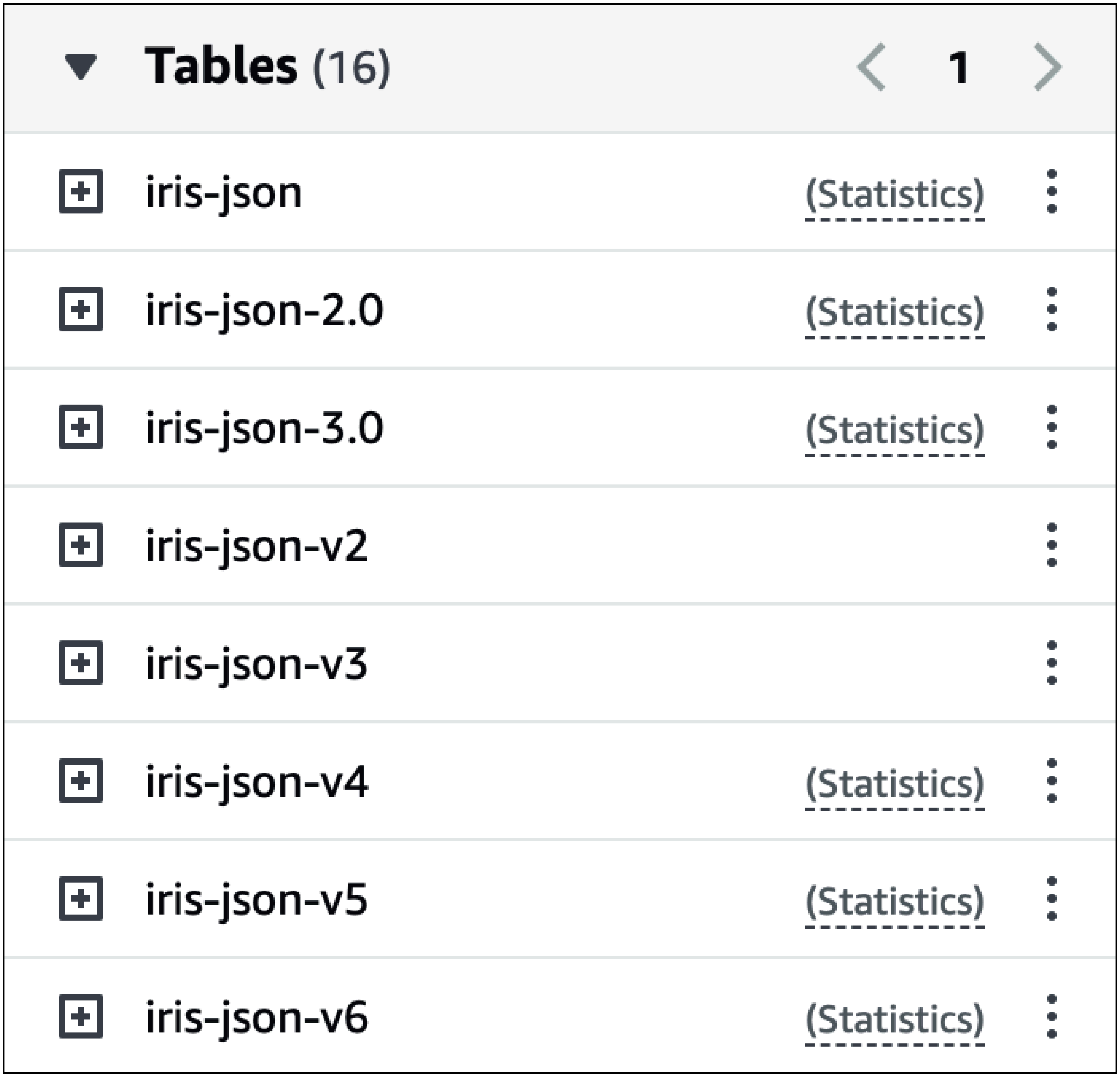
5. Choose **Generate statistics**. A **Generating statistics for *table_name*** notification banner displays the task status.



6. To view details in the AWS Glue console, choose **View in Glue**.

   For information about viewing statistics in the AWS Glue console, see [Viewing column statistics](#) in the *AWS Glue Developer Guide*.

7. After statistics have been generated, the tables and columns that have statistics show the word **Statistics** in parentheses, as in the following image.



Now when you run your queries, Athena will perform cost-based optimization on the tables and columns for which statistics were generated.

Additional resources

For additional information, see the following resource.

Query S3 Express One Zone data

The Amazon S3 Express One Zone storage class is a highly performant Amazon S3 storage class that provides single-digit millisecond response times. As such, it is useful for applications that frequently access data with hundreds of thousands of requests per second.

S3 Express One Zone replicates and stores data within the same Availability Zone to optimize for speed and cost. This differs from Amazon S3 Regional storage classes, which automatically replicate data across a minimum of three AWS Availability Zones within an AWS Region.

For more information, see [What is S3 Express One Zone?](#) in the *Amazon S3 User Guide*.

Prerequisites

Confirm that the following conditions are met before you begin:

- **Athena engine version 3 –** To use S3 Express One Zone with Athena SQL, your workgroup must be configured to use Athena engine version 3.
- **S3 Express One Zone permissions –** When S3 Express One Zone calls an action like `GET`, `LIST`, or `PUT` on an Amazon S3 object, the storage class calls `CreateSession` on your behalf. For this reason, your IAM policy must allow the `s3express:CreateSession` action, which allows Athena to invoke the corresponding API operation.

Considerations and limitations

When you query S3 Express One Zone with Athena, consider the following points.

- S3 Express One Zone buckets supports `SSE_S3` and `SSE-KMS` encryption. Athena query results are written using `SSE_S3` encryption regardless of the option that you choose in workgroup settings to encrypt query results. This limitation includes all scenarios in which Athena writes data to S3 Express One Zone buckets, including `CREATE TABLE AS` (CTAS) and `INSERT INTO` statements.
- The AWS Glue crawler is not supported for creating tables on S3 Express One Zone data.
- The `MSCK REPAIR TABLE` statement is not supported. As a workaround, use [ALTER TABLE ADD PARTITION](#).
- No table modifying DDL statements for Apache Iceberg (that is, no `ALTER TABLE` statements) are supported for S3 Express One Zone.

- Lake Formation is not supported with S3 Express One Zone buckets.
- The following file and table formats are unsupported or have limited support. If formats aren't listed, but are supported for Athena (such as Parquet, ORC, and JSON), then they're also supported for use with S3 Express One Zone storage.

File or table format	Limitation
Apache Avro	Not supported
CloudTrail logs	Not supported
Apache Hudi	Not supported
Amazon Ion	Not supported
Logstash logs	Not supported
Apache WebServer logs	Not supported
Delta Lake	DDL not supported. For information about creating a Delta Lake table using a dummy schema, see Synchronize Delta Lake metadata. `SELECT` queries against the table are supported.

Get started

Querying S3 Express One Zone data with Athena is straightforward. To get started, use the following procedure.

To use Athena SQL to query S3 Express One Zone data

1. Transition your data to S3 Express One Zone storage. For more information, see Setting the storage class of an object in the *Amazon S3 User Guide*.
2. Use a CREATE TABLE statement in Athena to catalog your data in AWS Glue Data Catalog. For information about creating tables in Athena, see Create tables in Athena and the CREATE TABLE statement.
3. (Optional) Configure the query result location of your Athena workgroup to use an Amazon S3 *directory bucket*. Amazon S3 directory buckets are more performant that general buckets and

are designed for workloads or performance-critical applications that require consistent single-digit millisecond latency. For more information, see Directory buckets overview in the *Amazon S3 User Guide*.

Query restored Amazon S3 Glacier objects

You can use Athena to query restored objects from the S3 Glacier Flexible Retrieval (formerly Glacier) and S3 Glacier Deep Archive Amazon S3 storage classes. You must enable this capability on a per-table basis. If you do not enable the feature on a table before you run a query, Athena skips all of the table's S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive objects during query execution.

Considerations and Limitations

- Querying restored Amazon S3 Glacier objects is supported only on Athena engine version 3.
- The feature is supported only for Apache Hive tables.
- You must restore your objects before you query your data; Athena does not restore objects for you.

Configure a table to use restored objects

To configure your Athena table to include restored objects in your queries, you must set its `read_restored_glacier_objects` table property to `true`. To do this, you can use the Athena query editor or the AWS Glue console. You can also use the AWS Glue CLI, the AWS Glue API, or the AWS Glue SDK.

Use the Athena query editor

In Athena, you can use the ALTER TABLE SET TBLPROPERTIES command to set the table property, as in the following example.

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'true')
```

Use the AWS Glue console

In the AWS Glue console, perform the following steps to add the `read_restored_glacier_objects` table property.

To configure table properties in the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. Do one of the following:
   - Choose **Go to the Data Catalog**.
   - In the navigation pane, choose **Data Catalog tables**.
3. On the **Tables** page, in the list of tables, choose the link for the table that you want to edit.
4. Choose **Actions**, **Edit table**.
5. On the **Edit table** page, in the **Table properties** section, add the following key-value pair.
   - For **Key**, add `read_restored_glacier_objects`.
   - For **Value**, enter `true`.
6. Choose **Save**.

Use the AWS CLI

In the AWS CLI, you can use the AWS Glue [update-table](https://docs.aws.amazon.com/cli/latest/reference/glue/update-table.html) command and its `--table-input` argument to redefine the table and in so doing add the `read_restored_glacier_objects` property. In the `--table-input` argument, use the `Parameters` structure to specify the `read_restored_glacier_objects` property and the value of `true`. Note that the argument for `--table-input` must not have spaces and must use backslashes to escape the double quotes. In the following example, replace *`my_database`* and *`my_table`* with the name of your database and table.

```
aws glue update-table \
   --database-name my_database \
   --table-input={\"Name\":\"my_table\",\"Parameters\":{\"read_restored_glacier_objects
\":\"true\"}}
```

> **Important**
>
> The AWS Glue `update-table` command works in overwrite mode, which means that it replaces the existing table definition with the new definition specified by the `table-input` parameter. For this reason, be sure to also specify all of the fields

that you want to be in your table in the `table-input` parameter when you add the `read_restored_glacier_objects` property.

Handle schema updates

This section provides guidance on handling schema updates for various data formats. Athena is a schema-on-read query engine. This means that when you create a table in Athena, it applies schemas when reading the data. It does not change or rewrite the underlying data.

If you anticipate changes in table schemas, consider creating them in a data format that is suitable for your needs. Your goals are to reuse existing Athena queries against evolving schemas, and avoid schema mismatch errors when querying tables with partitions.

To achieve these goals, choose a table's data format based on the table in the following topic.

Topics

Supported schema update operations by data format

The following table summarizes data storage formats and their supported schema manipulations. Use this table to help you choose the format that will enable you to continue using Athena queries even as your schemas change over time.

In this table, observe that Parquet and ORC are columnar formats with different default column access methods. By default, Parquet will access columns by name and ORC by index (ordinal value). Therefore, Athena provides a SerDe property defined when creating a table to toggle the default column access method which enables greater flexibility with schema evolution.

For Parquet, the `parquet.column.index.access` property may be set to `true`, which sets the column access method to use the column's ordinal number. Setting this property to `false` will change the column access method to use column name. Similarly, for ORC use the `orc.column.index.access` property to control the column access method. For more information, see Understand index access for Apache ORC and Apache Parquet.

CSV and TSV allow you to do all schema manipulations except reordering of columns, or adding columns at the beginning of the table. For example, if your schema evolution requires only renaming columns but not removing them, you can choose to create your tables in CSV or TSV. If you require removing columns, do not use CSV or TSV, and instead use any of the other supported formats, preferably, a columnar format, such as Parquet or ORC.

Schema updates and data formats in Athena

Expected type of schema update	Summary	CSV (with and without headers and TSV	JSO	AVI	PARQUE Read by name (default)	PARQUE Read by index	ORC: Read by index (default)	ORC: Read by name
Rename columns	Store your data in CSV and TSV, or in ORC and Parquet if they are read by index.	Y	N	N	N	Y	Y	N
Add columns at the beginning or in the middle of the table	Store your data in JSON, AVRO, or in Parquet and ORC if they are read by name. Do not use CSV and TSV.	N	Y	Y	Y	N	N	Y
Add columns at the end of the table	Store your data in CSV or TSV, JSON, AVRO, ORC, or Parquet.	Y	Y	Y	Y	Y	Y	Y
Remove columns	Store your data in JSON, AVRO, or Parquet and ORC, if they are read by	N	Y	Y	Y	N	N	Y

Expected type of schema update	Summary	CSV (with and without headers and TSV	JSO	AVI	PARQUE Read by name (default)	PARQUE Read by index	ORC: Read by index (default)	ORC: Read by name
	name. Do not use CSV and TSV.							
[Reorder columns]()	Store your data in AVRO, JSON or ORC and Parquet if they are read by name.	N	Y	Y	Y	N	N	Y
[Change a column's data type]()	Store your data in any format, but test your query in Athena to make sure the data types are compatible. For Parquet and ORC, changing a data type works only for partitioned tables.	Y	Y	Y	Y	Y	Y	Y

Understand index access for Apache ORC and Apache Parquet

PARQUET and ORC are columnar data storage formats that can be read by index, or by name. Storing your data in either of these formats lets you perform all operations on schemas and run Athena queries without schema mismatch errors.

- Athena *reads ORC by index by default*, as defined in `SERDEPROPERTIES ( 'orc.column.index.access'='true' )`. For more information, see ORC: Read by index.
- Athena reads *Parquet by name by default*, as defined in `SERDEPROPERTIES ( 'parquet.column.index.access'='false' )`. For more information, see Parquet: Read by name.

Since these are defaults, specifying these SerDe properties in your `CREATE TABLE` queries is optional, they are used implicitly. When used, they allow you to run some schema update operations while preventing other such operations. To enable those operations, run another `CREATE TABLE` query and change the SerDe settings.

> **Note**
>
> The SerDe properties are *not* automatically propagated to each partition. Use `ALTER TABLE ADD PARTITION` statements to set the SerDe properties for each partition. To automate this process, write a script that runs `ALTER TABLE ADD PARTITION` statements.

The following sections describe these cases in detail.

ORC: Read by index

A table in *ORC is read by index*, by default. This is defined by the following syntax:

```
WITH SERDEPROPERTIES (
  'orc.column.index.access'='true')
```

Reading by index allows you to rename columns. But then you lose the ability to remove columns or add them in the middle of the table.

To make ORC read by name, which will allow you to add columns in the middle of the table or remove columns in ORC, set the SerDe property `orc.column.index.access` to `false` in the `CREATE TABLE` statement. In this configuration, you will lose the ability to rename columns.

> **Note**
>
> In Athena engine version 2, when ORC tables are set to read by name, Athena requires that all column names in the ORC files be in lower case. Because Apache Spark does not

lowercase field names when it generates ORC files, Athena might not be able to read the data so generated. The workaround is to rename the columns to be in lower case, or use Athena engine version 3.

The following example illustrates how to change the ORC to make it read by name:

```
CREATE EXTERNAL TABLE orders_orc_read_by_name (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_orderdate` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'orc.column.index.access'='false')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/orders_orc/';
```

Parquet: Read by name

A table in *Parquet is read by name*, by default. This is defined by the following syntax:

```
WITH SERDEPROPERTIES (
  'parquet.column.index.access'='false')
```

Reading by name allows you to add columns in the middle of the table and remove columns. But then you lose the ability to rename columns.

To make Parquet read by index, which will allow you to rename columns, you must create a table with `parquet.column.index.access` SerDe property set to `true`.

Make schema updates

This topic describes some of the changes that you can make to the schema in `CREATE TABLE` statements without actually altering your data. To update a schema, you can in some cases use an `ALTER TABLE` command, but in other cases you do not actually modify an existing table. Instead, you create a table with a new name that modifies the schema that you used in your original `CREATE TABLE` statement.

Depending on how you expect your schemas to evolve, to continue using Athena queries, choose a compatible data format.

Consider an application that reads orders information from an `orders` table that exists in two formats: CSV and Parquet.

The following example creates a table in Parquet:

```
CREATE EXTERNAL TABLE orders_parquet (
   `orderkey` int,
   `orderstatus` string,
   `totalprice` double,
   `orderdate` string,
   `orderpriority` string,
   `clerk` string,
   `shippriority` int
) STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/orders_ parquet/';
```

The following example creates the same table in CSV:

```
CREATE EXTERNAL TABLE orders_csv (
   `orderkey` int,
   `orderstatus` string,
   `totalprice` double,
   `orderdate` string,
   `orderpriority` string,
   `clerk` string,
   `shippriority` int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://amzn-s3-demo-bucket/orders_csv/';
```

The following topics show how updates to these tables affect Athena queries.

Topics

Add columns at the beginning or in the middle of the table

Adding columns is one of the most frequent schema changes. For example, you may add a new column to enrich the table with new data. Or, you may add a new column if the source for an existing column has changed, and keep the previous version of this column, to adjust applications that depend on them.

To add columns at the beginning or in the middle of the table, and continue running queries against existing tables, use AVRO, JSON, and Parquet and ORC if their SerDe property is set to read by name. For information, see Understand index access for Apache ORC and Apache Parquet.

Do not add columns at the beginning or in the middle of the table in CSV and TSV, as these formats depend on ordering. Adding a column in such cases will lead to schema mismatch errors when the schema of partitions changes.

The following example creates a new table that adds an `o_comment` column in the middle of a table based on JSON data.

```
CREATE EXTERNAL TABLE orders_json_column_addition (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_comment` string,
   `o_totalprice` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket/orders_json/';
```

Add columns at the end of the table

If you create tables in any of the formats that Athena supports, such as Parquet, ORC, Avro, JSON, CSV, and TSV, you can use the `ALTER TABLE ADD COLUMNS` statement to add columns after existing columns but before partition columns.

The following example adds a `comment` column at the end of the `orders_parquet` table before any partition columns:

```
ALTER TABLE orders_parquet ADD COLUMNS (comment string)
```

> **Note**
>
> To see a new table column in the Athena Query Editor after you run `ALTER TABLE ADD COLUMNS`, manually refresh the table list in the editor, and then expand the table again.

Remove columns

You may need to remove columns from tables if they no longer contain data, or to restrict access to the data in them.

- You can remove columns from tables in JSON, Avro, and in Parquet and ORC if they are read by name. For information, see Understand index access for Apache ORC and Apache Parquet.
- We do not recommend removing columns from tables in CSV and TSV if you want to retain the tables you have already created in Athena. Removing a column breaks the schema and requires that you recreate the table without the removed column.

In this example, remove a column `` `totalprice` `` from a table in Parquet and run a query. In Athena, Parquet is read by name by default, this is why we omit the SERDEPROPERTIES configuration that specifies reading by name. Notice that the following query succeeds, even though you changed the schema:

```
CREATE EXTERNAL TABLE orders_parquet_column_removed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_orderdate` string,
   `o_orderpriority` string,
```

```
    `o_clerk` string,
    `o_shippriority` int,
    `o_comment` string
)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/orders_parquet/';
```

Rename columns

You may want to rename columns in your tables to correct spelling, make column names more descriptive, or to reuse an existing column to avoid column reordering.

You can rename columns if you store your data in CSV and TSV, or in Parquet and ORC that are configured to read by index. For information, see [Understand index access for Apache ORC and Apache Parquet](#).

Athena reads data in CSV and TSV in the order of the columns in the schema and returns them in the same order. It does not use column names for mapping data to a column, which is why you can rename columns in CSV or TSV without breaking Athena queries.

One strategy for renaming columns is to create a new table based on the same underlying data, but using new column names. The following example creates a new `orders_parquet` table called `orders_parquet_column_renamed`. The example changes the column `` `o_totalprice` `` name to `` `o_total_price` `` and then runs a query in Athena:

```
CREATE EXTERNAL TABLE orders_parquet_column_renamed (
    `o_orderkey` int,
    `o_custkey` int,
    `o_orderstatus` string,
    `o_total_price` double,
    `o_orderdate` string,
    `o_orderpriority` string,
    `o_clerk` string,
    `o_shippriority` int,
    `o_comment` string
)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/orders_parquet/';
```

In the Parquet table case, the following query runs, but the renamed column does not show data because the column was being accessed by name (a default in Parquet) rather than by index:

```
SELECT *
FROM orders_parquet_column_renamed;
```

A query with a table in CSV looks similar:

```
CREATE EXTERNAL TABLE orders_csv_column_renamed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_total_price` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://amzn-s3-demo-bucket/orders_csv/';
```

In the CSV table case, the following query runs and the data displays in all columns, including the one that was renamed:

```
SELECT *
FROM orders_csv_column_renamed;
```

Reorder columns

You can reorder columns only for tables with data in formats that read by name, such as JSON or Parquet, which reads by name by default. You can also make ORC read by name, if needed. For information, see Understand index access for Apache ORC and Apache Parquet.

The following example creates a new table with the columns in a different order:

```
CREATE EXTERNAL TABLE orders_parquet_columns_reordered (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
```

```
    `o_orderdate` string
)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/orders_parquet/';
```

Change a column data type

You might want to use a different column type when the existing type can no longer hold the amount of information required. For example, an ID column's values might exceed the size of the `INT` data type and require the use of the `BIGINT` data type.

Considerations

When planning to use a different data type for a column, consider the following points:

- In most cases, you cannot change the data type of a column directly. Instead, you re-create the Athena table and define the column with the new data type.
- Only certain data types can be read as other data types. See the table in this section for data types that can be so treated.
- For data in Parquet and ORC, you cannot use a different data type for a column if the table is not partitioned.
- For partitioned tables in Parquet and ORC, a partition's column type can be different from another partition's column type, and Athena will `CAST` to the desired type, if possible. For information, see Avoid schema mismatch errors for tables with partitions.
- For tables created using the LazySimpleSerDe only, it is possible to use the `ALTER TABLE REPLACE COLUMNS` statement to replace existing columns with a different data type, but all existing columns that you want to keep must also be redefined in the statement, or they will be dropped. For more information, see ALTER TABLE REPLACE COLUMNS.
- For Apache Iceberg tables only, you can use the ALTER TABLE CHANGE COLUMN statement to change the data type of a column. `ALTER TABLE REPLACE COLUMNS` is not supported for Iceberg tables. For more information, see Evolve Iceberg table schema.

> **Important**
>
> We strongly suggest that you test and verify your queries before performing data type translations. If Athena cannot use the target data type, the `CREATE TABLE` query may fail.

Use compatible data types

Whenever possible, use compatible data types. The following table lists data types that can be treated as other data types:

Original data type	Available target data types
STRING	BYTE, TINYINT, SMALLINT, INT, BIGINT
BYTE	TINYINT, SMALLINT, INT, BIGINT
TINYINT	SMALLINT, INT, BIGINT
SMALLINT	INT, BIGINT
INT	BIGINT
FLOAT	DOUBLE

The following example uses the CREATE TABLE statement for the original orders_json table to create a new table called orders_json_bigint. The new table uses BIGINT instead of INT as the data type for the `` `o_shippriority` `` column.

```
CREATE EXTERNAL TABLE orders_json_bigint (
    `o_orderkey` int,
    `o_custkey` int,
    `o_orderstatus` string,
    `o_totalprice` double,
    `o_orderdate` string,
    `o_orderpriority` string,
    `o_clerk` string,
    `o_shippriority` BIGINT
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket/orders_json';
```

The following query runs successfully, similar to the original SELECT query, before the data type change:

```
Select * from orders_json
```

```
LIMIT 10;
```

Update tables with partitions

In Athena, a table and its partitions must use the same data formats but their schemas may differ. When you create a new partition, that partition usually inherits the schema of the table. Over time, the schemas may start to differ. Reasons include:

- If your table's schema changes, the schemas for partitions are not updated to remain in sync with the table's schema.
- The AWS Glue Crawler allows you to discover data in partitions with different schemas. This means that if you create a table in Athena with AWS Glue, after the crawler finishes processing, the schemas for the table and its partitions may be different.
- If you add partitions directly using an AWS API.

Athena processes tables with partitions successfully if they meet the following constraints. If these constraints are not met, Athena issues a HIVE_PARTITION_SCHEMA_MISMATCH error.

- Each partition's schema is compatible with the table's schema.
- The table's data format allows the type of update you want to perform: add, delete, reorder columns, or change a column's data type.

  For example, for CSV and TSV formats, you can rename columns, add new columns at the end of the table, and change a column's data type if the types are compatible, but you cannot remove columns. For other formats, you can add or remove columns, or change a column's data type to another if the types are compatible. For information, see [Summary: Updates and Data Formats in Athena](#).

Avoid schema mismatch errors for tables with partitions

At the beginning of query execution, Athena verifies the table's schema by checking that each column data type is compatible between the table and the partition.

- For Parquet and ORC data storage types, Athena relies on the column names and uses them for its column name-based schema verification. This eliminates `HIVE_PARTITION_SCHEMA_MISMATCH` errors for tables with partitions in Parquet and ORC. (This is true for ORC if the SerDe property is set to access the index by name: `orc.column.index.access=FALSE`. Parquet reads the index by name by default).

- For CSV, JSON, and Avro, Athena uses an index-based schema verification. This means that if you encounter a schema mismatch error, you should drop the partition that is causing a schema mismatch and recreate it, so that Athena can query it without failing.

Athena compares the table's schema to the partition schemas. If you create a table in CSV, JSON, and AVRO in Athena with AWS Glue Crawler, after the Crawler finishes processing, the schemas for the table and its partitions may be different. If there is a mismatch between the table's schema and the partition schemas, your queries fail in Athena due to the schema verification error similar to this: 'crawler_test.click_avro' is declared as type 'string', but partition 'partition_0=2017-01-17' declared column 'col68' as type 'double'."

A typical workaround for such errors is to drop the partition that is causing the error and recreate it. For more information, see ALTER TABLE DROP PARTITION and ALTER TABLE ADD PARTITION.

Query arrays

Amazon Athena lets you create arrays, concatenate them, convert them to different data types, and then filter, flatten, and sort them.

Topics

Create arrays

To build an array literal in Athena, use the ARRAY keyword, followed by brackets [ ], and include the array elements separated by commas.

Examples

This query creates one array with four elements.

```
SELECT ARRAY [1,2,3,4] AS items
```

It returns:

```
+-----------+
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

This query creates two arrays.

```
SELECT ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

It returns:

```
+--------------------+
| items              |
+--------------------+
| [[1, 2], [3, 4]]   |
+--------------------+
```

To create an array from selected columns of compatible types, use a query, as in this example:

```
WITH
dataset AS (
  SELECT 1 AS x, 2 AS y, 3 AS z
)
SELECT ARRAY [x,y,z] AS items FROM dataset
```

This query returns:

```
+-----------+
| items     |
+-----------+
| [1,2,3]   |
+-----------+
```

In the following example, two arrays are selected and returned as a welcome message.

```
WITH
dataset AS (
  SELECT
    ARRAY ['hello', 'amazon', 'athena'] AS words,
    ARRAY ['hi', 'alexa'] AS alexa
)
SELECT ARRAY[words, alexa] AS welcome_msg
FROM dataset
```

This query returns:

```
+----------------------------------------+
| welcome_msg                            |
+----------------------------------------+
| [[hello, amazon, athena], [hi, alexa]] |
+----------------------------------------+
```

To create an array of key-value pairs, use the `MAP` operator that takes an array of keys followed by an array of values, as in this example:

```
SELECT ARRAY[
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
] AS people
```

This query returns:

```
+-----------------------------------------------------------------------------------------------------
+
| people
                |
```

```
+-----------------------------------------------------------------------------------------
+
| [{last=Smith, first=Bob, age=40}, {last=Doe, first=Jane, age=30}, {last=Smith,
 first=Billy, age=8}] |
+-----------------------------------------------------------------------------------------
+
```

Concatenate strings and arrays

Concatenating strings and concatenating arrays use similar techniques.

Concatenate strings

To concatenate two strings, you can use the double pipe **||** operator, as in the following example.

```
SELECT 'This' || ' is' || ' a' || ' test.' AS Concatenated_String
```

This query returns:

#	Concatenated_String
1	This is a test.

You can use the `concat()` function to achieve the same result.

```
SELECT concat('This', ' is', ' a', ' test.') AS Concatenated_String
```

This query returns:

#	Concatenated_String
1	This is a test.

You can use the `concat_ws()` function to concatenate strings with the separator specified in the first argument.

```
SELECT concat_ws(' ', 'This', 'is', 'a', 'test.') as Concatenated_String
```

This query returns:

#	Concatenated_String
1	This is a test.

To concatenate two columns of the string data type using a dot, reference the two columns using double quotes, and enclose the dot in single quotes as a hard-coded string. If a column is not of the string data type, you can use `CAST("column_name" as VARCHAR)` to cast the column first.

```
SELECT "col1" || '.' || "col2" as Concatenated_String
FROM my_table
```

This query returns:

#	Concatenated_String
1	*col1_string_value* .*col2_string_value*

Concatenate arrays

You can use the same techniques to concatenate arrays.

To concatenate multiple arrays, use the double pipe **||** operator.

```
SELECT ARRAY [4,5] || ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

This query returns:

#	items
1	[[4, 5], [1, 2], [3, 4]]

To combine multiple arrays into a single array, use the double pipe operator or the `concat()` function.

```
WITH
dataset AS (
  SELECT
    ARRAY ['Hello', 'Amazon', 'Athena'] AS words,
    ARRAY ['Hi', 'Alexa'] AS alexa
)
SELECT concat(words, alexa) AS welcome_msg
FROM dataset
```

This query returns:

#	welcome_msg
1	[Hello, Amazon, Athena, Hi, Alexa]

For more information about `concat()` other string functions, see String functions and operators in the Trino documentation.

Convert array data types

To convert data in arrays to supported data types, use the `CAST` operator, as `CAST(value AS type)`. Athena supports all of the native Presto data types.

```
SELECT
   ARRAY [CAST(4 AS VARCHAR), CAST(5 AS VARCHAR)]
AS items
```

This query returns:

```
+-------+
| items |
+-------+
| [4,5] |
+-------+
```

Create two arrays with key-value pair elements, convert them to JSON, and concatenate, as in this example:

```
SELECT
```

```
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items
```

This query returns:

```
+--------------------------------------------------+
| items                                            |
+--------------------------------------------------+
| [{"a1":1,"a2":2,"a3":3}, {"b1":4,"b2":5,"b3":6}] |
+--------------------------------------------------+
```

Find array lengths

The `cardinality` function returns the length of an array, as in this example:

```
SELECT cardinality(ARRAY[1,2,3,4]) AS item_count
```

This query returns:

```
+------------+
| item_count |
+------------+
| 4          |
+------------+
```

Access array elements

To access array elements, use the `[]` operator, with 1 specifying the first element, 2 specifying the second element, and so on, as in this example:

```
WITH dataset AS (
SELECT
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items )
SELECT items[1] AS item FROM dataset
```

This query returns:

```
+------------------------+
| item                   |
+------------------------+
| {"a1":1,"a2":2,"a3":3} |
+------------------------+
```

To access the elements of an array at a given position (known as the index position), use the `element_at()` function and specify the array name and the index position:

- If the index is greater than 0, `element_at()` returns the element that you specify, counting from the beginning to the end of the array. It behaves as the `[]` operator.
- If the index is less than 0, `element_at()` returns the element counting from the end to the beginning of the array.

The following query creates an array `words`, and selects the first element `hello` from it as the `first_word`, the second element `amazon` (counting from the end of the array) as the `middle_word`, and the third element `athena`, as the `last_word`.

```
WITH dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT
  element_at(words, 1) AS first_word,
  element_at(words, -2) AS middle_word,
  element_at(words, cardinality(words)) AS last_word
FROM dataset
```

This query returns:

```
+----------------------------------------+
| first_word  | middle_word | last_word  |
+----------------------------------------+
| hello       | amazon      | athena     |
+----------------------------------------+
```

Flatten nested arrays

When working with nested arrays, you often need to expand nested array elements into a single array, or expand the array into multiple rows.

Use the flatten function

To flatten a nested array's elements into a single array of values, use the `flatten` function. This query returns a row for each element in the array.

```
SELECT flatten(ARRAY[ ARRAY[1,2], ARRAY[3,4] ]) AS items
```

This query returns:

```
+-----------+
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

Use CROSS JOIN and UNNEST

To flatten an array into multiple rows, use `CROSS JOIN` in conjunction with the `UNNEST` operator, as in this example:

```
WITH dataset AS (
  SELECT
    'engineering' as department,
    ARRAY['Sharon', 'John', 'Bob', 'Sally'] as users
)
SELECT department, names FROM dataset
CROSS JOIN UNNEST(users) as t(names)
```

This query returns:

```
+----------------------+
| department  | names  |
+----------------------+
| engineering | Sharon |
+----------------------|
| engineering | John   |
+----------------------|
| engineering | Bob    |
+----------------------|
| engineering | Sally  |
+----------------------+
```

To flatten an array of key-value pairs, transpose selected keys into columns, as in this example:

```
WITH
dataset AS (
  SELECT
    'engineering' as department,
     ARRAY[
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
     ] AS people
  )
SELECT names['first'] AS
 first_name,
 names['last'] AS last_name,
 department FROM dataset
CROSS JOIN UNNEST(people) AS t(names)
```

This query returns:

```
+--------------------------------------+
| first_name | last_name | department  |
+--------------------------------------+
| Bob        | Smith     | engineering |
| Jane       | Doe       | engineering |
| Billy      | Smith     | engineering |
+--------------------------------------+
```

From a list of employees, select the employee with the highest combined scores. `UNNEST` can be used in the `FROM` clause without a preceding `CROSS JOIN` as it is the default join operator and therefore implied.

```
WITH
dataset AS (
  SELECT ARRAY[
    CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
    CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
    CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
  ] AS users
```

```
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, person.department, SUM(score) AS total_score FROM users
GROUP BY (person.name, person.department)
ORDER BY (total_score) DESC
LIMIT 1
```

This query returns:

```
+--------------------------------+
| name | department | total_score |
+--------------------------------+
| Amy  | devops     | 54          |
+--------------------------------+
```

From a list of employees, select the employee with the highest individual score.

```
WITH
dataset AS (
 SELECT ARRAY[
   CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
   CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
   CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
 ] AS users
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, score FROM users
ORDER BY (score) DESC
```

```
LIMIT 1
```

This query returns:

```
+--------------+
| name | score |
+--------------+
| Amy  | 15    |
+--------------+
```

Considerations for CROSS JOIN and UNNEST

If UNNEST is used on one or more arrays in the query, and one of the arrays is NULL, the query returns no rows. If UNNEST is used on an array that is an empty string, the empty string is returned.

For example, in the following query, because the second array is null, the query returns no rows.

```
SELECT
    col1,
    col2
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY []) AS t(col2)
```

In this next example, the second array is modified to contain an empty string. For each row, the query returns the value in col1 and an empty string for the value in col2. The empty string in the second array is required in order for the values in the first array to be returned.

```
SELECT
    col1,
    col2
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY ['']) AS t(col2)
```

Create arrays from subqueries

Create an array from a collection of rows.

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
```

```
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

This query returns:

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+
```

To create an array of unique values from a set of rows, use the `distinct` keyword.

```
WITH
dataset AS (
  SELECT ARRAY [1,2,2,3,3,4,5] AS items
)
SELECT array_agg(distinct i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

This query returns the following result. Note that ordering is not guaranteed.

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+
```

For more information about using the `array_agg` function, see Aggregate functions in the Trino documentation.

Filter arrays

Create an array from a collection of rows if they match the filter criteria.

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
```

```
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE i > 3
```

This query returns:

```
+-------------+
| array_items |
+-------------+
| [4, 5]      |
+-------------+
```

Filter an array based on whether one of its elements contain a specific value, such as 2, as in this example:

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
)
SELECT i AS array_items FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE contains(i, 2)
```

This query returns:

```
+--------------+
| array_items  |
+--------------+
| [1, 2, 3, 4] |
+--------------+
```

Use the `filter` function

```
filter(ARRAY [list_of_values], boolean_function)
```

You can use the `filter` function on an `ARRAY` expression to create a new array that is the subset of the items in the *`list_of_values`* for which *`boolean_function`* is true. The `filter` function can be useful in cases in which you cannot use the *`UNNEST`* function.

The following example filters for values greater than zero in the array `[1,0,5,-1]`.

```
SELECT filter(ARRAY [1,0,5,-1], x -> x>0)
```

Results

`[1,5]`

The following example filters for the non-null values in the array `[-1, NULL, 10, NULL]`.

```
SELECT filter(ARRAY [-1, NULL, 10, NULL], q -> q IS NOT NULL)
```

Results

`[-1,10]`

Sort arrays

To create a sorted array of unique values from a set of rows, you can use the array_sort function, as in the following example.

```
WITH
dataset AS (
  SELECT ARRAY[3,1,2,5,2,3,6,3,4,5] AS items
)
SELECT array_sort(array_agg(distinct i)) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

This query returns:

```
+--------------------+
| array_items        |
+--------------------+
| [1, 2, 3, 4, 5, 6] |
```

```
+---------------------+
```

For information about expanding an array into multiple rows, see Flatten nested arrays.

Use aggregation functions with arrays

- To add values within an array, use SUM, as in the following example.
- To aggregate multiple rows within an array, use array_agg. For information, see Create arrays from subqueries.

> **Note**
>
> ORDER BY is supported for aggregation functions starting in Athena engine version 2.

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, sum(val) AS total
FROM item, UNNEST(array_items) AS t(val)
GROUP BY array_items;
```

In the last SELECT statement, instead of using sum() and UNNEST, you can use reduce() to decrease processing time and data transfer, as in the following example.

```
WITH
dataset AS (
  SELECT ARRAY
  [
```

```
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, reduce(array_items, 0 , (s, x) -> s + x, s -> s) AS total
FROM item;
```

Either query returns the following results. The order of returned results is not guaranteed.

```
+----------------------+
| array_items  | total |
+----------------------+
| [1, 2, 3, 4] | 10    |
| [5, 6, 7, 8] | 26    |
| [9, 0]       | 9     |
+----------------------+
```

Convert arrays to strings

To convert an array into a single string, use the `array_join` function. The following standalone example creates a table called `dataset` that contains an aliased array called `words`. The query uses `array_join` to join the array elements in `words`, separate them with spaces, and return the resulting string in an aliased column called `welcome_msg`.

```
WITH
dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT array_join(words, ' ') AS welcome_msg
FROM dataset
```

This query returns:

```
+---------------------+
| welcome_msg         |
+---------------------+
```

```
| hello amazon athena |
+---------------------+
```

Use arrays to create maps

Maps are key-value pairs that consist of data types available in Athena. To create maps, use the MAP operator and pass it two arrays: the first is the column (key) names, and the second is values. All values in the arrays must be of the same type. If any of the map value array elements need to be of different types, you can convert them later.

Examples

This example selects a user from a dataset. It uses the MAP operator and passes it two arrays. The first array includes values for column names, such as "first", "last", and "age". The second array consists of values for each of these columns, such as "Bob", "Smith", "35".

```
WITH dataset AS (
  SELECT MAP(
    ARRAY['first', 'last', 'age'],
    ARRAY['Bob', 'Smith', '35']
  ) AS user
)
SELECT user FROM dataset
```

This query returns:

```
+---------------------------------+
| user                            |
+---------------------------------+
| {last=Smith, first=Bob, age=35} |
+---------------------------------+
```

You can retrieve Map values by selecting the field name followed by [key_name], as in this example:

```
WITH dataset AS (
 SELECT MAP(
   ARRAY['first', 'last', 'age'],
   ARRAY['Bob', 'Smith', '35']
 ) AS user
```

```
)
SELECT user['first'] AS first_name FROM dataset
```

This query returns:

```
+------------+
| first_name |
+------------+
| Bob        |
+------------+
```

Query arrays with complex types and nested structures

Your source data often contains arrays with complex data types and nested structures. Examples in this section show how to change element's data type, locate elements within arrays, and find keywords using Athena queries.

Topics

Create a ROW

> **Note**
>
> The examples in this section use ROW as a means to create sample data to work with. When you query tables within Athena, you do not need to create ROW data types, as they are already created from your data source. When you use CREATE_TABLE, Athena defines a STRUCT in it, populates it with data, and creates the ROW data type for you, for each row in the dataset. The underlying ROW data type consists of named fields of any supported SQL data types.

```
WITH dataset AS (
 SELECT
   ROW('Bob', 38) AS users
 )
SELECT * FROM dataset
```

This query returns:

```
+-------------------------+
| users                   |
+-------------------------+
| {field0=Bob, field1=38} |
+-------------------------+
```

Change field names in arrays using CAST

To change the field name in an array that contains ROW values, you can CAST the ROW declaration:

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)
    ) AS users
)
SELECT * FROM dataset
```

This query returns:

```
+--------------------+
| users              |
+--------------------+
| {NAME=Bob, AGE=38} |
+--------------------+
```

> **Note**
>
> In the example above, you declare `name` as a `VARCHAR` because this is its type in Presto. If you declare this `STRUCT` inside a `CREATE TABLE` statement, use `String` type because Hive defines this data type as `String`.

Filter arrays using the . notation

In the following example, select the `accountId` field from the `userIdentity` column of a AWS CloudTrail logs table by using the dot . notation. For more information, see Querying AWS CloudTrail Logs.

```
SELECT
  CAST(useridentity.accountid AS bigint) as newid
FROM cloudtrail_logs
LIMIT 2;
```

This query returns:

```
+--------------+
| newid        |
+--------------+
| 112233445566 |
+--------------+
| 998877665544 |
+--------------+
```

To query an array of values, issue this query:

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)),
    CAST(ROW('Alice', 35) AS ROW(name VARCHAR, age INTEGER)),
    CAST(ROW('Jane', 27) AS ROW(name VARCHAR, age INTEGER))
  ] AS users
)
SELECT * FROM dataset
```

It returns this result:

```
+-----------------------------------------------------------------+
| users                                                           |
+-----------------------------------------------------------------+
| [{NAME=Bob, AGE=38}, {NAME=Alice, AGE=35}, {NAME=Jane, AGE=27}] |
+-----------------------------------------------------------------+
```

Filter arrays with nested values

Large arrays often contain nested structures, and you need to be able to filter, or search, for values within them.

To define a dataset for an array of values that includes a nested BOOLEAN value, issue this query:

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT * FROM dataset
```

It returns this result:

```
+-----------------------------------------------------------+
| sites                                                     |
+-----------------------------------------------------------+
| {HOSTNAME=aws.amazon.com, FLAGGEDACTIVITY={ISNEW=true}}  |
+-----------------------------------------------------------+
```

Next, to filter and access the BOOLEAN value of that element, continue to use the dot . notation.

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT sites.hostname, sites.flaggedactivity.isnew
FROM dataset
```

This query selects the nested fields and returns this result:

```
+------------------------+
| hostname       | isnew |
+------------------------+
```

```
| aws.amazon.com | true  |
+------------------------+
```

Filter arrays using UNNEST

To filter an array that includes a nested structure by one of its child elements, issue a query with an UNNEST operator. For more information about UNNEST, see Flattening Nested Arrays.

For example, this query finds host names of sites in the dataset.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('news.cnn.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('netflix.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity ROW(isNew
 BOOLEAN))
    )
  ] as items
)
SELECT sites.hostname, sites.flaggedActivity.isNew
FROM dataset, UNNEST(items) t(sites)
WHERE sites.flaggedActivity.isNew = true
```

It returns:

```
+------------------------+
| hostname       | isnew |
+------------------------+
| aws.amazon.com | true  |
+------------------------+
```

Find keywords in arrays using `regexp_like`

The following examples illustrate how to search a dataset for a keyword within an element inside an array, using the regexp_like function. It takes as an input a regular expression pattern to evaluate, or a list of terms separated by a pipe (|), evaluates the pattern, and determines if the specified string contains it.

The regular expression pattern needs to be contained within the string, and does not have to match it. To match the entire string, enclose the pattern with ^ at the beginning of it, and $ at the end, such as `'^pattern$'`.

Consider an array of sites containing their host name, and a `flaggedActivity` element. This element includes an `ARRAY`, containing several `MAP` elements, each listing different popular keywords and their popularity count. Assume you want to find a particular keyword inside a `MAP` in this array.

To search this dataset for sites with a specific keyword, we use `regexp_like` instead of the similar SQL `LIKE` operator, because searching for a large number of keywords is more efficient with `regexp_like`.

Example Example 1: Using `regexp_like`

The query in this example uses the `regexp_like` function to search for terms `'politics|bigdata'`, found in values within arrays:

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('news.cnn.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
        MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
        MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
```

```
    ),
    CAST(
      ROW('netflix.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
        MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
        MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    )
 ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
SELECT hostname
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
```

This query returns two sites:

```
+----------------+
| hostname       |
+----------------+
| aws.amazon.com |
+----------------+
| news.cnn.com   |
+----------------+
```

Example Example 2: Using `regexp_like`

The query in the following example adds up the total popularity scores for the sites matching your search terms with the `regexp_like` function, and then orders them from highest to lowest.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
```

```
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('news.cnn.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
        MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
        MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('netflix.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
        MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
        MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    )
  ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
SELECT hostname, array_agg(flags['term']) AS terms, SUM(CAST(flags['count'] AS
 INTEGER)) AS total
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
ORDER BY total DESC
```

This query returns two sites:

```
+------------------------------------+
| hostname        | terms    | total  |
```

```
+----------------+-------------------+
| news.cnn.com   | politics |  241   |
+----------------+-------------------+
| aws.amazon.com | bigdata |  10    |
+----------------+-------------------+
```

Query geospatial data

Geospatial data contains identifiers that specify a geographic position for an object. Examples of this type of data include weather reports, map directions, tweets with geographic positions, store locations, and airline routes. Geospatial data plays an important role in business analytics, reporting, and forecasting.

Geospatial identifiers, such as latitude and longitude, allow you to convert any mailing address into a set of geographic coordinates.

What is a geospatial query?

Geospatial queries are specialized types of SQL queries supported in Athena. They differ from non-spatial SQL queries in the following ways:

- Using the following specialized geometry data types: `point`, `line`, `multiline`, `polygon`, and `multipolygon`.
- Expressing relationships between geometry data types, such as `distance`, `equals`, `crosses`, `touches`, `overlaps`, `disjoint`, and others.

Using geospatial queries in Athena, you can run these and other similar operations:

- Find the distance between two points.
- Check whether one area (polygon) contains another.
- Check whether one line crosses or touches another line or polygon.

For example, to obtain a `point` geometry data type from values of type `double` for the geographic coordinates of Mount Rainier in Athena, use the `ST_Point (longitude, latitude)` geospatial function, as in the following example.

```
ST_Point(-121.7602, 46.8527)
```

Input data formats and geometry data types

To use geospatial functions in Athena, input your data in the WKT format, or use the Hive JSON SerDe. You can also use the geometry data types supported in Athena.

Input data formats

To handle geospatial queries, Athena supports input data in these data formats:

- **WKT (Well-known Text)**. In Athena, WKT is represented as a `varchar(x)` or `string` data type.
- **JSON-encoded geospatial data**. To parse JSON files with geospatial data and create tables for them, Athena uses the Hive JSON SerDe. For more information about using this SerDe in Athena, see JSON SerDe libraries.

Geometry data types

To handle geospatial queries, Athena supports these specialized geometry data types:

- `point`
- `line`
- `polygon`
- `multiline`
- `multipolygon`

Supported geospatial functions

For information about the geospatial functions in Athena engine version 3, see Geospatial functions in the Trino documentation.

Examples: Geospatial queries

The examples in this topic create two tables from sample data available on GitHub and query the tables based on the data. The sample data, which are for illustration purposes only and are not guaranteed to be accurate, are in the following files:

- **`earthquakes.csv`** – Lists earthquakes that occurred in California. The example `earthquakes` table uses fields from this data.

- **[california-counties.json](#)** – Lists county data for the state of California in [ESRI-compliant GeoJSON format](#). The data includes many fields such as AREA, PERIMETER, STATE, COUNTY, and NAME, but the example counties table uses only two: Name (string), and BoundaryShape (binary).

> **Note**
>
> Athena uses the com.esri.json.hadoop.EnclosedEsriJsonInputFormat to convert the JSON data to geospatial binary format.

The following code example creates a table called earthquakes:

```
CREATE external TABLE earthquakes
(
 earthquake_date string,
 latitude double,
 longitude double,
 depth double,
 magnitude double,
 magtype string,
 mbstations string,
 gap string,
 distance string,
 rms string,
 source string,
 eventid string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION 's3://amzn-s3-demo-bucket/my-query-log/csv/';
```

The following code example creates a table called counties:

```
CREATE external TABLE IF NOT EXISTS counties
 (
 Name string,
 BoundaryShape binary
 )
ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.EsriJsonSerDe'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedEsriJsonInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
```

```
LOCATION 's3://amzn-s3-demo-bucket/my-query-log/json/';
```

The following example query uses the CROSS JOIN function on the counties and earthquake tables. The example uses ST_CONTAINS to query for counties whose boundaries include earthquake locations, which are specified with ST_POINT. The query groups such counties by name, orders them by count, and returns them in descending order.

```
SELECT counties.name,
        COUNT(*) cnt
FROM counties
CROSS JOIN earthquakes
WHERE ST_CONTAINS (ST_GeomFromLegacyBinary(counties.boundaryshape),
 ST_POINT(earthquakes.longitude, earthquakes.latitude))
GROUP BY  counties.name
ORDER BY  cnt DESC
```

This query returns:

```
+------------------------+
| name             | cnt |
+------------------------+
| Kern             | 36  |
+------------------------+
| San Bernardino   | 35  |
+------------------------+
| Imperial         | 28  |
+------------------------+
| Inyo             | 20  |
+------------------------+
| Los Angeles      | 18  |
+------------------------+
| Riverside        | 14  |
+------------------------+
| Monterey         | 14  |
+------------------------+
| Santa Clara      | 12  |
+------------------------+
| San Benito       | 11  |
+------------------------+
| Fresno           | 11  |
+------------------------+
| San Diego        | 7   |
```

```
+------------------------+
| Santa Cruz       | 5   |
+------------------------+
| Ventura          | 3   |
+------------------------+
| San Luis Obispo  | 3   |
+------------------------+
| Orange           | 2   |
+------------------------+
| San Mateo        | 1   |
+------------------------+
```

Additional resources

For additional examples of geospatial queries, see the following blog posts:

- Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda
- Visualize over 200 years of global climate data using Amazon Athena and Amazon QuickSight.
- Querying OpenStreetMap with Amazon Athena

Query JSON data

Amazon Athena lets you query JSON-encoded data, extract data from nested JSON, search for values, and find length and size of JSON arrays. To learn the basics of querying JSON data in Athena, consider the following sample planet data:

```
{name:"Mercury",distanceFromSun:0.39,orbitalPeriod:0.24,dayLength:58.65}
{name:"Venus",distanceFromSun:0.72,orbitalPeriod:0.62,dayLength:243.02}
{name:"Earth",distanceFromSun:1.00,orbitalPeriod:1.00,dayLength:1.00}
{name:"Mars",distanceFromSun:1.52,orbitalPeriod:1.88,dayLength:1.03}
```

Notice how each record (essentially, each row in the table) is on a separate line. To query this JSON data, you can use a `CREATE TABLE` statement like the following:

```
CREATE EXTERNAL TABLE `planets_json`(
  `name` string,
  `distancefromsun` double,
  `orbitalperiod` double,
  `daylength` double)
ROW FORMAT SERDE
```

```
  'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/json/'
```

To query the data, use a simple SELECT statement like the following example.

```
SELECT * FROM planets_json
```

The query results look like the following.

#	name	distancefromsun	orbitalperiod	daylength
1	Mercury	0.39	0.24	58.65
2	Venus	0.72	0.62	243.02
3	Earth	1.0	1.0	1.0
4	Mars	1.52	1.88	1.03

Notice how the CREATE TABLE statement uses the OpenX JSON SerDe, which requires each JSON record to be on a separate line. If the JSON is in pretty print format, or if all records are on a single line, the data will not be read correctly.

To query JSON data that is in pretty print format, you can use the Amazon Ion Hive SerDe instead of the OpenX JSON SerDe. Consider the previous data stored in pretty print format:

```
{
  name:"Mercury",
  distanceFromSun:0.39,
  orbitalPeriod:0.24,
  dayLength:58.65
}
{
  name:"Venus",
  distanceFromSun:0.72,
```

```
  orbitalPeriod:0.62,
  dayLength:243.02
}
{
  name:"Earth",
  distanceFromSun:1.00,
  orbitalPeriod:1.00,
  dayLength:1.00
}
{
  name:"Mars",
  distanceFromSun:1.52,
  orbitalPeriod:1.88,
  dayLength:1.03
}
```

To query this data without reformatting, you can use a `CREATE TABLE` statement like the following. Notice that, instead of specifying the OpenX JSON SerDe, the statement specifies `STORED AS ION`.

```
CREATE EXTERNAL TABLE `planets_ion`(
  `name` string,
  `distancefromsun` DECIMAL(10, 2),
  `orbitalperiod` DECIMAL(10, 2),
  `daylength` DECIMAL(10, 2))
STORED AS ION
LOCATION
  's3://amzn-s3-demo-bucket/json-ion/'
```

The query `SELECT * FROM planets_ion` produces the same results as before. For more information about creating tables in this way using the Amazon Ion Hive SerDe, see Create Amazon Ion tables.

The preceding example JSON data does not contain complex data types such as nested arrays or structs. For more information about querying nested JSON data, see Example: deserializing nested JSON.

Topics

- Best practices for reading JSON data
- Extract JSON data from strings
- Search for values in JSON arrays

- Get the length and size of JSON arrays
- Troubleshoot JSON queries

Best practices for reading JSON data

JavaScript Object Notation (JSON) is a common method for encoding data structures as text. Many applications and tools output data that is JSON-encoded.

In Amazon Athena, you can create tables from external data and include the JSON-encoded data in them. For such types of source data, use Athena together with JSON SerDe libraries.

Use the following tips to read JSON-encoded data:

- Choose the right SerDe, a native JSON SerDe, `org.apache.hive.hcatalog.data.JsonSerDe`, or an OpenX SerDe, `org.openx.data.jsonserde.JsonSerDe`. For more information, see JSON SerDe libraries.
- Make sure that each JSON-encoded record is represented on a separate line, not pretty-printed.

> **Note**
>
> The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

- Generate your JSON-encoded data in case-insensitive columns.
- Provide an option to ignore malformed records, as in this example.

```
CREATE EXTERNAL TABLE json_table (
  column_a string,
  column_b int
 )
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 WITH SERDEPROPERTIES ('ignore.malformed.json' = 'true')
 LOCATION 's3://amzn-s3-demo-bucket/path/';
```

- Convert fields in source data that have an undetermined schema to JSON-encoded strings in Athena.

When Athena creates tables backed by JSON data, it parses the data based on the existing and predefined schema. However, not all of your data may have a predefined schema. To simplify schema management in such cases, it is often useful to convert fields in source data that have an undetermined schema to JSON strings in Athena, and then use JSON SerDe libraries.

For example, consider an IoT application that publishes events with common fields from different sensors. One of those fields must store a custom payload that is unique to the sensor sending the event. In this case, since you don't know the schema, we recommend that you store the information as a JSON-encoded string. To do this, convert data in your Athena table to JSON, as in the following example. You can also convert JSON-encoded data to Athena data types.

Topics

Convert Athena data types to JSON

To convert Athena data types to JSON, use CAST.

```
WITH dataset AS (
  SELECT
    CAST('HELLO ATHENA' AS JSON) AS hello_msg,
    CAST(12345 AS JSON) AS some_int,
    CAST(MAP(ARRAY['a', 'b'], ARRAY[1,2]) AS JSON) AS some_map
)
SELECT * FROM dataset
```

This query returns:

```

+-------------------------------------------+
| hello_msg      | some_int | some_map      |
+-------------------------------------------+
| "HELLO ATHENA" | 12345    | {"a":1,"b":2} |
+-------------------------------------------+
```

Convert JSON to Athena data types

To convert JSON data to Athena data types, use CAST.

> **Note**
>
> In this example, to denote strings as JSON-encoded, start with the JSON keyword and use single quotes, such as JSON '12345'

```
WITH dataset AS (
  SELECT
    CAST(JSON '"HELLO ATHENA"' AS VARCHAR) AS hello_msg,
    CAST(JSON '12345' AS INTEGER) AS some_int,
    CAST(JSON '{"a":1,"b":2}' AS MAP(VARCHAR, INTEGER)) AS some_map
)
SELECT * FROM dataset
```

This query returns:

```

+-------------------------------------+
| hello_msg    | some_int | some_map  |
+-------------------------------------+
| HELLO ATHENA | 12345    | {a:1,b:2} |
+-------------------------------------+
```

Extract JSON data from strings

You may have source data containing JSON-encoded strings that you do not necessarily want to deserialize into a table in Athena. In this case, you can still run SQL operations on this data, using the JSON functions available in Presto.

Consider this JSON string as an example dataset.

```
{"name": "Susan Smith",
"org": "engineering",
"projects":
    [
     {"name":"project1", "completed":false},
     {"name":"project2", "completed":true}
```

```
    ]
}
```

Examples: Extract properties

To extract the `name` and `projects` properties from the JSON string, use the `json_extract` function as in the following example. The `json_extract` function takes the column containing the JSON string, and searches it using a `JSONPath`-like expression with the dot `.` notation.

> **Note**
>
> `JSONPath` performs a simple tree traversal. It uses the `$` sign to denote the root of the JSON document, followed by a period and an element nested directly under the root, such as `$.name`.

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},
           {"name":"project2", "completed":true}]}'
    AS myblob
)
SELECT
  json_extract(myblob, '$.name') AS name,
  json_extract(myblob, '$.projects') AS projects
FROM dataset
```

The returned value is a JSON-encoded string, and not a native Athena data type.

```
+-----------------------------------------------------------------------------------------
+
| name           | projects
         |
+-----------------------------------------------------------------------------------------
+
| "Susan Smith"  | [{"name":"project1","completed":false},
{"name":"project2","completed":true}] |
+-----------------------------------------------------------------------------------------
+
```

To extract the scalar value from the JSON string, use the `json_extract_scalar(json, json_path)` function. It is similar to `json_extract`, but returns a `varchar` string value instead of a JSON-encoded string. The value for the *json_path* parameter must be a scalar (a Boolean, number, or string).

> **Note**
>
> Do not use the `json_extract_scalar` function on arrays, maps, or structs.

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT
  json_extract_scalar(myblob, '$.name') AS name,
  json_extract_scalar(myblob, '$.projects') AS projects
FROM dataset
```

This query returns:

```
+---------------------------+
| name           | projects |
+---------------------------+
| Susan Smith    |          |
+---------------------------+
```

To obtain the first element of the `projects` property in the example array, use the `json_array_get` function and specify the index position.

```
WITH dataset AS (
  SELECT '{"name": "Bob Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT json_array_get(json_extract(myblob, '$.projects'), 0) AS item
```

```
FROM dataset
```

It returns the value at the specified index position in the JSON-encoded array.

```
+---------------------------------------+
| item                                  |
+---------------------------------------+
| {"name":"project1","completed":false} |
+---------------------------------------+
```

To return an Athena string type, use the `[]` operator inside a `JSONPath` expression, then Use the `json_extract_scalar` function. For more information about `[]`, see Access array elements.

```
WITH dataset AS (
   SELECT '{"name": "Bob Smith",
             "org": "engineering",
             "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
     AS myblob
)
SELECT json_extract_scalar(myblob, '$.projects[0].name') AS project_name
FROM dataset
```

It returns this result:

```
+--------------+
| project_name |
+--------------+
| project1     |
+--------------+
```

Search for values in JSON arrays

To determine if a specific value exists inside a JSON-encoded array, use the `json_array_contains` function.

The following query lists the names of the users who are participating in "project2".

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": ["project1"]}'),
```

```
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects": ["project1",
 "project2", "project3"]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": ["project1",
 "project2"]}')
  ) AS t (users)
)
SELECT json_extract_scalar(users, '$.name') AS user
FROM dataset
WHERE json_array_contains(json_extract(users, '$.projects'), 'project2')
```

This query returns a list of users.

```
+-------------+
| user        |
+-------------+
| Susan Smith |
+-------------+
| Jane Smith  |
+-------------+
```

The following query example lists the names of users who have completed projects along with the total number of completed projects. It performs these actions:

- Uses nested SELECT statements for clarity.
- Extracts the array of projects.
- Converts the array to a native array of key-value pairs using CAST.
- Extracts each individual array element using the UNNEST operator.
- Filters obtained values by completed projects and counts them.

Note

When using CAST to MAP you can specify the key element as VARCHAR (native String in Presto), but leave the value as JSON, because the values in the MAP are of different types: String for the first key-value pair, and Boolean for the second.

```
WITH dataset AS (
  SELECT * FROM (VALUES
```

```
    (JSON '{"name": "Bob Smith",
             "org": "legal",
             "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
             "org": "engineering",
             "projects": [{"name":"project2", "completed":true},
                          {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
),
employees AS (
  SELECT users, CAST(json_extract(users, '$.projects') AS
    ARRAY(MAP(VARCHAR, JSON))) AS projects_array
  FROM dataset
),
names AS (
  SELECT json_extract_scalar(users, '$.name') AS name, projects
  FROM employees, UNNEST (projects_array) AS t(projects)
)
SELECT name, count(projects) AS completed_projects FROM names
WHERE cast(element_at(projects, 'completed') AS BOOLEAN) = true
GROUP BY name
```

This query returns the following result:

```
+----------------------------------+
| name        | completed_projects |
+----------------------------------+
| Susan Smith | 2                  |
+----------------------------------+
| Jane Smith  | 1                  |
+----------------------------------+
```

Get the length and size of JSON arrays

To get the length and size of JSON arrays, you can use the `json_array_length` and `json_size` functions.

Example: `json_array_length`

To obtain the length of a JSON-encoded array, use the `json_array_length` function.

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name":
            "Bob Smith",
            "org":
            "legal",
            "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
            "org": "engineering",
            "projects": [{"name":"project2", "completed":true},
                         {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_array_length(json_extract(users, '$.projects')) as count
FROM dataset
ORDER BY count DESC
```

This query returns this result:

```
+---------------------+
| name        | count |
+---------------------+
| Susan Smith | 2     |
+---------------------+
| Bob Smith   | 1     |
+---------------------+
| Jane Smith  | 1     |
+---------------------+
```

Example: `json_size`

To obtain the size of a JSON-encoded array or object, use the `json_size` function, and specify the column containing the JSON string and the `JSONPath` expression to the array or object.

```
WITH dataset AS (
  SELECT * FROM (VALUES
```

```
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": [{"name":"project1",
 "completed":false}]}'),
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects":
 [{"name":"project2", "completed":true},{"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": [{"name":"project2",
 "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_size(users, '$.projects') as count
FROM dataset
ORDER BY count DESC
```

This query returns this result:

```
+----------------------+
| name         | count |
+----------------------+
| Susan Smith  | 2     |
+----------------------+
| Bob Smith    | 1     |
+----------------------+
| Jane Smith   | 1     |
+----------------------+
```

Troubleshoot JSON queries

For help on troubleshooting issues with JSON-related queries, see JSON related errors or consult the following resources:

- I get errors when I try to read JSON data in Amazon Athena
- How do I resolve "HIVE_CURSOR_ERROR: Row is not a valid JSON object - JSONException: Duplicate key" when reading files from AWS Config in Athena?
- The SELECT COUNT query in Amazon Athena returns only one record even though the input JSON file has multiple records
- How can I see the Amazon S3 source file for a row in an Athena table?

See also Considerations and limitations for SQL queries in Amazon Athena.

Use Machine Learning (ML) with Amazon Athena

Machine Learning (ML) with Amazon Athena lets you use Athena to write SQL statements that run Machine Learning (ML) inference using Amazon SageMaker AI. This feature simplifies access to ML models for data analysis, eliminating the need to use complex programming methods to run inference.

To use ML with Athena, you define an ML with Athena function with the `USING EXTERNAL FUNCTION` clause. The function points to the SageMaker AI model endpoint that you want to use and specifies the variable names and data types to pass to the model. Subsequent clauses in the query reference the function to pass values to the model. The model runs inference based on the values that the query passes and then returns inference results. For more information about SageMaker AI and how SageMaker AI endpoints work, see the Amazon SageMaker AI Developer Guide.

For an example that uses ML with Athena and SageMaker AI inference to detect an anomalous value in a result set, see the AWS Big Data Blog article Detecting anomalous values by invoking the Amazon Athena machine learning inference function.

Considerations and limitations

- **Available Regions –** The Athena ML feature is available in the AWS Regions where Athena engine version 2 or later are supported.
- **SageMaker AI model endpoint must accept and return `text/csv` –** For more information about data formats, see Common data formats for inference in the *Amazon SageMaker AI Developer Guide*.
- **Athena does not send CSV headers –** If your SageMaker AI endpoint is `text/csv`, your input handler should not assume that the first line of the input is a CSV header. Because Athena does not send CSV headers, the output returned to Athena will contain one less row than Athena expects and cause an error.
- **SageMaker AI endpoint scaling –** Make sure that the referenced SageMaker AI model endpoint is sufficiently scaled up for Athena calls to the endpoint. For more information, see Automatically scale SageMaker AI models in the *Amazon SageMaker AI Developer Guide* and CreateEndpointConfig in the *Amazon SageMaker AI API Reference*.
- **IAM permissions –** To run a query that specifies an ML with Athena function, the IAM principal running the query must be allowed to perform the `sagemaker:InvokeEndpoint` action for the

referenced SageMaker AI model endpoint. For more information, see [Allow access for ML with Athena](#).

- **ML with Athena functions cannot be used in `GROUP BY` clauses directly**

Topics

- [Use ML with Athena syntax](#)
- [See customer use examples](#)

Use ML with Athena syntax

The `USING EXTERNAL FUNCTION` clause specifies an ML with Athena function or multiple functions that can be referenced by a subsequent `SELECT` statement in the query. You define the function name, variable names, and data types for the variables and return values.

Synopsis

The following syntax shows a `USING EXTERNAL FUNCTION` clause that specifies an ML with Athena function.

```
USING EXTERNAL FUNCTION ml_function_name (variable1 data_type[, variable2 data_type]
[,...])
RETURNS data_type
SAGEMAKER 'sagemaker_endpoint'
SELECT ml_function_name()
```

Parameters

USING EXTERNAL FUNCTION *ml_function_name* (*variable1 data_type*[, *variable2 data_type*][,...])

ml_function_name defines the function name, which can be used in subsequent query clauses. Each *variable data_type* specifies a named variable and its corresponding data type that the SageMaker AI model accepts as input. The data type specified must be a supported Athena data type.

RETURNS *data_type*

data_type specifies the SQL data type that *ml_function_name* returns to the query as output from the SageMaker AI model.

SAGEMAKER '*sagemaker_endpoint*'

sagemaker_endpoint specifies the endpoint of the SageMaker AI model.

SELECT [...] *ml_function_name*(*expression*) [...]

The SELECT query that passes values to function variables and the SageMaker AI model to return a result. *ml_function_name* specifies the function defined earlier in the query, followed by an *expression* that is evaluated to pass values. Values that are passed and returned must match the corresponding data types specified for the function in the `USING EXTERNAL FUNCTION` clause.

Example

The following example demonstrates a query using ML with Athena.

Example

```
USING EXTERNAL FUNCTION predict_customer_registration(age INTEGER)
    RETURNS DOUBLE
    SAGEMAKER 'xgboost-2019-09-20-04-49-29-303'
SELECT predict_customer_registration(age) AS probability_of_enrolling, customer_id
     FROM "sampledb"."ml_test_dataset"
     WHERE predict_customer_registration(age) < 0.5;
```

See customer use examples

The following videos, which use the Preview version of Machine Learning (ML) with Amazon Athena, showcase ways in which you can use SageMaker AI with Athena.

Predicting customer churn

The following video shows how to combine Athena with the machine learning capabilities of Amazon SageMaker AI to predict customer churn.

[Predict customer churn using Amazon Athena and Amazon SageMaker AI]

Detecting botnets

The following video shows how one company uses Amazon Athena and Amazon SageMaker AI to detect botnets.

Detect botnets using Amazon Athena and Amazon SageMaker AI

Query with user defined functions

User Defined Functions (UDF) in Amazon Athena allow you to create custom functions to process records or groups of records. A UDF accepts parameters, performs work, and then returns a result.

To use a UDF in Athena, you write a `USING EXTERNAL FUNCTION` clause before a `SELECT` statement in a SQL query. The `SELECT` statement references the UDF and defines the variables that are passed to the UDF when the query runs. The SQL query invokes a Lambda function using the Java runtime when it calls the UDF. UDFs are defined within the Lambda function as methods in a Java deployment package. Multiple UDFs can be defined in the same Java deployment package for a Lambda function. You also specify the name of the Lambda function in the `USING EXTERNAL FUNCTION` clause.

You have two options for deploying a Lambda function for Athena UDFs. You can deploy the function directly using Lambda, or you can use the AWS Serverless Application Repository. To find existing Lambda functions for UDFs, you can search the public AWS Serverless Application Repository or your private repository and then deploy to Lambda. You can also create or modify Java source code, package it into a JAR file, and deploy it using Lambda or the AWS Serverless Application Repository. For example Java source code and packages to get you started, see Create and deploy a UDF using Lambda. For more information about Lambda, see AWS Lambda Developer Guide. For more information about AWS Serverless Application Repository, see the AWS Serverless Application Repository Developer Guide.

For an example that uses UDFs with Athena to translate and analyze text, see the AWS Machine Learning Blog article Translate and analyze text using SQL functions with Amazon Athena, Amazon Translate, and Amazon Comprehend, or watch the video.

For an example of using UDFs to extend geospatial queries in Amazon Athena, see Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda in the *AWS Big Data Blog*.

Topics

Videos on UDFs in Athena

Watch the following videos to learn more about using UDFs in Athena.

Video: Introducing User Defined Functions (UDFs) in Amazon Athena

The following video shows how you can use UDFs in Amazon Athena to redact sensitive information.

> **Note**
>
> The syntax in this video is prerelease, but the concepts are the same. Use Athena without the `AmazonAthenaPreviewFunctionality` workgroup.

Introducing user defined functions (UDFs) in Amazon Athena

Video: Translate, analyze, and redact text fields using SQL queries in Amazon Athena

The following video shows how you can use UDFs in Amazon Athena together with other AWS services to translate and analyze text.

> **Note**
>
> The syntax in this video is prerelease, but the concepts are the same. For the correct syntax, see the related blog post Translate, redact, and analyze text using SQL functions with Amazon Athena, Amazon Translate, and Amazon Comprehend on the *AWS Machine Learning Blog*.

Translate, analyze, and redact text fields using SQL queries in Amazon Athena

Considerations and limitations

Consider the following points when you use user defined function (UDFs) in Athena.

- **Built-in Athena functions** – Built-in functions in Athena are designed to be highly performant. We recommend that you use built-in functions over UDFs when possible. For more information about built-in functions, see Functions in Amazon Athena.
- **Scalar UDFs only** – Athena only supports scalar UDFs, which process one row at a time and return a single column value. Athena passes a batch of rows, potentially in parallel, to the UDF

each time it invokes Lambda. When designing UDFs and queries, be mindful of the potential impact to network traffic of this processing.

- **UDF handler functions use abbreviated format** – Use abbreviated format (not full format), for your UDF functions (for example, `package.Class` instead of `package.Class::method`).
- **UDF methods must be lowercase** – UDF methods must be in lowercase; camel case is not permitted.
- **UDF methods require parameters** – UDF methods must have at least one input parameter. Attempting to invoke a UDF defined without input parameters causes a runtime exception. UDFs are meant to perform functions against data records, but a UDF without arguments takes in no data, so an exception occurs.
- **Java runtime support** – Currently, Athena UDFs support the Java 8 and Java 11 runtimes for Lambda. For more information, see Building Lambda functions with Java in the *AWS Lambda Developer Guide*.
- **IAM permissions** – To run and create UDF query statements in Athena, the IAM principal running the query must be allowed to perform actions in addition to Athena functions. For more information, see Allow access to Athena UDFs: Example policies.
- **Lambda quotas** – Lambda quotas apply to UDFs. For more information, see Lambda quotas in the *AWS Lambda Developer Guide*.
- **Row-level filtering** – Lake Formation row-level filtering is not supported for UDFs.
- **Views** – You cannot use views with UDFs.
- **Known issues** – For the most up-to-date list of known issues, see Limitations and issues in the awslabs/aws-athena-query-federation section of GitHub.

Query using UDF query syntax

The `USING EXTERNAL FUNCTION` clause specifies a UDF or multiple UDFs that can be referenced by a subsequent `SELECT` statement in the query. You need the method name for the UDF and the name of the Lambda function that hosts the UDF. In place of the Lambda function name, you can use the Lambda ARN. In cross-account scenarios, the Lambda ARN is required.

Synopsis

```
USING EXTERNAL FUNCTION UDF_name(variable1 data_type[, variable2 data_type][,...])
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'
[, EXTERNAL FUNCTION UDF_name2(variable1 data_type[, variable2 data_type][,...])
```

```
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'[,...]]
SELECT  [...] UDF_name(expression) [, UDF_name2(expression)] [...]
```

Parameters

USING EXTERNAL FUNCTION *UDF_name*(*variable1 data_type*[, *variable2 data_type*][,...])

UDF_name specifies the name of the UDF, which must correspond to a Java method within the referenced Lambda function. Each *variable data_type* specifies a named variable and its corresponding data type that the UDF accepts as input. The *data_type* must be one of the supported Athena data types listed in the following table and map to the corresponding Java data type.

Athena data type	Java data type
TIMESTAMP	java.time.LocalDateTime (UTC)
DATE	java.time.LocalDate (UTC)
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
REAL	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL (see RETURNS note)	java.math.BigDecimal
BIGINT	java.lang.Long
INTEGER	java.lang.Int
VARCHAR	java.lang.String
VARBINARY	byte[]
BOOLEAN	java.lang.Boolean

Athena data type	Java data type
ARRAY	java.util.List
ROW	java.util.Map<String, Object>

RETURNS *data_type*

`data_type` specifies the SQL data type that the UDF returns as output. Athena data types listed in the table above are supported. For the `DECIMAL` data type, use the syntax `RETURNS DECIMAL(precision, scale)` where *precision* and *scale* are integers.

LAMBDA '*lambda_function*'

lambda_function specifies the name of the Lambda function to be invoked when running the UDF.

SELECT [...] *UDF_name*(*expression*) [...]

The `SELECT` query that passes values to the UDF and returns a result. *UDF_name* specifies the UDF to use, followed by an *expression* that is evaluated to pass values. Values that are passed and returned must match the corresponding data types specified for the UDF in the `USING EXTERNAL FUNCTION` clause.

Examples

For example queries based on the AthenaUDFHandler.java code on GitHub, see the GitHub Amazon Athena UDF connector page.

Create and deploy a UDF using Lambda

To create a custom UDF, you create a new Java class by extending the `UserDefinedFunctionHandler` class. The source code for the UserDefinedFunctionHandler.java in the SDK is available on GitHub in the awslabs/aws-athena-query-federation/athena-federation-sdk repository, along with example UDF implementations that you can examine and modify to create a custom UDF.

The steps in this section demonstrate writing and building a custom UDF Jar file using Apache Maven from the command line and a deploy.

Perform the following steps to create a custom UDF for Athena using Maven

1. Clone the SDK and prepare your development environment
2. Create your Maven project
3. Add dependencies and plugins to your Maven project
4. Write Java code for the UDFs
5. Build the JAR file
6. Deploy the JAR to AWS Lambda

Clone the SDK and prepare your development environment

Before you begin, make sure that git is installed on your system using `sudo yum install git -y`.

To install the AWS query federation SDK

- Enter the following at the command line to clone the SDK repository. This repository includes the SDK, examples and a suite of data source connectors. For more information about data source connectors, see Use Amazon Athena Federated Query.

```
git clone https://github.com/awslabs/aws-athena-query-federation.git
```

To install prerequisites for this procedure

If you are working on a development machine that already has Apache Maven, the AWS CLI, and the AWS Serverless Application Model build tool installed, you can skip this step.

1. From the root of the `aws-athena-query-federation` directory that you created when you cloned, run the prepare_dev_env.sh script that prepares your development environment.
2. Update your shell to source new variables created by the installation process or restart your terminal session.

```
source ~/.profile
```

> **Important**
>
> If you skip this step, you will get errors later about the AWS CLI or AWS SAM build tool not being able to publish your Lambda function.

Create your Maven project

Run the following command to create your Maven project. Replace *groupId* with the unique ID of your organization, and replace *my-athena-udf* with the name of your application For more information, see How do I make my first Maven project? in Apache Maven documentation.

```
mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=groupId \
-DartifactId=my-athena-udfs
```

Add dependencies and plugins to your Maven project

Add the following configurations to your Maven project `pom.xml` file. For an example, see the pom.xml file in GitHub.

```
<properties>
    <aws-athena-federation-sdk.version>2022.47.1</aws-athena-federation-sdk.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-athena-federation-sdk</artifactId>
        <version>${aws-athena-federation-sdk.version}</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.1</version>
            <configuration>
```

```
                  <createDependencyReducedPom>false</createDependencyReducedPom>
                  <filters>
                      <filter>
                          <artifact>*:*</artifact>
                          <excludes>
                              <exclude>META-INF/*.SF</exclude>
                              <exclude>META-INF/*.DSA</exclude>
                              <exclude>META-INF/*.RSA</exclude>
                          </excludes>
                      </filter>
                  </filters>
              </configuration>
              <executions>
                  <execution>
                      <phase>package</phase>
                      <goals>
                          <goal>shade</goal>
                      </goals>
                  </execution>
              </executions>
          </plugin>
      </plugins>
</build>
```

Write Java code for the UDFs

Create a new class by extending UserDefinedFunctionHandler.java. Write your UDFs inside the class.

In the following example, two Java methods for UDFs, `compress()` and `decompress()`, are created inside the class `MyUserDefinedFunctions`.

```
*package *com.mycompany.athena.udfs;

public class MyUserDefinedFunctions
        extends UserDefinedFunctionHandler
{
    private static final String SOURCE_TYPE = "MyCompany";

    public MyUserDefinedFunctions()
    {
        super(SOURCE_TYPE);
    }
```

```
    /**
     * Compresses a valid UTF-8 String using the zlib compression library.
     * Encodes bytes with Base64 encoding scheme.
     *
     * @param input the String to be compressed
     * @return the compressed String
     */
    public String compress(String input)
    {
        byte[] inputBytes = input.getBytes(StandardCharsets.UTF_8);

        // create compressor
        Deflater compressor = new Deflater();
        compressor.setInput(inputBytes);
        compressor.finish();

        // compress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new 
 ByteArrayOutputStream(inputBytes.length);
        while (!compressor.finished()) {
            int bytes = compressor.deflate(buffer);
            byteArrayOutputStream.write(buffer, 0, bytes);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return encoded string
        byte[] compressedBytes = byteArrayOutputStream.toByteArray();
        return Base64.getEncoder().encodeToString(compressedBytes);
    }

    /**
     * Decompresses a valid String that has been compressed using the zlib compression 
 library.
     * Decodes bytes with Base64 decoding scheme.
     *
     * @param input the String to be decompressed
```

```
     * @return the decompressed String
     */
    public String decompress(String input)
    {
        byte[] inputBytes = Base64.getDecoder().decode((input));

        // create decompressor
        Inflater decompressor = new Inflater();
        decompressor.setInput(inputBytes, 0, inputBytes.length);

        // decompress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new
 ByteArrayOutputStream(inputBytes.length);
        try {
            while (!decompressor.finished()) {
                int bytes = decompressor.inflate(buffer);
                if (bytes == 0 && decompressor.needsInput()) {
                    throw new DataFormatException("Input is truncated");
                }
                byteArrayOutputStream.write(buffer, 0, bytes);
            }
        }
        catch (DataFormatException e) {
            throw new RuntimeException("Failed to decompress string", e);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return decoded string
        byte[] decompressedBytes = byteArrayOutputStream.toByteArray();
        return new String(decompressedBytes, StandardCharsets.UTF_8);
    }
}
```

Build the JAR file

Run `mvn clean install` to build your project. After it successfully builds, a JAR file is created in the `target` folder of your project named `artifactId-version.jar`, where `artifactId` is the name you provided in the Maven project, for example, `my-athena-udfs`.

Deploy the JAR to AWS Lambda

You have two options to deploy your code to Lambda:

- Deploy Using AWS Serverless Application Repository (Recommended)
- Create a Lambda Function from the JAR file

Option 1: Deploy to the AWS Serverless Application Repository

When you deploy your JAR file to the AWS Serverless Application Repository, you create an AWS SAM template YAML file that represents the architecture of your application. You then specify this YAML file and an Amazon S3 bucket where artifacts for your application are uploaded and made available to the AWS Serverless Application Repository. The procedure below uses the publish.sh script located in the `athena-query-federation/tools` directory of the Athena Query Federation SDK that you cloned earlier.

For more information and requirements, see Publishing applications in the *AWS Serverless Application Repository Developer Guide*, AWS SAM template concepts in the *AWS Serverless Application Model Developer Guide*, and Publishing serverless applications using the AWS SAM CLI.

The following example demonstrates parameters in a YAML file. Add similar parameters to your YAML file and save it in your project directory. See athena-udf.yaml in GitHub for a full example.

```
Transform: 'AWS::Serverless-2016-10-31'
Metadata:
  'AWS::ServerlessRepo::Application':
    Name: MyApplicationName
    Description: 'The description I write for my application'
    Author: 'Author Name'
    Labels:
      - athena-federation
    SemanticVersion: 1.0.0
Parameters:
  LambdaFunctionName:
```

```
    Description: 'The name of the Lambda function that will contain your UDFs.'
    Type: String
  LambdaTimeout:
    Description: 'Maximum Lambda invocation runtime in seconds. (min 1 - 900 max)'
    Default: 900
    Type: Number
  LambdaMemory:
    Description: 'Lambda memory in MB (min 128 - 3008 max).'
    Default: 3008
    Type: Number
Resources:
  ConnectorConfig:
    Type: 'AWS::Serverless::Function'
    Properties:
      FunctionName: !Ref LambdaFunctionName
      Handler: "full.path.to.your.handler. For example,
 com.amazonaws.athena.connectors.udfs.MyUDFHandler"
      CodeUri: "Relative path to your JAR file. For example, ./target/athena-
udfs-1.0.jar"
      Description: "My description of the UDFs that this Lambda function enables."
      Runtime: java8
      Timeout: !Ref LambdaTimeout
      MemorySize: !Ref LambdaMemory
```

Copy the `publish.sh` script to the project directory where you saved your YAML file, and run the following command:

```
./publish.sh MyS3Location MyYamlFile
```

For example, if your bucket location is `s3://amzn-s3-demo-bucket/mysarapps/athenaudf` and your YAML file was saved as `my-athena-udfs.yaml`:

```
./publish.sh amzn-s3-demo-bucket/mysarapps/athenaudf my-athena-udfs
```

To create a Lambda function

1. Open the Lambda console at https://console.aws.amazon.com/lambda/, choose **Create function**, and then choose **Browse serverless app repository**
2. Choose **Private applications**, find your application in the list, or search for it using key words, and select it.

3. Review and provide application details, and then choose **Deploy.**

   You can now use the method names defined in your Lambda function JAR file as UDFs in Athena.

Option 2: Create a Lambda function directly

You can also create a Lambda function directly using the console or AWS CLI. The following example demonstrates using the Lambda `create-function` CLI command.

```
aws lambda create-function \
 --function-name MyLambdaFunctionName \
 --runtime java8 \
 --role arn:aws:iam::1234567890123:role/my_lambda_role \
 --handler com.mycompany.athena.udfs.MyUserDefinedFunctions \
 --timeout 900 \
 --zip-file fileb://./target/my-athena-udfs-1.0-SNAPSHOT.jar
```

Query across regions

Athena supports the ability to query Amazon S3 data in an AWS Region that is different from the Region in which you are using Athena. Querying across Regions can be an option when moving the data is not practical or permissible, or if you want to query data across multiple regions. Even if Athena is not available in a particular Region, data from that Region can be queried from another Region in which Athena is available.

To query data in a Region, your account must be enabled in that Region even if the Amazon S3 data does not belong to your account. For some regions such as US East (Ohio), your access to the Region is automatically enabled when your account is created. Other Regions require that your account be "opted-in" to the Region before you can use it. For a list of Regions that require opt-in, see Available regions in the *Amazon EC2 User Guide*. For specific instructions about opting-in to a Region, see Managing AWS regions in the *Amazon Web Services General Reference*.

Considerations and limitations

- **Data access permissions** – To successfully query Amazon S3 data from Athena across Regions, your account must have permissions to read the data. If the data that you want to query belongs to another account, the other account must grant you access to the Amazon S3 location that contains the data.

- **Data transfer charges** – Amazon S3 data transfer charges apply for cross-region queries. Running a query can result in more data transferred than the size of the dataset. We recommend that you start by testing your queries on a subset of data and reviewing the costs in AWS Cost Explorer.
- **AWS Glue** – You can use AWS Glue across Regions. Additional charges may apply for cross-region AWS Glue traffic. For more information, see Create cross-account and cross-region AWS Glue connections in the *AWS Big Data Blog*.
- **Amazon S3 encryption options** – The SSE-S3 and SSE-KMS encryption options are supported for queries across Regions; CSE-KMS is not. For more information, see Supported Amazon S3 encryption options.
- **Federated queries** – Using federated queries across AWS Regions is not supported.
- **China Regions** – Cross-region queries are not supported in the China Regions.

Provided the above conditions are met, you can create an Athena table that points to the `LOCATION` value that you specify and query the data transparently. No special syntax is required. For information about creating Athena tables, see Create tables in Athena.

Query the AWS Glue Data Catalog

Because AWS Glue Data Catalog is used by many AWS services as their central metadata repository, you might want to query Data Catalog metadata. To do so, you can use SQL queries in Athena. You can use Athena to query AWS Glue catalog metadata like databases, tables, partitions, and columns.

To obtain AWS Glue Catalog metadata, you query the `information_schema` database on the Athena backend. The example queries in this topic show how to use Athena to query AWS Glue Catalog metadata for common use cases.

Considerations and limitations

- Instead of querying the `information_schema` database, it is possible to use individual Apache Hive DDL commands to extract metadata information for specific databases, tables, views, partitions, and columns from Athena. However, the output is in a non-tabular format.
- Querying `information_schema` is most performant if you have a small to moderate amount of AWS Glue metadata. If you have a large amount of metadata, errors can occur.
- You cannot use `CREATE VIEW` to create a view on the `information_schema` database.

Topics

List databases and searching a specified database

The examples in this section show how to list the databases in metadata by schema name.

Example – Listing databases

The following example query lists the databases from the `information_schema.schemata` table.

```
SELECT schema_name
FROM   information_schema.schemata
LIMIT  10;
```

The following table shows sample results.

6	alb-databas1
7	alb_original_cust
8	alblogsdatabase
9	athena_db_test
10	athena_ddl_db

Example – Searching a specified database

In the following example query, `rdspostgresql` is a sample database.

```
SELECT schema_name
```

```
FROM   information_schema.schemata
WHERE  schema_name = 'rdspostgresql'
```

The following table shows sample results.

	schema_name
1	rdspostgresql

List tables in a specified database and searching for a table by name

To list metadata for tables, you can query by table schema or by table name.

Example – Listing tables by schema

The following query lists tables that use the `rdspostgresql` table schema.

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
WHERE  table_schema = 'rdspostgresql'
```

The following table shows a sample result.

	table_schema	table_name	table_type
1	rdspostgresql	rdspostgresqldb1_public_account	BASE TABLE

Example – Searching for a table by name

The following query obtains metadata information for the table `athena1`.

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
```

```
WHERE  table_name = 'athena1'
```

The following table shows a sample result.

	table_schema	table_name	table_type
1	default	athena1	BASE TABLE

List partitions for a specific table

You can use SHOW PARTITIONS *table_name* to list the partitions for a specified table, as in the following example.

```
SHOW PARTITIONS cloudtrail_logs_test2
```

You can also use a $partitions metadata query to list the partition numbers and partition values for a specific table.

Example – Querying the partitions for a table using the $partitions syntax

The following example query lists the partitions for the table cloudtrail_logs_test2 using the $partitions syntax.

```
SELECT * FROM default."cloudtrail_logs_test2$partitions" ORDER BY partition_number
```

The following table shows sample results.

	table_catalog	table_schema	table_name	Year	Month	Day
1	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	10
2	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	11
3	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	12

List or search columns for a specified table or view

You can list all columns for a table, all columns for a view, or search for a column by name in a specified database and table.

To list the columns, use a `SELECT *` query. In the `FROM` clause, specify `information_schema.columns`. In the `WHERE` clause, use `table_schema='database_name'` to specify the database and `table_name = 'table_name'` to specify the table or view that has the columns that you want to list.

Example – Listing all columns for a specified table

The following example query lists all columns for the table `rdspostgresqldb1_public_account`.

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'rdspostgresql'
       AND table_name = 'rdspostgresqldb1_public_account'
```

The following table shows sample results.

	table_catalog	table_sc ema	table_name	column_ me	ordinal_position	column_c fault	is_null le	data_	comn	extra_info
1	awsdatacatalog	rdspostg esql	rdspostgresqldb1_public_account	passwor	1		YES	varcha		
2	awsdatacatalog	rdspostg esql	rdspostgresqldb1_public_account	user_id	2		YES	intege		
3	awsdatacatalog	rdspostg esql	rdspostgresqldb1_public_account	created_ n	3		YES	timest		

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
4	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	last_login	4		YES	timest		
5	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	email	5		YES	varcha		
6	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	usernam	6		YES	varcha		

Example – Listing the columns for a specified view

The following example query lists all the columns in the `default` database for the view `arrayview`.

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
```

The following table shows sample results.

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
1	awsdatacatalog	default	arrayvi	searchdate	1		YES	varchar		
2	awsdatacatalog	default	arrayvi	sid	2		YES	varchar		

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
3	awsdatacatalog	default	arrayvi	btid	3		YES	varchar		
4	awsdatacatalog	default	arrayvi	p	4		YES	varchar		
5	awsdatacatalog	default	arrayvi	infantprice	5		YES	varchar		
6	awsdatacatalog	default	arrayvi	sump	6		YES	varchar		
7	awsdatacatalog	default	arrayvi	journeymaparray	7		YES	array(varchar)		

Example – Searching for a column by name in a specified database and table

The following example query searches for metadata for the `sid` column in the `arrayview` view of the `default` database.

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
       AND column_name='sid'
```

The following table shows a sample result.

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
1	awsdatacatalog	default	arrayview	sid	2		YES	varcha		

List the columns that specific tables have in common

You can list the columns that specific tables in a database have in common.

- Use the syntax `SELECT column_name FROM information_schema.columns`.
- For the `WHERE` clause, use the syntax `WHERE table_name IN ('table1', 'table2')`.

Example – Listing common columns for two tables in the same database

The following example query lists the columns that the tables `table1` and `table2` have in common.

```
SELECT column_name
FROM information_schema.columns
WHERE table_name IN ('table1', 'table2')
GROUP BY column_name
HAVING COUNT(*) > 1;
```

List all columns for all tables

You can list all columns for all tables in `AwsDataCatalog` or for all tables in a specific database in `AwsDataCatalog`.

- To list all columns for all databases in `AwsDataCatalog`, use the query `SELECT * FROM information_schema.columns`.
- To restrict the results to a specific database, use `table_schema='`*`database_name`*`'` in the `WHERE` clause.

Example – Listing all columns for all tables in a specific database

The following example query lists all columns for all tables in the database `webdata`.

```
SELECT * FROM information_schema.columns WHERE table_schema = 'webdata'
```

Query AWS service logs

This section includes several procedures for using Amazon Athena to query popular datasets, such as AWS CloudTrail logs, Amazon CloudFront logs, Classic Load Balancer logs, Application Load Balancer logs, Amazon VPC flow logs, and Network Load Balancer logs.

The tasks in this section use the Athena console, but you can also use other tools like the Athena JDBC driver, the AWS CLI, or the Amazon Athena API Reference.

For information about using AWS CloudFormation to automatically create AWS service log tables, partitions, and example queries in Athena, see Automating AWS service logs table creation and querying them with Amazon Athena in the AWS Big Data Blog. For information about using a Python library for AWS Glue to create a common framework for processing AWS service logs and querying them in Athena, see Easily query AWS service logs using Amazon Athena.

The topics in this section assume that you have configured appropriate permissions to access Athena and the Amazon S3 bucket where the data to query should reside. For more information, see Set up, administrative, and programmatic access and Get started.

Topics

Query Application Load Balancer logs

An Application Load Balancer is a load balancing option for Elastic Load Balancing that enables traffic distribution in a microservices deployment using containers. Querying Application Load Balancer logs allows you to see the source of traffic, latency, and bytes transferred to and from Elastic Load Balancing instances and backend applications. For more information, see Access logs for your Application Load Balancer and Connection logs for your Application Load Balancer in the *User Guide for Application Load Balancers*.

Prerequisites

- Enable access logging or connection logging so that Application Load Balancer logs can be saved to your Amazon S3 bucket.
- A database to hold the table that you will create for Athena. To create a database, you can use the Athena or AWS Glue console. For more information, see Create databases in Athena in this guide or Working with databases on the AWS glue console in the *AWS Glue Developer Guide*.

Topics

- Create the table for ALB access logs
- Create the table for ALB access logs in Athena using partition projection
- Example queries for ALB access logs
- Create the table for ALB connection logs
- Create the table for ALB connection logs in Athena using partition projection
- Example queries for ALB connection logs
- Additional resources

Create the table for ALB access logs

1. Copy and paste the following `CREATE TABLE` statement into the query editor in the Athena console, and then modify it as necessary for your own log entry requirements. For information about getting started with the Athena console, see Get started. Replace the path in the `LOCATION` clause with your Amazon S3 access log folder location. For more information about access log file location, see Access log files in the *User Guide for Application Load Balancers*.

   For information about each log file field, see Access log entries in the *User Guide for Application Load Balancers*.

   > **Note**
   >
   > The following example `CREATE TABLE` statement includes the recently added `classification`, `classification_reason`, and `conn_trace_id` ('traceability ID', or TID) columns. To create a table for Application Load Balancer access logs that do

not contain these entries, remove the corresponding columns from the `CREATE TABLE` statement and modify the regular expression accordingly.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
            type string,
            time string,
            elb string,
            client_ip string,
            client_port int,
            target_ip string,
            target_port int,
            request_processing_time double,
            target_processing_time double,
            response_processing_time double,
            elb_status_code int,
            target_status_code string,
            received_bytes bigint,
            sent_bytes bigint,
            request_verb string,
            request_url string,
            request_proto string,
            user_agent string,
            ssl_cipher string,
            ssl_protocol string,
            target_group_arn string,
            trace_id string,
            domain_name string,
            chosen_cert_arn string,
            matched_rule_priority string,
            request_creation_time string,
            actions_executed string,
            redirect_url string,
            lambda_error_reason string,
            target_port_list string,
            target_status_code_list string,
            classification string,
            classification_reason string,
            conn_trace_id string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
```

```
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (-
 |[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^
\"]*)\" \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^\
\s]+?)\" \"([^\\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?'
            )
            LOCATION 's3://amzn-s3-demo-bucket/access-log-folder-path/'
```

2. Run the query in the Athena console. After the query completes, Athena registers the `alb_access_logs` table, making the data in it ready for you to issue queries.

Create the table for ALB access logs in Athena using partition projection

Because ALB access logs have a known structure whose partition scheme you can specify in advance, you can reduce query runtime and automate partition management by using the Athena partition projection feature. Partition projection automatically adds new partitions as new data is added. This removes the need for you to manually add partitions by using `ALTER TABLE ADD PARTITION`.

The following example `CREATE TABLE` statement automatically uses partition projection on ALB access logs from a specified date until the present for a single AWS region. The statement is based on the example in the previous section but adds `PARTITIONED BY` and `TBLPROPERTIES` clauses to enable partition projection. In the `LOCATION` and `storage.location.template` clauses, replace the placeholders with values that identify the Amazon S3 bucket location of your ALB access logs. For more information about access log file location, see Access log files in the *User Guide for Application Load Balancers*. For `projection.day.range`, replace *2022*/*01*/*01* with the starting date that you want to use. After you run the query successfully, you can query the table. You do not have to run `ALTER TABLE ADD PARTITION` to load the partitions. For information about each log file field, see Access log entries.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
            type string,
            time string,
            elb string,
            client_ip string,
            client_port int,
            target_ip string,
            target_port int,
```

```
            request_processing_time double,
            target_processing_time double,
            response_processing_time double,
            elb_status_code int,
            target_status_code string,
            received_bytes bigint,
            sent_bytes bigint,
            request_verb string,
            request_url string,
            request_proto string,
            user_agent string,
            ssl_cipher string,
            ssl_protocol string,
            target_group_arn string,
            trace_id string,
            domain_name string,
            chosen_cert_arn string,
            matched_rule_priority string,
            request_creation_time string,
            actions_executed string,
            redirect_url string,
            lambda_error_reason string,
            target_port_list string,
            target_status_code_list string,
            classification string,
            classification_reason string,
            conn_trace_id string
            )
            PARTITIONED BY
            (
             day STRING
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (- |
[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\"
 \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^\\s]+?)\"
 \"([^\\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?'
            )
            LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
```

```
            TBLPROPERTIES
            (
             "projection.enabled" = "true",
             "projection.day.type" = "date",
             "projection.day.range" = "2022/01/01,NOW",
             "projection.day.format" = "yyyy/MM/dd",
             "projection.day.interval" = "1",
             "projection.day.interval.unit" = "DAYS",
             "storage.location.template" = "s3://amzn-s3-demo-bucket/AWSLogs/<ACCOUNT-NUMBER>/elasticloadbalancing/<REGION>/${day}"
            )
```

For more information about partition projection, see Use partition projection with Amazon Athena.

Example queries for ALB access logs

The following query counts the number of HTTP GET requests received by the load balancer grouped by the client IP address:

```
SELECT COUNT(request_verb) AS
 count,
 request_verb,
 client_ip
FROM alb_access_logs
GROUP BY request_verb, client_ip
LIMIT 100;
```

Another query shows the URLs visited by Safari browser users:

```
SELECT request_url
FROM alb_access_logs
WHERE user_agent LIKE '%Safari%'
LIMIT 10;
```

The following query shows records that have ELB status code values greater than or equal to 500.

```
SELECT * FROM alb_access_logs
WHERE elb_status_code >= 500
```

The following example shows how to parse the logs by `datetime`:

```
SELECT client_ip, sum(received_bytes)
FROM alb_access_logs
WHERE parse_datetime(time,'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
     BETWEEN parse_datetime('2018-05-30-12:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2018-05-31-00:00:00','yyyy-MM-dd-HH:mm:ss')
GROUP BY client_ip;
```

The following query queries the table that uses partition projection for all ALB access logs from the specified day.

```
SELECT *
FROM alb_access_logs
WHERE day = '2022/02/12'
```

Create the table for ALB connection logs

1. Copy and paste the following example `CREATE TABLE` statement into the query editor in the Athena console, and then modify it as necessary for your own log entry requirements. For information about getting started with the Athena console, see [Get started](). Replace the path in the `LOCATION` clause with your Amazon S3 connection log folder location. For more information about connection log file location, see [Connection log files]() in the *User Guide for Application Load Balancers*. For information about each log file field, see [Connection log entries]().

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
         time string,
         client_ip string,
         client_port int,
         listener_port int,
         tls_protocol string,
         tls_cipher string,
         tls_handshake_latency double,
         leaf_client_cert_subject string,
         leaf_client_cert_validity string,
         leaf_client_cert_serial_number string,
         tls_verify_status string,
         conn_trace_id string
         )
         ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
         WITH SERDEPROPERTIES (
         'serialization.format' = '1',
         'input.regex' =
```

```
        '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
\"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?'
        )
        LOCATION 's3://amzn-s3-demo-bucket/connection-log-folder-path/'
```

2. Run the query in the Athena console. After the query completes, Athena registers the `alb_connection_logs` table, making the data in it ready for you to issue queries.

Create the table for ALB connection logs in Athena using partition projection

Because ALB connection logs have a known structure whose partition scheme you can specify in advance, you can reduce query runtime and automate partition management by using the Athena partition projection feature. Partition projection automatically adds new partitions as new data is added. This removes the need for you to manually add partitions by using `ALTER TABLE ADD PARTITION`.

The following example `CREATE TABLE` statement automatically uses partition projection on ALB connection logs from a specified date until the present for a single AWS region. The statement is based on the example in the previous section but adds `PARTITIONED BY` and `TBLPROPERTIES` clauses to enable partition projection. In the `LOCATION` and `storage.location.template` clauses, replace the placeholders with values that identify the Amazon S3 bucket location of your ALB connection logs. For more information about connection log file location, see [Connection log files](#) in the *User Guide for Application Load Balancers*. For `projection.day.range`, replace *`2023`*/*`01`*/*`01`* with the starting date that you want to use. After you run the query successfully, you can query the table. You do not have to run `ALTER TABLE ADD PARTITION` to load the partitions. For information about each log file field, see [Connection log entries](#).

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
         time string,
         client_ip string,
         client_port int,
         listener_port int,
         tls_protocol string,
         tls_cipher string,
         tls_handshake_latency double,
         leaf_client_cert_subject string,
         leaf_client_cert_validity string,
         leaf_client_cert_serial_number string,
         tls_verify_status string,
         conn_trace_id string
```

```
          )
             PARTITIONED BY
             (
              day STRING
             )
             ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
             WITH SERDEPROPERTIES (
             'serialization.format' = '1',
             'input.regex' =
              '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
 \"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?'
             )
             LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
             TBLPROPERTIES
             (
              "projection.enabled" = "true",
              "projection.day.type" = "date",
              "projection.day.range" = "2023/01/01,NOW",
              "projection.day.format" = "yyyy/MM/dd",
              "projection.day.interval" = "1",
              "projection.day.interval.unit" = "DAYS",
              "storage.location.template" = "s3://amzn-s3-demo-bucket/AWSLogs/<ACCOUNT-
NUMBER>/elasticloadbalancing/<REGION>/${day}"
             )
```

For more information about partition projection, see Use partition projection with Amazon Athena.

Example queries for ALB connection logs

The following query count occurrences where the value for `tls_verify_status` was not `'Success'`, grouped by client IP address:

```
SELECT DISTINCT client_ip, count() AS count FROM alb_connection_logs
WHERE tls_verify_status != 'Success'
GROUP BY client_ip
ORDER BY count() DESC;
```

The following query searches occurrences where the value for `tls_handshake_latency` was over 2 seconds in the specified time range:

```
SELECT * FROM alb_connection_logs
```

```
WHERE
  (
    parse_datetime(time, 'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
    BETWEEN
    parse_datetime('2024-01-01-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
    AND
    parse_datetime('2024-03-20-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
  )
  AND
    (tls_handshake_latency >= 2.0);
```

Additional resources

For more information about using ALB logs, see the following resources.

- How do I analyze my Application Load Balancer access logs using Amazon Athena in the *AWS Knowledge Center*.
- For information about HTTP status codes in Elastic Load Balancing, see Troubleshoot your application load balancers in the *User Guide for Application Load Balancers*.
- Catalog and analyze Application Load Balancer logs more efficiently with AWS Glue custom classifiers and Amazon Athena in the *AWS Big Data Blog*.

Query Classic Load Balancer logs

Use Classic Load Balancer logs to analyze and understand traffic patterns to and from Elastic Load Balancing instances and backend applications. You can see the source of traffic, latency, and bytes that have been transferred.

Before you analyze the Elastic Load Balancing logs, configure them for saving in the destination Amazon S3 bucket. For more information, see Enable access logs for your Classic Load Balancer.

To create the table for Elastic Load Balancing logs

1. Copy and paste the following DDL statement into the Athena console. Check the syntax of the Elastic Load Balancing log records. You may need to update the following query to include the columns and the Regex syntax for latest version of the record.

```
CREATE EXTERNAL TABLE IF NOT EXISTS elb_logs (

 timestamp string,
```

```
 elb_name string,
 request_ip string,
 request_port int,
 backend_ip string,
 backend_port int,
 request_processing_time double,
 backend_processing_time double,
 client_response_time double,
 elb_response_code string,
 backend_response_code string,
 received_bytes bigint,
 sent_bytes bigint,
 request_verb string,
 url string,
 protocol string,
 user_agent string,
 ssl_cipher string,
 ssl_protocol string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
 'serialization.format' = '1',
 'input.regex' = '([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \\\"([^ ]*)
 ([^ ]*) (- |[^ ]*)\\\" (\"[^\"]*\") ([A-Z0-9-]+) ([A-Za-z0-9.-]*)$'
)
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/AWS_account_ID/elasticloadbalancing/';
```

2. Modify the `LOCATION` Amazon S3 bucket to specify the destination of your Elastic Load Balancing logs.
3. Run the query in the Athena console. After the query completes, Athena registers the `elb_logs` table, making the data in it ready for queries. For more information, see Example queries.

Example queries

Use a query similar to the following example. It lists the backend application servers that returned a `4XX` or `5XX` error response code. Use the `LIMIT` operator to limit the number of logs to query at a time.

```
SELECT
 timestamp,
```

```
 elb_name,
 backend_ip,
 backend_response_code
FROM elb_logs
WHERE backend_response_code LIKE '4%' OR
      backend_response_code LIKE '5%'
LIMIT 100;
```

Use a subsequent query to sum up the response time of all the transactions grouped by the backend IP address and Elastic Load Balancing instance name.

```
SELECT sum(backend_processing_time) AS
 total_ms,
 elb_name,
 backend_ip
FROM elb_logs WHERE backend_ip <> ''
GROUP BY backend_ip, elb_name
LIMIT 100;
```

For more information, see [Analyzing data in S3 using Athena](#).

Query Amazon CloudFront logs

You can configure Amazon CloudFront CDN to export Web distribution access logs to Amazon Simple Storage Service. Use these logs to explore users' surfing patterns across your web properties served by CloudFront.

Before you begin querying the logs, enable Web distributions access log on your preferred CloudFront distribution. For information, see Access logs in the *Amazon CloudFront Developer Guide*. Make a note of the Amazon S3 bucket in which you save these logs.

Topics

- Create a table for CloudFront standard logs (legacy)
- Create a table for CloudFront logs in Athena using manual partitioning with JSON
- Create a table for CloudFront logs in Athena using manual partitioning with Parquet
- Create a table for CloudFront logs in Athena using partition projection with JSON
- Create a table for CloudFront logs in Athena using partition projection with Parquet
- Create a table for CloudFront real-time logs
- Additional resources

Create a table for CloudFront standard logs (legacy)

> **Note**
>
> The following procedure works for the Web distribution access logs in CloudFront. It does not apply to streaming logs from RTMP distributions.

To create a table for CloudFront standard log file fields

1. Copy and paste the following example DDL statement into the Query Editor in the Athena console. The example statement uses the log file fields documented in the Standard log file fields section of the *Amazon CloudFront Developer Guide*. Modify the `LOCATION` for the Amazon S3 bucket that stores your logs. For information about using the Query Editor, see Get started.

   This query specifies `ROW FORMAT DELIMITED` and `FIELDS TERMINATED BY '\t'` to indicate that the fields are delimited by tab characters. For `ROW FORMAT DELIMITED`, Athena uses the LazySimpleSerDe by default. The column `date` is escaped using backticks (`) because it is a reserved word in Athena. For information, see Escape reserved keywords in queries.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_standard_logs (
  `date` DATE,
  time STRING,
  x_edge_location STRING,
  sc_bytes BIGINT,
  c_ip STRING,
  cs_method STRING,
  cs_host STRING,
  cs_uri_stem STRING,
  sc_status INT,
  cs_referrer STRING,
  cs_user_agent STRING,
  cs_uri_query STRING,
  cs_cookie STRING,
  x_edge_result_type STRING,
  x_edge_request_id STRING,
  x_host_header STRING,
  cs_protocol STRING,
  cs_bytes BIGINT,
  time_taken FLOAT,
```

```
  x_forwarded_for STRING,
  ssl_protocol STRING,
  ssl_cipher STRING,
  x_edge_response_result_type STRING,
  cs_protocol_version STRING,
  fle_status STRING,
  fle_encrypted_fields INT,
  c_port INT,
  time_to_first_byte FLOAT,
  x_edge_detailed_result_type STRING,
  sc_content_type STRING,
  sc_content_len BIGINT,
  sc_range_start BIGINT,
  sc_range_end BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://amzn-s3-demo-bucket/'
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Run the query in Athena console. After the query completes, Athena registers the `cloudfront_standard_logs` table, making the data in it ready for you to issue queries.

Example queries

The following query adds up the number of bytes served by CloudFront between June 9 and June 11, 2018. Surround the date column name with double quotes because it is a reserved word.

```
SELECT SUM(bytes) AS total_bytes
FROM cloudfront_standard_logs
WHERE "date" BETWEEN DATE '2018-06-09' AND DATE '2018-06-11'
LIMIT 100;
```

To eliminate duplicate rows (for example, duplicate empty rows) from the query results, you can use the `SELECT DISTINCT` statement, as in the following example.

```
SELECT DISTINCT *
FROM cloudfront_standard_logs
LIMIT 10;
```

Create a table for CloudFront logs in Athena using manual partitioning with JSON

To create a table for CloudFront standard log file fields using a JSON format

1. Copy and paste the following example DDL statement into the Query Editor in the Athena console. The example statement uses the log file fields documented in the [Standard log file fields](#) section of the *Amazon CloudFront Developer Guide*. Modify the `LOCATION` for the Amazon S3 bucket that stores your logs.

   This query uses OpenX JSON SerDe with the following SerDe properties to read JSON fields correctly in Athena.

```
CREATE EXTERNAL TABLE `cf_logs_manual_partition_json`(
  `date` string ,
  `time` string ,
  `x-edge-location` string ,
  `sc-bytes` string ,
  `c-ip` string ,
  `cs-method` string ,
  `cs(host)` string ,
  `cs-uri-stem` string ,
  `sc-status` string ,
  `cs(referer)` string ,
  `cs(user-agent)` string ,
  `cs-uri-query` string ,
  `cs(cookie)` string ,
  `x-edge-result-type` string ,
  `x-edge-request-id` string ,
  `x-host-header` string ,
  `cs-protocol` string ,
  `cs-bytes` string ,
  `time-taken` string ,
  `x-forwarded-for` string ,
  `ssl-protocol` string ,
  `ssl-cipher` string ,
  `x-edge-response-result-type` string ,
  `cs-protocol-version` string ,
  `fle-status` string ,
  `fle-encrypted-fields` string ,
  `c-port` string ,
  `time-to-first-byte` string ,
  `x-edge-detailed-result-type` string ,
  `sc-content-type` string ,
```

```
  `sc-content-len` string ,
  `sc-range-start` string ,
  `sc-range-end` string )
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'paths'='c-ip,c-port,cs(Cookie),cs(Host),cs(Referer),cs(User-Agent),cs-bytes,cs-method,cs-protocol,cs-protocol-version,cs-uri-query,cs-uri-stem,date,fle-encrypted-fields,fle-status,sc-bytes,sc-content-len,sc-content-type,sc-range-end,sc-range-start,sc-status,ssl-cipher,ssl-protocol,time,time-taken,time-to-first-byte,x-edge-detailed-result-type,x-edge-location,x-edge-request-id,x-edge-response-result-type,x-edge-result-type,x-forwarded-for,x-host-header')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/'
```

2. Run the query in Athena console. After the query completes, Athena registers the `cf_logs_manual_partition_json` table, making the data in it ready for you to issue queries.

Example queries

The following query adds up the number of bytes served by CloudFront for January 15, 2025.

```
SELECT sum(cast("sc-bytes" as BIGINT)) as sc
FROM cf_logs_manual_partition_json
WHERE "date"='2025-01-15'
```

To eliminate duplicate rows (for example, duplicate empty rows) from the query results, you can use the `SELECT DISTINCT` statement, as in the following example.

```
SELECT DISTINCT * FROM cf_logs_manual_partition_json
```

Create a table for CloudFront logs in Athena using manual partitioning with Parquet

To create a table for CloudFront standard log file fields using a Parquet format

1. Copy and paste the following example DDL statement into the Query Editor in the Athena console. The example statement uses the log file fields documented in the Standard log file fields section of the *Amazon CloudFront Developer Guide*.

   This query uses ParquetHiveSerDe with the following SerDe properties to read Parquet fields correctly in Athena.

```
CREATE EXTERNAL TABLE `cf_logs_manual_partition_parquet`(
  `date` string,
  `time` string,
  `x_edge_location` string,
  `sc_bytes` string,
  `c_ip` string,
  `cs_method` string,
  `cs_host` string,
  `cs_uri_stem` string,
  `sc_status` string,
  `cs_referer` string,
  `cs_user_agent` string,
  `cs_uri_query` string,
  `cs_cookie` string,
  `x_edge_result_type` string,
  `x_edge_request_id` string,
  `x_host_header` string,
  `cs_protocol` string,
  `cs_bytes` string,
  `time_taken` string,
  `x_forwarded_for` string,
  `ssl_protocol` string,
  `ssl_cipher` string,
  `x_edge_response_result_type` string,
  `cs_protocol_version` string,
  `fle_status` string,
  `fle_encrypted_fields` string,
  `c_port` string,
  `time_to_first_byte` string,
  `x_edge_detailed_result_type` string,
  `sc_content_type` string,
  `sc_content_len` string,
```

```
  `sc_range_start` string,
  `sc_range_end` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/'
```

2. Run the query in Athena console. After the query completes, Athena registers the `cf_logs_manual_partition_parquet` table, making the data in it ready for you to issue queries.

Example queries

The following query adds up the number of bytes served by CloudFront for January 19, 2025.

```
SELECT sum(cast("sc_bytes" as BIGINT)) as sc
FROM cf_logs_manual_partition_parquet
WHERE "date"='2025-01-19'
```

To eliminate duplicate rows (for example, duplicate empty rows) from the query results, you can use the `SELECT DISTINCT` statement, as in the following example.

```
SELECT DISTINCT * FROM cf_logs_manual_partition_parquet
```

Create a table for CloudFront logs in Athena using partition projection with JSON

You can reduce query runtime and automate partition management with Athena partition projection feature. Partition projection automatically adds new partitions as new data is added. This removes the need for you to manually add partitions by using `ALTER TABLE ADD PARTITION`.

The following example CREATE TABLE statement automatically uses partition projection on CloudFront logs from a specified CloudFront distribution until present for a single AWS Region. After you run the query successfully, you can query the table.

```
CREATE EXTERNAL TABLE `cloudfront_logs_pp`(
  `date` string,
```

```
  `time` string,
  `x-edge-location` string,
  `sc-bytes` string,
  `c-ip` string,
  `cs-method` string,
  `cs(host)` string,
  `cs-uri-stem` string,
  `sc-status` string,
  `cs(referer)` string,
  `cs(user-agent)` string,
  `cs-uri-query` string,
  `cs(cookie)` string,
  `x-edge-result-type` string,
  `x-edge-request-id` string,
  `x-host-header` string,
  `cs-protocol` string,
  `cs-bytes` string,
  `time-taken` string,
  `x-forwarded-for` string,
  `ssl-protocol` string,
  `ssl-cipher` string,
  `x-edge-response-result-type` string,
  `cs-protocol-version` string,
  `fle-status` string,
  `fle-encrypted-fields` string,
  `c-port` string,
  `time-to-first-byte` string,
  `x-edge-detailed-result-type` string,
  `sc-content-type` string,
  `sc-content-len` string,
  `sc-range-start` string,
  `sc-range-end` string)
  PARTITIONED BY(
         distributionid string,
         year int,
         month int,
         day int,
         hour int )
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'paths'='c-ip,c-port,cs(Cookie),cs(Host),cs(Referer),cs(User-Agent),cs-bytes,cs-
method,cs-protocol,cs-protocol-version,cs-uri-query,cs-uri-stem,date,fle-encrypted-
fields,fle-status,sc-bytes,sc-content-len,sc-content-type,sc-range-end,sc-range-
```

```
start,sc-status,ssl-cipher,ssl-protocol,time,time-taken,time-to-first-byte,x-edge-
detailed-result-type,x-edge-location,x-edge-request-id,x-edge-response-result-type,x-
edge-result-type,x-forwarded-for,x-host-header')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/CloudFront/'
TBLPROPERTIES (
  'projection.distributionid.type'='enum',
  'projection.distributionid.values'='E2Oxxxxxxxxxxx',
  'projection.day.range'='01,31',
  'projection.day.type'='integer',
  'projection.day.digits'='2',
  'projection.enabled'='true',
  'projection.month.range'='01,12',
  'projection.month.type'='integer',
  'projection.month.digits'='2',
  'projection.year.range'='2025,2026',
  'projection.year.type'='integer',
  'projection.hour.range'='01,12',
  'projection.hour.type'='integer',
  'projection.hour.digits'='2',
  'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/
CloudFront/${distributionid}/${year}/${month}/${day}/${hour}/')
```

Following are some considerations for the properties used in the previous example.

- **Table name** – The table name *cloudfront_logs_pp* is replaceable. You can change it to any name that you prefer.
- **Location** – Modify `s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/` to point to your Amazon S3 bucket.
- **Distribution IDs** – For `projection.distributionid.values`, you can specify multiple distribution IDs if you separate them with commas. For example, *<distributionID1>*, *<distributionID2>*.
- **Year range** – In `projection.year.range`, you can define the range of years based on your data. For example, you can adjust it to any period, such as *2025*, *2026*.

> **Note**
>
> Including empty partitions, such as those for future dates (example: 2025-2040), can impact query performance. However, partition projection is designed to effectively handle future dates. To maintain optimal performance, ensure that partitions are managed thoughtfully and avoid excessive empty partitions when possible.

- **Storage location template –** You must ensure to update the `storage.location.template` correctly based on the following CloudFront partitioning structure and S3 path.

Parameter	Pattern
CloudFront partitioning structure	`AWSLogs/{ AWS_ACCOUNT_ID }/CloudFront/{ DistributionId }/folder2/{yyyy}/{MM}/{dd}/{HH}/folder3`
S3 path	`s3://amzn-s3-demo-bucket /AWSLogs/ AWS_ACCOUNT_ID /CloudFront/E2Oxxxxxxxxxx/folder2/2025/01/25/03/folder3/`

After you confirm that the CloudFront partitioning structure and S3 structure match the required patterns, update the `storage.location.template` as follows:

```
'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/account_id/CloudFront/
${distributionid}/folder2/${year}/${month}/${day}/${hour}/folder3/'
```

> **Note**
>
> Proper configuration of the `storage.location.template` is crucial for ensuring correct data storage and retrieval.

Create a table for CloudFront logs in Athena using partition projection with Parquet

The following example CREATE TABLE statement automatically uses partition projection on CloudFront logs in Parquet, from a specified CloudFront distribution until present for a single AWS Region. After you run the query successfully, you can query the table.

```
CREATE EXTERNAL TABLE `cloudfront_logs_parquet_pp`(
`date` string,
`time` string,
`x_edge_location` string,
`sc_bytes` string,
`c_ip` string,
`cs_method` string,
`cs_host` string,
`cs_uri_stem` string,
`sc_status` string,
`cs_referer` string,
`cs_user_agent` string,
`cs_uri_query` string,
`cs_cookie` string,
`x_edge_result_type` string,
`x_edge_request_id` string,
`x_host_header` string,
`cs_protocol` string,
`cs_bytes` string,
`time_taken` string,
`x_forwarded_for` string,
`ssl_protocol` string,
`ssl_cipher` string,
`x_edge_response_result_type` string,
`cs_protocol_version` string,
`fle_status` string,
`fle_encrypted_fields` string,
`c_port` string,
`time_to_first_byte` string,
`x_edge_detailed_result_type` string,
`sc_content_type` string,
`sc_content_len` string,
`sc_range_start` string,
`sc_range_end` string)
PARTITIONED BY(
 distributionid string,
 year int,
```

```
 month int,
 day int,
 hour int )
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
's3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/CloudFront/'
TBLPROPERTIES (
'projection.distributionid.type'='enum',
'projection.distributionid.values'='E3OK0LPUNWWO3',
'projection.day.range'='01,31',
'projection.day.type'='integer',
'projection.day.digits'='2',
'projection.enabled'='true',
'projection.month.range'='01,12',
'projection.month.type'='integer',
'projection.month.digits'='2',
'projection.year.range'='2019,2025',
'projection.year.type'='integer',
'projection.hour.range'='01,12',
'projection.hour.type'='integer',
'projection.hour.digits'='2',
'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/
CloudFront/${distributionid}/${year}/${month}/${day}/${hour}/')
```

Following are some considerations for the properties used in the previous example.

- **Table name** – The table name *`cloudfront_logs_pp`* is replaceable. You can change it to any name that you prefer.
- **Location** – Modify `s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_ID/` to point to your Amazon S3 bucket.
- **Distribution IDs** – For `projection.distributionid.values`, you can specify multiple distribution IDs if you separate them with commas. For example, *`<distributionID1>`*, *`<distributionID2>`*.
- **Year range** – In `projection.year.range`, you can define the range of years based on your data. For example, you can adjust it to any period, such as *2025*, *2026*.

Note

Including empty partitions, such as those for future dates (example: 2025-2040), can impact query performance. However, partition projection is designed to effectively handle future dates. To maintain optimal performance, ensure that partitions are managed thoughtfully and avoid excessive empty partitions when possible.

- **Storage location template** – You must ensure to update the `storage.location.template` correctly based on the following CloudFront partitioning structure and S3 path.

Parameter	Pattern
CloudFront partitioning structure	`AWSLogs/{ AWS_ACCOUNT_ID }/CloudFront/{ DistributionId }/folder2/{yyyy}/{MM}/{dd}/{HH}/folder3`
S3 path	`s3://amzn-s3-demo-bucket /AWSLogs/ AWS_ACCOUNT_ID /CloudFront/E2Oxxxxxxxxxx/folder2/2025/01/25/03/folder3/`

After you confirm that the CloudFront partitioning structure and S3 structure match the required patterns, update the `storage.location.template` as follows:

```
'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/account_id/CloudFront/
${distributionid}/folder2/${year}/${month}/${day}/${hour}/folder3/'
```

Note

Proper configuration of the `storage.location.template` is crucial for ensuring correct data storage and retrieval.

Create a table for CloudFront real-time logs

To create a table for CloudFront real-time log file fields

1. Copy and paste the following example DDL statement into the Query Editor in the Athena console. The example statement uses the log file fields documented in the [Real-time logs](#) section of the *Amazon CloudFront Developer Guide*. Modify the `LOCATION` for the Amazon S3 bucket that stores your logs. For information about using the Query Editor, see [Get started](#).

   This query specifies `ROW FORMAT DELIMITED` and `FIELDS TERMINATED BY '\t'` to indicate that the fields are delimited by tab characters. For `ROW FORMAT DELIMITED`, Athena uses the [LazySimpleSerDe](#) by default. The column `timestamp` is escaped using backticks (`) because it is a reserved word in Athena. For information, see [Escape reserved keywords in queries](#).

   The follow example contains all of the available fields. You can comment out or remove fields that you do not require.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_real_time_logs (
`timestamp` STRING,
c_ip STRING,
time_to_first_byte BIGINT,
sc_status BIGINT,
sc_bytes BIGINT,
cs_method STRING,
cs_protocol STRING,
cs_host STRING,
cs_uri_stem STRING,
cs_bytes BIGINT,
x_edge_location STRING,
x_edge_request_id STRING,
x_host_header STRING,
time_taken BIGINT,
cs_protocol_version STRING,
c_ip_version STRING,
cs_user_agent STRING,
cs_referer STRING,
cs_cookie STRING,
cs_uri_query STRING,
x_edge_response_result_type STRING,
x_forwarded_for STRING,
ssl_protocol STRING,
```

```
ssl_cipher STRING,
x_edge_result_type STRING,
fle_encrypted_fields STRING,
fle_status STRING,
sc_content_type STRING,
sc_content_len BIGINT,
sc_range_start STRING,
sc_range_end STRING,
c_port BIGINT,
x_edge_detailed_result_type STRING,
c_country STRING,
cs_accept_encoding STRING,
cs_accept STRING,
cache_behavior_path_pattern STRING,
cs_headers STRING,
cs_header_names STRING,
cs_headers_count BIGINT,
primary_distribution_id STRING,
primary_distribution_dns_name STRING,
origin_fbl STRING,
origin_lbl STRING,
asn STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://amzn-s3-demo-bucket/'
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Run the query in Athena console. After the query completes, Athena registers the `cloudfront_real_time_logs` table, making the data in it ready for you to issue queries.

Additional resources

For more information about using Athena to query CloudFront logs, see the following posts from the AWS big data blog.

Easily query AWS service logs using Amazon Athena (May 29, 2019).

Analyze your Amazon CloudFront access logs at scale (December 21, 2018).

Build a serverless architecture to analyze Amazon CloudFront access logs using AWS Lambda, Amazon Athena, and Amazon Managed Service for Apache Flink (May 26, 2017).

Query AWS CloudTrail logs

AWS CloudTrail is a service that records AWS API calls and events for Amazon Web Services accounts.

CloudTrail logs include details about any API calls made to your AWS services, including the console. CloudTrail generates encrypted log files and stores them in Amazon S3. For more information, see the AWS CloudTrail User Guide.

> **Note**
>
> If you want to perform SQL queries on CloudTrail event information across accounts, regions, and dates, consider using CloudTrail Lake. CloudTrail Lake is an AWS alternative to creating trails that aggregates information from an enterprise into a single, searchable event data store. Instead of using Amazon S3 bucket storage, it stores events in a data lake, which allows richer, faster queries. You can use it to create SQL queries that search events across organizations, regions, and within custom time ranges. Because you perform CloudTrail Lake queries within the CloudTrail console itself, using CloudTrail Lake does not require Athena. For more information, see the CloudTrail Lake documentation.

Using Athena with CloudTrail logs is a powerful way to enhance your analysis of AWS service activity. For example, you can use queries to identify trends and further isolate activity by attributes, such as source IP address or user.

A common application is to use CloudTrail logs to analyze operational activity for security and compliance. For information about a detailed example, see the AWS Big Data Blog post, Analyze security, compliance, and operational activity using AWS CloudTrail and Amazon Athena.

You can use Athena to query these log files directly from Amazon S3, specifying the `LOCATION` of log files. You can do this one of two ways:

- By creating tables for CloudTrail log files directly from the CloudTrail console.
- By manually creating tables for CloudTrail log files in the Athena console.

Topics

- Create a table for CloudTrail logs in Athena using manual partitioning
- Create a table for an organization wide trail using manual partitioning
- Create the table for CloudTrail logs in Athena using partition projection
- Example CloudTrail log queries

Understand CloudTrail logs and Athena tables

Before you begin creating tables, you should understand a little more about CloudTrail and how it stores data. This can help you create the tables that you need, whether you create them from the CloudTrail console or from Athena.

CloudTrail saves logs as JSON text files in compressed gzip format (`*.json.gz`). The location of the log files depends on how you set up trails, the AWS Region or Regions in which you are logging, and other factors.

For more information about where logs are stored, the JSON structure, and the record file contents, see the following topics in the AWS CloudTrail User Guide:

- Finding your CloudTrail log files
- CloudTrail Log File examples
- CloudTrail record contents
- CloudTrail event reference

To collect logs and save them to Amazon S3, enable CloudTrail from the AWS Management Console. For more information, see Creating a trail in the *AWS CloudTrail User Guide*.

Use the CloudTrail console to create an Athena table for CloudTrail logs

You can create a non-partitioned Athena table for querying CloudTrail logs directly from the CloudTrail console. Creating an Athena table from the CloudTrail console requires that you be logged in with a role that has sufficient permissions to create tables in Athena.

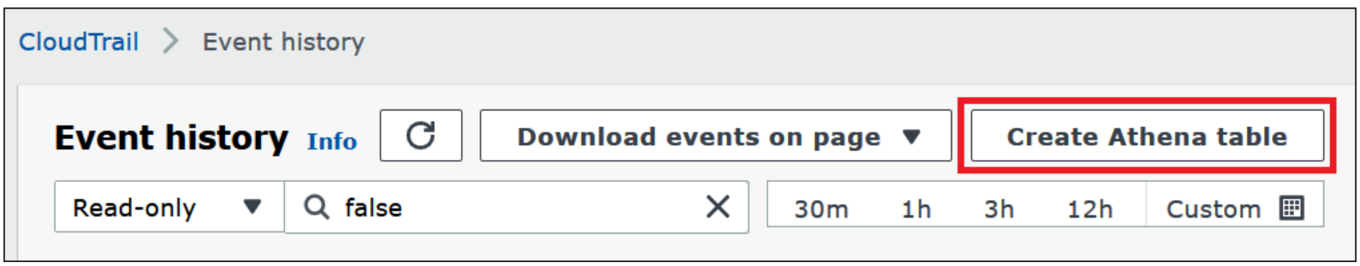
> **Note**
>
> You cannot use the CloudTrail console to create an Athena table for organization trail logs. Instead, create the table manually using the Athena console so that you can specify the

correct storage location. For information about organization trails, see Creating a trail for an organization in the *AWS CloudTrail User Guide*.

- For information about setting up permissions for Athena, see Set up, administrative, and programmatic access.
- For information about creating a table with partitions, see Create a table for CloudTrail logs in Athena using manual partitioning.

To create an Athena table for a CloudTrail trail using the CloudTrail console

1. Open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. In the navigation pane, choose **Event history**.
3. Choose **Create Athena table**.



4. For **Storage location**, use the down arrow to select the Amazon S3 bucket where log files are stored for the trail to query.

> **Note**
>
> To find the name of the bucket that is associated with a trail, choose **Trails** in the CloudTrail navigation pane and view the trail's **S3 bucket** column. To see the Amazon S3 location for the bucket, choose the link for the bucket in the **S3 bucket** column. This opens the Amazon S3 console to the CloudTrail bucket location.

5. Choose **Create table**. The table is created with a default name that includes the name of the Amazon S3 bucket.

Create a table for CloudTrail logs in Athena using manual partitioning

You can manually create tables for CloudTrail log files in the Athena console, and then run queries in Athena.

To create an Athena table for a CloudTrail trail using the Athena console

1. Copy and paste the following DDL statement into the Athena console query editor, then modify it according to your requirements. Note that because CloudTrail log files are not an ordered stack trace of public API calls, the fields in the log files do not appear in any specific order.

```
CREATE EXTERNAL TABLE cloudtrail_logs (
eventversion STRING,
useridentity STRUCT<
               type:STRING,
               principalid:STRING,
               arn:STRING,
               accountid:STRING,
               invokedby:STRING,
               accesskeyid:STRING,
               username:STRING,
               onbehalfof: STRUCT<
                    userid: STRING,
                    identitystorearn: STRING>,
  sessioncontext:STRUCT<
    attributes:STRUCT<
               mfaauthenticated:STRING,
               creationdate:STRING>,
    sessionissuer:STRUCT<
               type:STRING,
               principalid:STRING,
               arn:STRING,
               accountid:STRING,
               username:STRING>,
    ec2roledelivery:string,
    webidfederationdata: STRUCT<
               federatedprovider: STRING,
               attributes: map<string,string>>
  >
>,
eventtime STRING,
eventsource STRING,
eventname STRING,
awsregion STRING,
sourceipaddress STRING,
useragent STRING,
errorcode STRING,
```

```
errormessage STRING,
requestparameters STRING,
responseelements STRING,
additionaleventdata STRING,
requestid STRING,
eventid STRING,
resources ARRAY<STRUCT<
               arn:STRING,
               accountid:STRING,
               type:STRING>>,
eventtype STRING,
apiversion STRING,
readonly STRING,
recipientaccountid STRING,
serviceeventdetails STRING,
sharedeventid STRING,
vpcendpointid STRING,
vpcendpointaccountid STRING,
eventcategory STRING,
addendum STRUCT<
  reason:STRING,
  updatedfields:STRING,
  originalrequestid:STRING,
  originaleventid:STRING>,
sessioncredentialfromconsole STRING,
edgedevicedetails STRING,
tlsdetails STRUCT<
  tlsversion:STRING,
  ciphersuite:STRING,
  clientprovidedhostheader:STRING>
)
PARTITIONED BY (region string, year string, month string, day string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/Account_ID/';
```

Note

We suggest using the `org.apache.hive.hcatalog.data.JsonSerDe` shown in the example. Although a `com.amazon.emr.hive.serde.CloudTrailSerde` exists, it does not currently handle some of the newer CloudTrail fields.

2. (Optional) Remove any fields not required for your table. If you need to read only a certain set of columns, your table definition can exclude the other columns.

3. Modify `s3://amzn-s3-demo-bucket/AWSLogs/Account_ID/` to point to the Amazon S3 bucket that contains the log data that you want to query. The example uses a `LOCATION` value of logs for a particular account, but you can use the degree of specificity that suits your application. For example:

   - To analyze data from multiple accounts, you can roll back the `LOCATION` specifier to indicate all `AWSLogs` by using `LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/'`.
   - To analyze data from a specific date, account, and Region, use `LOCATION 's3://amzn-s3-demo-bucket/123456789012/CloudTrail/us-east-1/2016/03/14/'.`
   - To analyze network activity data instead of management events, replace `/CloudTrail/` in the `LOCATION` clause with `/CloudTrail-NetworkActivity/`.

   Using the highest level in the object hierarchy gives you the greatest flexibility when you query using Athena.

4. Verify that fields are listed correctly. For more information about the full list of fields in a CloudTrail record, see [CloudTrail record contents](#).

   The example `CREATE TABLE` statement in Step 1 uses the [Hive JSON SerDe](#). In the example, the fields `requestparameters`, `responseelements`, and `additionaleventdata` are listed as type `STRING` in the query, but are `STRUCT` data type used in JSON. Therefore, to get data out of these fields, use `JSON_EXTRACT` functions. For more information, see [the section called “Extract JSON data from strings”](#). For performance improvements, the example partitions the data by AWS Region, year, month, and day.

5. Run the `CREATE TABLE` statement in the Athena console.

6. Use the [ALTER TABLE ADD PARTITION](#) command to load the partitions so that you can query them, as in the following example.

```
ALTER TABLE table_name ADD
   PARTITION (region='us-east-1',
              year='2019',
              month='02',
              day='01')
   LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/Account_ID/CloudTrail/us-east-1/2019/02/01/'
```

Create a table for an organization wide trail using manual partitioning

To create a table for organization wide CloudTrail log files in Athena, follow the steps in Create a table for CloudTrail logs in Athena using manual partitioning, but make the modifications noted in the following procedure.

To create an Athena table for organization wide CloudTrail logs

1. In the CREATE TABLE statement, modify the LOCATION clause to include the organization ID, as in the following example:

```
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/organization_id/'
```

2. In the PARTITIONED BY clause, add an entry for the account ID as a string, as in the following example:

```
PARTITIONED BY (account string, region string, year string, month string, day
 string)
```

The following example shows the combined result:

```
...

PARTITIONED BY (account string, region string, year string, month string, day
 string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/organization_id/Account_ID/CloudTrail/'
```

3. In the ALTER TABLE statement ADD PARTITION clause, include the account ID, as in the following example:

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
region='us-east-1',
year='2022',
month='08',
day='08')
```

4. In the ALTER TABLE statement LOCATION clause, include the organization ID, the account ID, and the partition that you want to add, as in the following example:

```
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/organization_id/Account_ID/CloudTrail/
us-east-1/2022/08/08/'
```

   The following example ALTER TABLE statement shows the combined result:

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
region='us-east-1',
year='2022',
month='08',
day='08')
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/organization_id/111122223333/CloudTrail/
us-east-1/2022/08/08/'
```

Note that, in a large organization, using this method to manually add and maintain a partition for each organization account ID can be cumbersome. In such a scenario, consider using CloudTrail Lake rather than Athena. CloudTrail Lake in such a scenario offers the following advantages:

- Automatically aggregates logs across an entire organization
- Does not require setting up or maintaining partitions or an Athena table
- Queries are run directly in the CloudTrail console
- Uses a SQL-compatible query language

For more information, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

Create the table for CloudTrail logs in Athena using partition projection

Because CloudTrail logs have a known structure whose partition scheme you can specify in advance, you can reduce query runtime and automate partition management by using the Athena partition projection feature. Partition projection automatically adds new partitions as new data is added. This removes the need for you to manually add partitions by using ALTER TABLE ADD PARTITION.

The following example CREATE TABLE statement automatically uses partition projection on CloudTrail logs from a specified date until the present for a single AWS Region. In the LOCATION and `storage.location.template` clauses, replace the *bucket*, *account-id*, and *aws-region* placeholders with correspondingly identical values. For `projection.timestamp.range`, replace *2020*/*01*/*01* with the starting date that you want to use. After you run the query successfully, you can query the table. You do not have to run `ALTER TABLE ADD PARTITION` to load the partitions.

```
CREATE EXTERNAL TABLE cloudtrail_logs_pp(
    eventversion STRING,
    useridentity STRUCT<
        type: STRING,
        principalid: STRING,
        arn: STRING,
        accountid: STRING,
        invokedby: STRING,
        accesskeyid: STRING,
        username: STRING,
        onbehalfof: STRUCT<
             userid: STRING,
             identitystorearn: STRING>,
        sessioncontext: STRUCT<
            attributes: STRUCT<
                mfaauthenticated: STRING,
                creationdate: STRING>,
            sessionissuer: STRUCT<
                type: STRING,
                principalid: STRING,
                arn: STRING,
                accountid: STRING,
                username: STRING>,
            ec2roledelivery:string,
            webidfederationdata: STRUCT<
                federatedprovider: STRING,
                attributes: map<string,string>>
        >
    >,
    eventtime STRING,
    eventsource STRING,
    eventname STRING,
    awsregion STRING,
    sourceipaddress STRING,
    useragent STRING,
    errorcode STRING,
    errormessage STRING,
```

```
    requestparameters STRING,
    responseelements STRING,
    additionaleventdata STRING,
    requestid STRING,
    eventid STRING,
    readonly STRING,
    resources ARRAY<STRUCT<
        arn: STRING,
        accountid: STRING,
        type: STRING>>,
    eventtype STRING,
    apiversion STRING,
    recipientaccountid STRING,
    serviceeventdetails STRING,
    sharedeventid STRING,
    vpcendpointid STRING,
    vpcendpointaccountid STRING,
    eventcategory STRING,
    addendum STRUCT<
      reason:STRING,
      updatedfields:STRING,
      originalrequestid:STRING,
      originaleventid:STRING>,
    sessioncredentialfromconsole STRING,
    edgedevicedetails STRING,
    tlsdetails STRUCT<
      tlsversion:STRING,
      ciphersuite:STRING,
      clientprovidedhostheader:STRING>
  )
PARTITIONED BY (
   `timestamp` string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/AWSLogs/account-id/CloudTrail/aws-region'
TBLPROPERTIES (
  'projection.enabled'='true',
  'projection.timestamp.format'='yyyy/MM/dd',
  'projection.timestamp.interval'='1',
  'projection.timestamp.interval.unit'='DAYS',
  'projection.timestamp.range'='2020/01/01,NOW',
  'projection.timestamp.type'='date',
```

```
  'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/account-
id/CloudTrail/aws-region/${timestamp}')
```

For more information about partition projection, see Use partition projection with Amazon Athena.

Example CloudTrail log queries

The following example shows a portion of a query that returns all anonymous (unsigned) requests from the table created for CloudTrail event logs. This query selects those requests where `useridentity.accountid` is anonymous, and `useridentity.arn` is not specified:

```
SELECT *
FROM cloudtrail_logs
WHERE
    eventsource = 's3.amazonaws.com' AND
    eventname in ('GetObject') AND
    useridentity.accountid = 'anonymous' AND
    useridentity.arn IS NULL AND
    requestparameters LIKE '%[your bucket name ]%';
```

For more information, see the AWS Big Data blog post Analyze security, compliance, and operational activity using AWS CloudTrail and Amazon Athena.

Query nested fields in CloudTrail logs

Because the `userIdentity` and `resources` fields are nested data types, querying them requires special treatment.

The `userIdentity` object consists of nested `STRUCT` types. These can be queried using a dot to separate the fields, as in the following example:

```
SELECT
    eventsource,
    eventname,
    useridentity.sessioncontext.attributes.creationdate,
    useridentity.sessioncontext.sessionissuer.arn
FROM cloudtrail_logs
WHERE useridentity.sessioncontext.sessionissuer.arn IS NOT NULL
ORDER BY eventsource, eventname
LIMIT 10
```

The `resources` field is an array of STRUCT objects. For these arrays, use `CROSS JOIN UNNEST` to unnest the array so that you can query its objects.

The following example returns all rows where the resource ARN ends in `example/datafile.txt`. For readability, the [replace](#) function removes the initial `arn:aws:s3:::` substring from the ARN.

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as s3_resource,
    eventname,
    eventtime,
    useragent
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE unnested.resources_entry.ARN LIKE '%example/datafile.txt'
ORDER BY eventtime
```

The following example queries for `DeleteBucket` events. The query extracts the name of the bucket and the account ID to which the bucket belongs from the `resources` object.

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as deleted_bucket,
    eventtime AS time_deleted,
    useridentity.username,
    unnested.resources_entry.accountid as bucket_acct_id
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE eventname = 'DeleteBucket'
ORDER BY eventtime
```

For more information about unnesting, see [Filter arrays](#).

Tips for querying CloudTrail logs

Consider the following when exploring CloudTrail log data:

- Before querying the logs, verify that your logs table looks the same as the one in [the section called “Use manual partitioning”](#). If it is not the first table, delete the existing table using the following command: `DROP TABLE cloudtrail_logs`.

- After you drop the existing table, re-create it. For more information, see Create a table for CloudTrail logs in Athena using manual partitioning.

  Verify that fields in your Athena query are listed correctly. For information about the full list of fields in a CloudTrail record, see CloudTrail record contents.

  If your query includes fields in JSON formats, such as STRUCT, extract data from JSON. For more information, see Extract JSON data from strings.

  Some suggestions for issuing queries against your CloudTrail table:

- Start by looking at which users called which API operations and from which source IP addresses.
- Use the following basic SQL query as your template. Paste the query to the Athena console and run it.

```
SELECT
 useridentity.arn,
 eventname,
 sourceipaddress,
 eventtime
FROM cloudtrail_logs
LIMIT 100;
```

- Modify the query to further explore your data.
- To improve performance, include the LIMIT clause to return a specified subset of rows.

Query Amazon EMR logs

Amazon EMR and big data applications that run on Amazon EMR produce log files. Log files are written to the primary node, and you can also configure Amazon EMR to archive log files to Amazon S3 automatically. You can use Amazon Athena to query these logs to identify events and trends for applications and clusters. For more information about the types of log files in Amazon EMR and saving them to Amazon S3, see View log files in the *Amazon EMR Management Guide*.

Topics

Create and query a basic table based on Amazon EMR log files

The following example creates a basic table, `myemrlogs`, based on log files saved to `s3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/elasticmapreduce/`. The Amazon S3 location used in the examples below reflects the pattern of the default log location for an EMR cluster created by Amazon Web Services account *123456789012* in Region *us-west-2*. If you use a custom location, the pattern is s3://amzn-s3-demo-bucket/*ClusterID*.

For information about creating a partitioned table to potentially improve query performance and reduce data transfer, see Create and query a partitioned table based on Amazon EMR logs.

```
CREATE EXTERNAL TABLE `myemrlogs`(
  `data` string COMMENT 'from deserializer')
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

Example queries

The following example queries can be run on the `myemrlogs` table created by the previous example.

Example – Query step logs for occurrences of ERROR, WARN, INFO, EXCEPTION, FATAL, or DEBUG

```
SELECT data,
        "$PATH"
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'s-86URH188Z6B1')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – Query a specific instance log, i-00b3c0a839ece0a9c, for ERROR, WARN, INFO, EXCEPTION, FATAL, or DEBUG

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'i-00b3c0a839ece0a9c')
        AND regexp_like("$PATH",'state')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – Query presto application logs for ERROR, WARN, INFO, EXCEPTION, FATAL, or DEBUG

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'presto')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – Query Namenode application logs for ERROR, WARN, INFO, EXCEPTION, FATAL, or DEBUG

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'namenode')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – Query all logs by date and hour for ERROR, WARN, INFO, EXCEPTION, FATAL, or DEBUG

```
SELECT distinct("$PATH") AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'2019-07-23-10')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Create and query a partitioned table based on Amazon EMR logs

These examples use the same log location to create an Athena table, but the table is partitioned, and a partition is then created for each log location. For more information, see Partition your data.

The following query creates the partitioned table named `mypartitionedemrlogs`:

```
CREATE EXTERNAL TABLE `mypartitionedemrlogs`(
  `data` string COMMENT 'from deserializer')
 partitioned by (logtype string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

The following query statements then create table partitions based on sub-directories for different log types that Amazon EMR creates in Amazon S3:

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='containers')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
containers/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-mapreduce')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-mapreduce/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-state-pusher')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-state-pusher/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='node')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
node/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='steps')
```

```
    LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
steps/'
```

After you create the partitions, you can run a `SHOW PARTITIONS` query on the table to confirm:

```
SHOW PARTITIONS mypartitionedemrlogs;
```

Example queries

The following examples demonstrate queries for specific log entries use the table and partitions created by the examples above.

Example – Querying application application_1561661818238_0002 logs in the containers partition for ERROR or WARN

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='containers'
        AND regexp_like("$PATH",'application_1561661818238_0002')
        AND regexp_like(data, 'ERROR|WARN') limit 100;
```

Example – Querying the hadoop-Mapreduce partition for job job_1561661818238_0004 and failed reduces

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='hadoop-mapreduce'
        AND regexp_like(data,'job_1561661818238_0004|Failed Reduces') limit 100;
```

Example – Querying Hive logs in the node partition for query ID 056e0609-33e1-4611-956c-7a31b42d2663

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like("$PATH",'hive')
        AND regexp_like(data,'056e0609-33e1-4611-956c-7a31b42d2663') limit 100;
```

Example – Querying resourcemanager logs in the node partition for application 1567660019320_0001_01_000001

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like(data,'resourcemanager')
        AND regexp_like(data,'1567660019320_0001_01_000001') limit 100
```

Query AWS Global Accelerator flow logs

You can use AWS Global Accelerator to create accelerators that direct network traffic to optimal endpoints over the AWS global network. For more information about Global Accelerator, see What is AWS Global Accelerator.

Global Accelerator flow logs enable you to capture information about the IP address traffic going to and from network interfaces in your accelerators. Flow log data is published to Amazon S3, where you can retrieve and view your data. For more information, see Flow logs in AWS Global Accelerator.

You can use Athena to query your Global Accelerator flow logs by creating a table that specifies their location in Amazon S3.

To create the table for Global Accelerator flow logs

1. Copy and paste the following DDL statement into the Athena console. This query specifies *ROW FORMAT DELIMITED* and omits specifying a SerDe, which means that the query uses the `LazySimpleSerDe`. In this query, fields are terminated by a space.

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS aga_flow_logs (
     version string,
     account string,
     acceleratorid string,
     clientip string,
     clientport int,
     gip string,
     gipport int,
     endpointip string,
     endpointport int,
     protocol string,
   ```

```
  ipaddresstype string,
  numpackets bigint,
  numbytes int,
  starttime int,
  endtime int,
  action string,
  logstatus string,
  agasourceip string,
  agasourceport int,
  endpointregion string,
  agaregion string,
  direction string,
  vpc_id string,
  reject_reason string
)
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/account_id/globalaccelerator/
region/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

2. Modify the `LOCATION` value to point to the Amazon S3 bucket that contains your log data.

```
's3://amzn-s3-demo-bucket/prefix/AWSLogs/account_id/globalaccelerator/region_code/'
```

3. Run the query in the Athena console. After the query completes, Athena registers the `aga_flow_logs` table, making the data in it available for queries.

4. Create partitions to read the data, as in the following sample query. The query creates a single partition for a specified date. Replace the placeholders for date and location.

```
ALTER TABLE aga_flow_logs
ADD PARTITION (dt='YYYY-MM-dd')
LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/account_id/
globalaccelerator/region_code/YYYY/MM/dd';
```

Example queries for AWS Global Accelerator flow logs

Example – List the requests that pass through a specific edge location

The following example query lists requests that passed through the LHR edge location. Use the `LIMIT` operator to limit the number of logs to query at one time.

```
SELECT
  clientip,
  agaregion,
  protocol,
  action
FROM
  aga_flow_logs
WHERE
  agaregion LIKE 'LHR%'
LIMIT
  100;
```

Example – List the endpoint IP addresses that receive the most HTTPS requests

To see which endpoint IP addresses are receiving the highest number of HTTPS requests, use the following query. This query counts the number of packets received on HTTPS port 443, groups them by destination IP address, and returns the top 10 IP addresses.

```
SELECT
  SUM(numpackets) AS packetcount,
  endpointip
FROM
  aga_flow_logs
WHERE
  endpointport = 443
GROUP BY
  endpointip
ORDER BY
  packetcount DESC
LIMIT
  10;
```

Query Amazon GuardDuty findings

Amazon GuardDuty is a security monitoring service for helping to identify unexpected and potentially unauthorized or malicious activity in your AWS environment. When it detects unexpected and potentially malicious activity, GuardDuty generates security findings that you can export to Amazon S3 for storage and analysis. After you export your findings to Amazon S3, you can use Athena to query them. This article shows how to create a table in Athena for your GuardDuty findings and query them.

For more information about Amazon GuardDuty, see the Amazon GuardDuty User Guide.

Prerequisites

- Enable the GuardDuty feature for exporting findings to Amazon S3. For steps, see Exporting findings in the Amazon GuardDuty User Guide.

Create a table in Athena for GuardDuty findings

To query your GuardDuty findings from Athena, you must create a table for them.

To create a table in Athena for GuardDuty findings

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Paste the following DDL statement into the Athena console. Modify the values in `LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/`*`account-id`*`/GuardDuty/'` to point to your GuardDuty findings in Amazon S3.

```
CREATE EXTERNAL TABLE `gd_logs` (
  `schemaversion` string,
  `accountid` string,
  `region` string,
  `partition` string,
  `id` string,
  `arn` string,
  `type` string,
  `resource` string,
  `service` string,
  `severity` string,
  `createdat` string,
  `updatedat` string,
  `title` string,
```

```
  `description` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/account-id/GuardDuty/'
 TBLPROPERTIES ('has_encrypted_data'='true')
```

Note

The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

3. Run the query in the Athena console to register the `gd_logs` table. When the query completes, the findings are ready for you to query from Athena.

Example queries

The following examples show how to query GuardDuty findings from Athena.

Example – DNS data exfiltration

The following query returns information about Amazon EC2 instances that might be exfiltrating data through DNS queries.

```
SELECT
    title,
    severity,
    type,
    id AS FindingID,
    accountid,
    region,
    createdat,
    updatedat,
    json_extract_scalar(service, '$.count') AS Count,
    json_extract_scalar(resource, '$.instancedetails.instanceid') AS InstanceID,
    json_extract_scalar(service, '$.action.actiontype') AS DNS_ActionType,
    json_extract_scalar(service, '$.action.dnsrequestaction.domain') AS DomainName,
    json_extract_scalar(service, '$.action.dnsrequestaction.protocol') AS protocol,
```

```
    json_extract_scalar(service, '$.action.dnsrequestaction.blocked') AS blocked
FROM gd_logs
WHERE type = 'Trojan:EC2/DNSDataExfiltration'
ORDER BY severity DESC
```

Example – Unauthorized IAM user access

The following query returns all `UnauthorizedAccess:IAMUser` finding types for an IAM Principal from all regions.

```
SELECT title,
         severity,
         type,
         id,
         accountid,
         region,
         createdat,
         updatedat,
         json_extract_scalar(service, '$.count') AS Count,
         json_extract_scalar(resource, '$.accesskeydetails.username') AS IAMPrincipal,
         json_extract_scalar(service,'$.action.awsapicallaction.api') AS
 APIActionCalled
FROM gd_logs
WHERE type LIKE '%UnauthorizedAccess:IAMUser%'
ORDER BY severity desc;
```

Tips for querying GuardDuty findings

When you create your query, keep the following points in mind.

- To extract data from nested JSON fields, use the Presto `json_extract` or `json_extract_scalar` functions. For more information, see Extract JSON data from strings.
- Make sure that all characters in the JSON fields are in lower case.
- For information about downloading query results, see Download query results files using the Athena console.

Query AWS Network Firewall logs

AWS Network Firewall is a managed service that you can use to deploy essential network protections for your Amazon Virtual Private Cloud instances. AWS Network Firewall works together with AWS Firewall Manager so you can build policies based on AWS Network Firewall rules and then centrally apply those policies across your VPCs and accounts. For more information about AWS Network Firewall, see AWS Network Firewall.

You can configure AWS Network Firewall logging for traffic that you forward to your firewall's stateful rules engine. Logging gives you detailed information about network traffic, including the time that the stateful engine received a packet, detailed information about the packet, and any stateful rule action taken against the packet. The logs are published to the log destination that you've configured, where you can retrieve and view them. For more information, see Logging network traffic from AWS Network Firewall in the *AWS Network Firewall Developer Guide*.

Topics

Create and query a table for alert logs

1. Modify the following sample DDL statement to conform to the structure of your alert log. You may need to update the statement to include the columns for the latest version of the logs. For more information, see Contents of a firewall log in the *AWS Network Firewall Developer Guide*.

```
CREATE EXTERNAL TABLE network_firewall_alert_logs (
  firewall_name string,
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    sni:string,
    tls_inspected:boolean,
    tls_error:struct<
      error_message:string>,
    revocation_check:struct<
```

```
        leaf_cert_fpr:string,
        status:string,
        action:string>,
      alert:struct<
        alert_id:string,
        alert_type:string,
        action:string,
        signature_id:int,
        rev:int,
        signature:string,
        category:string,
        severity:int,
        rule_name:string,
        alert_name:string,
        alert_severity:string,
        alert_description:string,
        file_name:string,
        file_hash:string,
        packet_capture:string,
        reference_links:array<string>
      >,
      src_country:string,
      dest_country:string,
      src_hostname:string,
      dest_hostname:string,
      user_agent:string,
      url:string
     >
)
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://amzn-s3-demo-bucket/path_to_alert_logs_folder/';
```

2. Modify the `LOCATION` clause to specify the folder for your logs in Amazon S3.
3. Run your `CREATE TABLE` query in the Athena query editor. After the query completes, Athena registers the `network_firewall_alert_logs` table, making the data that it points to ready for queries.

Example query

The sample alert log query in this section filters for events in which TLS inspection was performed that have alerts with a severity level of 2 or higher.

The query uses aliases to create output column headings that show the `struct` that the column belongs to. For example, the column heading for the `event.alert.category` field is `event_alert_category` instead of just `category`. To customize the column names further, you can modify the aliases to suit your preferences. For example, you can use underscores or other separators to delimit the `struct` names and field names.

Remember to modify column names and `struct` references based on your table definition and on the fields that you want in the query result.

```
SELECT
  firewall_name,
  availability_zone,
  event_timestamp,
  event.timestamp AS event_timestamp,
  event.flow_id AS event_flow_id,
  event.event_type AS event_type,
  event.src_ip AS event_src_ip,
  event.src_port AS event_src_port,
  event.dest_ip AS event_dest_ip,
  event.dest_port AS event_dest_port,
  event.proto AS event_protol,
  event.app_proto AS event_app_proto,
  event.sni AS event_sni,
  event.tls_inspected AS event_tls_inspected,
  event.tls_error.error_message AS event_tls_error_message,
  event.revocation_check.leaf_cert_fpr AS event_revocation_leaf_cert,
  event.revocation_check.status AS event_revocation_check_status,
  event.revocation_check.action AS event_revocation_check_action,
  event.alert.alert_id AS event_alert_alert_id,
  event.alert.alert_type AS event_alert_alert_type,
  event.alert.action AS event_alert_action,
  event.alert.signature_id AS event_alert_signature_id,
  event.alert.rev AS event_alert_rev,
  event.alert.signature AS event_alert_signature,
  event.alert.category AS event_alert_category,
  event.alert.severity AS event_alert_severity,
  event.alert.rule_name AS event_alert_rule_name,
  event.alert.alert_name AS event_alert_alert_name,
  event.alert.alert_severity AS event_alert_alert_severity,
  event.alert.alert_description AS event_alert_alert_description,
  event.alert.file_name AS event_alert_file_name,
  event.alert.file_hash AS event_alert_file_hash,
  event.alert.packet_capture AS event_alert_packet_capture,
```

```
  event.alert.reference_links AS event_alert_reference_links,
  event.src_country AS event_src_country,
  event.dest_country AS event_dest_country,
  event.src_hostname AS event_src_hostname,
  event.dest_hostname AS event_dest_hostname,
  event.user_agent AS event_user_agent,
  event.url AS event_url
FROM
  network_firewall_alert_logs
WHERE
  event.alert.severity >= 2
  AND event.tls_inspected = true
LIMIT 10;
```

Create and query a table for netflow logs

1. Modify the following sample DDL statement to conform to the structure of your netflow logs. You may need to update the statement to include the columns for the latest version of the logs. For more information, see [Contents of a firewall log](https://docs.aws.amazon.com/network-firewall/latest/developerguide/firewall-logging.html) in the *AWS Network Firewall Developer Guide*.

```
CREATE EXTERNAL TABLE network_firewall_netflow_logs (
  firewall_name string,
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    tls_inspected:boolean,
    netflow:struct<
      pkts:int,
      bytes:bigint,
      start:string,
      `end`:string,
      age:int,
      min_ttl:int,
```

```
      max_ttl:int,
      tcp_flags:struct<
        syn:boolean,
        fin:boolean,
        rst:boolean,
        psh:boolean,
        ack:boolean,
        urg:boolean
        >
      >
    >
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://amzn-s3-demo-bucket/path_to_netflow_logs_folder/';
```

2. Modify the `LOCATION` clause to specify the folder for your logs in Amazon S3.
3. Run the `CREATE TABLE` query in the Athena query editor. After the query completes, Athena registers the `network_firewall_netflow_logs` table, making the data that it points to ready for queries.

Example query

The sample netflow log query in this section filters for events in which TLS inspection was performed.

The query uses aliases to create output column headings that show the `struct` that the column belongs to. For example, the column heading for the `event.netflow.bytes` field is `event_netflow_bytes` instead of just `bytes`. To customize the column names further, you can modify the aliases to suit your preferences. For example, you can use underscores or other separators to delimit the `struct` names and field names.

Remember to modify column names and `struct` references based on your table definition and on the fields that you want in the query result.

```
SELECT
  event.src_ip AS event_src_ip,
  event.dest_ip AS event_dest_ip,
  event.proto AS event_proto,
  event.app_proto AS event_app_proto,
  event.tls_inspected AS event_tls_inspected,
  event.netflow.pkts AS event_netflow_pkts,
```

```
  event.netflow.bytes AS event_netflow_bytes,
  event.netflow.tcp_flags.syn AS event_netflow_tcp_flags_syn
FROM network_firewall_netflow_logs
WHERE event.tls_inspected = true
```

Query Network Load Balancer logs

Use Athena to analyze and process logs from Network Load Balancer. These logs receive detailed information about the Transport Layer Security (TLS) requests sent to the Network Load Balancer. You can use these access logs to analyze traffic patterns and troubleshoot issues.

Before you analyze the Network Load Balancer access logs, enable and configure them for saving in the destination Amazon S3 bucket. For more information, and for information about each Network Load Balancer access log entry, see Access logs for your Network Load Balancer.

To create the table for Network Load Balancer logs

1. Copy and paste the following DDL statement into the Athena console. Check the syntax of the Network Load Balancer log records. Update the statement as required to include the columns and the regex corresponding to your log records.

```
CREATE EXTERNAL TABLE IF NOT EXISTS nlb_tls_logs (
            type string,
            version string,
            time string,
            elb string,
            listener_id string,
            client_ip string,
            client_port int,
            target_ip string,
            target_port int,
            tcp_connection_time_ms double,
            tls_handshake_time_ms double,
            received_bytes bigint,
            sent_bytes bigint,
            incoming_tls_alert int,
            cert_arn string,
            certificate_serial string,
            tls_cipher_suite string,
            tls_protocol_version string,
            tls_named_group string,
            domain_name string,
```

```
            alpn_fe_protocol string,
            alpn_be_protocol string,
            alpn_client_preference_list string,
            tls_connection_creation_time string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
            '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*):
([0-9]*) ([-.0-9]*) ([-.0-9]*) ([-0-9]*) ([-0-9]*) ([-0-9]*) ([^ ]*) ([^ ]*)
 ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?'
            )
            LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/AWS_account_ID/
elasticloadbalancing/region';
```

2. Modify the LOCATION Amazon S3 bucket to specify the destination of your Network Load Balancer logs.
3. Run the query in the Athena console. After the query completes, Athena registers the nlb_tls_logs table, making the data in it ready for queries.

Example queries

To see how many times a certificate is used, use a query similar to this example:

```
SELECT count(*) AS
         ct,
         cert_arn
FROM "nlb_tls_logs"
GROUP BY  cert_arn;
```

The following query shows how many users are using a TLS version earlier than 1.3:

```
SELECT tls_protocol_version,
         COUNT(tls_protocol_version) AS
         num_connections,
         client_ip
FROM "nlb_tls_logs"
WHERE tls_protocol_version < 'tlsv13'
GROUP BY tls_protocol_version, client_ip;
```

Use the following query to identify connections that take a long TLS handshake time:

```
SELECT *
FROM "nlb_tls_logs"
ORDER BY  tls_handshake_time_ms DESC
LIMIT 10;
```

Use the following query to identify and count which TLS protocol versions and cipher suites have been negotiated in the past 30 days.

```
SELECT tls_cipher_suite,
         tls_protocol_version,
         COUNT(*) AS ct
FROM "nlb_tls_logs"
WHERE from_iso8601_timestamp(time) > current_timestamp - interval '30' day
        AND NOT tls_protocol_version = '-'
GROUP BY tls_cipher_suite, tls_protocol_version
ORDER BY ct DESC;
```

Query Amazon Route 53 resolver query logs

You can create Athena tables for your Amazon Route 53 Resolver query logs and query them from Athena.

Route 53 Resolver query logging is for logging of DNS queries made by resources within a VPC, on-premises resources that use inbound resolver endpoints, queries that use an outbound Resolver endpoint for recursive DNS resolution, and queries that use Route 53 Resolver DNS firewall rules to block, allow, or monitor a domain list. For more information about Resolver query logging, see Resolver query logging in the *Amazon Route 53 Developer Guide*. For information about each of the fields in the logs, see Values that appear in resolver query logs in the *Amazon Route 53 Developer Guide*.

Topics

Create the table for resolver query logs

You can use the Query Editor in the Athena console to create and query a table for your Route 53 Resolver query logs.

To create and query an Athena table for Route 53 resolver query logs

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena Query Editor, enter the following `CREATE TABLE` statement. Replace the `LOCATION` clause values with those corresponding to the location of your Resolver logs in Amazon S3.

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class
    string,
  rcode string,
  answers array<
    struct<
      Rdata: string,
      Type: string,
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
 )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
```

```
LOCATION 's3://amzn-s3-demo-bucket/AWSLogs/aws_account_id/vpcdnsquerylogs/{vpc-
id}/'
```

Because Resolver query log data is in JSON format, the CREATE TABLE statement uses a JSON SerDe library to analyze the data.

> **Note**
>
> The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

3. Choose **Run query**. The statement creates an Athena table named `r53_rlogs` whose columns represent each of the fields in your Resolver log data.
4. In the Athena console Query Editor, run the following query to verify that your table has been created.

```
SELECT * FROM "r53_rlogs" LIMIT 10
```

Use partition projection

The following example shows a `CREATE TABLE` statement for Resolver query logs that uses partition projection and is partitioned by vpc and by date. For more information about partition projection, see Use partition projection with Amazon Athena.

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class string,
  rcode string,
```

```
  answers array<
    struct<
      Rdata: string,
      Type: string,
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
)
PARTITIONED BY (
`date` string,
`vpc` string
)
ROW FORMAT SERDE      'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT          'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION              's3://amzn-s3-demo-bucket/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/'
TBLPROPERTIES(
'projection.enabled' = 'true',
'projection.vpc.type' = 'enum',
'projection.vpc.values' = 'vpc-6446ae02',
'projection.date.type' = 'date',
'projection.date.range' = '2023/06/26,NOW',
'projection.date.format' = 'yyyy/MM/dd',
'projection.date.interval' = '1',
'projection.date.interval.unit' = 'DAYS',
'storage.location.template' = 's3://amzn-s3-demo-bucket/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/${vpc}/${date}/'
)
```

Example queries

The following examples show some queries that you can perform from Athena on your Resolver query logs.

Example 1 - query logs in descending query_timestamp order

The following query displays log results in descending `query_timestamp` order.

```
SELECT * FROM "r53_rlogs"
ORDER BY query_timestamp DESC
```

Example 2 - query logs within specified start and end times

The following query queries logs between midnight and 8am on September 24, 2020. Substitute the start and end times according to your own requirements.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode
FROM "r53_rlogs"
WHERE (parse_datetime(query_timestamp,'yyyy-MM-dd''T''HH:mm:ss''Z')
     BETWEEN parse_datetime('2020-09-24-00:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2020-09-24-00:08:00','yyyy-MM-dd-HH:mm:ss'))
ORDER BY query_timestamp DESC
```

Example 3 - query logs based on a specified DNS query name pattern

The following query selects records whose query name includes the string "example.com".

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE query_name LIKE '%example.com%'
ORDER BY query_timestamp DESC
```

Example 4 - query log requests with no answer

The following query selects log entries in which the request received no answer.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE cardinality(answers) = 0
```

Example 5 - query logs with a specific answer

The following query shows logs in which the `answer.Rdata` value has the specified IP address.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode,
 answer.Rdata
```

```
FROM "r53_rlogs"
CROSS JOIN UNNEST(r53_rlogs.answers) as st(answer)
WHERE answer.Rdata='203.0.113.16';
```

Query Amazon SES event logs

You can use Amazon Athena to query Amazon Simple Email Service (Amazon SES) event logs.

Amazon SES is an email platform that provides a convenient and cost-effective way to send and receive email using your own email addresses and domains. You can monitor your Amazon SES sending activity at a granular level using events, metrics, and statistics.

Based on the characteristics that you define, you can publish Amazon SES events to Amazon CloudWatch, Amazon Data Firehose, or Amazon Simple Notification Service. After the information is stored in Amazon S3, you can query it from Amazon Athena.

For an example Athena `CREATE TABLE` statement for Amazon SES logs, including steps on how to create views and flatten nested arrays in Amazon SES event log data, see "Step 3: Using Amazon Athena to query the SES event logs" in the AWS blog post Analyzing Amazon SES event data with AWS Analytics Services.

Query Amazon VPC flow logs

Amazon Virtual Private Cloud flow logs capture information about the IP traffic going to and from network interfaces in a VPC. Use the logs to investigate network traffic patterns and identify threats and risks across your VPC network.

To query your Amazon VPC flow logs, you have two options:

- **Amazon VPC Console –** Use the Athena integration feature in the Amazon VPC Console to generate an AWS CloudFormation template that creates an Athena database, workgroup, and flow logs table with partitioning for you. The template also creates a set of predefined flow log queries that you can use to obtain insights about the traffic flowing through your VPC.

  For information about this approach, see Query flow logs using Amazon Athena in the *Amazon VPC User Guide*.
- **Amazon Athena console –** Create your tables and queries directly in the Athena console. For more information, continue reading this page.

Before you begin querying the logs in Athena, enable VPC flow logs, and configure them to be saved to your Amazon S3 bucket. After you create the logs, let them run for a few minutes to collect some data. The logs are created in a GZIP compression format that Athena lets you query directly.

When you create a VPC flow log, you can use a custom format when you want to specify the fields to return in the flow log and the order in which the fields appear. For more information about flow log records, see Flow log records in the *Amazon VPC User Guide*.

Considerations and limitations

When you create tables in Athena for Amazon VPC flow logs, remember the following points:

- By default, in Athena, Parquet will access columns by name. For more information, see Handle schema updates.
- Use the names in the flow log records for the column names in Athena. The names of the columns in the Athena schema should exactly match the field names in the Amazon VPC flow logs, with the following differences:
  - Replace the hyphens in the Amazon VPC log field names with underscores in the Athena column names. For information about acceptable characters for database names, table names, and column names in Athena, see Name databases, tables, and columns.
  - Escape the flow log record names that are reserved keywords in Athena by enclosing them with backticks.
- VPC flow logs are AWS account specific. When you publish your log files to Amazon S3, the path that Amazon VPC creates in Amazon S3 includes the ID of the AWS account that was used to create the flow log. For more information, see Publish flow logs to Amazon S3 in the *Amazon VPC User Guide*.

Topics

Create a table for Amazon VPC flow logs and query it

The following procedure creates an Amazon VPC table for Amazon VPC flow logs. When you create a flow log with a custom format, you create a table with fields that match the fields that you specified when you created the flow log in the same order that you specified them.

To create an Athena table for Amazon VPC flow logs

1. Enter a DDL statement like the following into the Athena console query editor, following the guidelines in the [Considerations and limitations](#) section. The sample statement creates a table that has the columns for Amazon VPC flow logs versions 2 through 5 as documented in [Flow log records](#). If you use a different set of columns or order of columns, modify the statement accordingly.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `vpc_flow_logs` (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
```

```
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (`date` date)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Note the following points:

- The query specifies `ROW FORMAT DELIMITED` and omits specifying a SerDe. This means that the query uses the Lazy Simple SerDe for CSV, TSV, and custom-delimited files. In this query, fields are terminated by a space.
- The `PARTITIONED BY` clause uses the `date` type. This makes it possible to use mathematical operators in queries to select what's older or newer than a certain date.

> **Note**
>
> Because `date` is a reserved keyword in DDL statements, it is escaped by back tick characters. For more information, see Escape reserved keywords in queries.

- For a VPC flow log with a different custom format, modify the fields to match the fields that you specified when you created the flow log.

2. Modify the `LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/{account_id}/vpcflowlogs/{region_code}/'` to point to the Amazon S3 bucket that contains your log data.
3. Run the query in Athena console. After the query completes, Athena registers the `vpc_flow_logs` table, making the data in it ready for you to issue queries.
4. Create partitions to be able to read the data, as in the following sample query. This query creates a single partition for a specified date. Replace the placeholders for date and location as needed.

Note

This query creates a single partition only, for a date that you specify. To automate the process, use a script that runs this query and creates partitions this way for the `year/month/day`, or use a `CREATE TABLE` statement that specifies partition projection.

```
ALTER TABLE vpc_flow_logs
ADD PARTITION (`date`='YYYY-MM-dd')
LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/YYYY/MM/dd';
```

Example queries for the vpc_flow_logs table

Use the query editor in the Athena console to run SQL statements on the table that you create. You can save the queries, view previous queries, or download query results in CSV format. In the following examples, replace `vpc_flow_logs` with the name of your table. Modify the column values and other variables according to your requirements.

The following example query lists a maximum of 100 flow logs for the date specified.

```
SELECT *
FROM vpc_flow_logs
WHERE date = DATE('2020-05-04')
LIMIT 100;
```

The following query lists all of the rejected TCP connections and uses the newly created date partition column, `date`, to extract from it the day of the week for which these events occurred.

```
SELECT day_of_week(date) AS
  day,
  date,
  interface_id,
  srcaddr,
  action,
  protocol
FROM vpc_flow_logs
WHERE action = 'REJECT' AND protocol = 6
```

```
LIMIT 100;
```

To see which one of your servers is receiving the highest number of HTTPS requests, use the following query. It counts the number of packets received on HTTPS port 443, groups them by destination IP address, and returns the top 10 from the last week.

```
SELECT SUM(packets) AS
  packetcount,
  dstaddr
FROM vpc_flow_logs
WHERE dstport = 443 AND date > current_date - interval '7' day
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10;
```

Create tables for flow logs in Apache Parquet format

The following procedure creates an Amazon VPC table for Amazon VPC flow logs in Apache Parquet format.

To create an Athena table for Amazon VPC flow logs in Parquet format

1. Enter a DDL statement like the following into the Athena console query editor, following the guidelines in the Considerations and limitations section. The sample statement creates a table that has the columns for Amazon VPC flow logs versions 2 through 5 as documented in Flow log records in Parquet format, Hive partitioned hourly. If you do not have hourly partitions, remove hour from the `PARTITIONED BY` clause.

```
CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs_parquet (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
```

```
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (
  `aws-account-id` string,
  `aws-service` string,
  `aws-region` string,
  `year` string,
  `month` string,
  `day` string,
  `hour` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/prefix/AWSLogs/'
TBLPROPERTIES (
  'EXTERNAL'='true',
  'skip.header.line.count'='1'
  )
```

2. Modify the sample `LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/'` to point to the Amazon S3 path that contains your log data.
3. Run the query in Athena console.

4. If your data is in Hive-compatible format, run the following command in the Athena console to update and load the Hive partitions in the metastore. After the query completes, you can query the data in the `vpc_flow_logs_parquet` table.

   ```
   MSCK REPAIR TABLE vpc_flow_logs_parquet
   ```

   If you are not using Hive compatible data, run ALTER TABLE ADD PARTITION to load the partitions.

For more information about using Athena to query Amazon VPC flow logs in Parquet format, see the post Optimize performance and reduce costs for network analytics with VPC Flow Logs in Apache Parquet format in the *AWS Big Data Blog*.

Create and query a table for Amazon VPC flow logs using partition projection

Use a `CREATE TABLE` statement like the following to create a table, partition the table, and populate the partitions automatically by using partition projection. Replace the table name `test_table_vpclogs` in the example with the name of your table. Edit the `LOCATION` clause to specify the Amazon S3 bucket that contains your Amazon VPC log data.

The following `CREATE TABLE` statement is for VPC flow logs delivered in non-Hive style partitioning format. The example allows for multi-account aggregation. If you are centralizing VPC Flow logs from multiple accounts into one Amazon S3 bucket, the account ID must be entered in the Amazon S3 path.

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
```

```
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (accid string, region string, day string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '$LOCATION_OF_LOGS'
TBLPROPERTIES
(
"skip.header.line.count"="1",
"projection.enabled" = "true",
"projection.accid.type" = "enum",
"projection.accid.values" = "$ACCID_1,$ACCID_2",
"projection.region.type" = "enum",
"projection.region.values" = "$REGION_1,$REGION_2,$REGION_3",
"projection.day.type" = "date",
"projection.day.range" = "$START_RANGE,NOW",
"projection.day.format" = "yyyy/MM/dd",
"storage.location.template" = "s3://amzn-s3-demo-bucket/AWSLogs/${accid}/vpcflowlogs/
${region}/${day}"
)
```

Example queries for test_table_vpclogs

The following example queries query the `test_table_vpclogs` created by the preceding `CREATE TABLE` statement. Replace `test_table_vpclogs` in the queries with the name of your own table. Modify the column values and other variables according to your requirements.

To return the first 100 access log entries in chronological order for a specified period of time, run a query like the following.

```
SELECT *
FROM test_table_vpclogs
WHERE day >= '2021/02/01' AND day < '2021/02/28'
ORDER BY day ASC
LIMIT 100
```

To view which server receives the top ten number of HTTP packets for a specified period of time, run a query like the following. The query counts the number of packets received on HTTPS port 443, groups them by destination IP address, and returns the top 10 entries from the previous week.

```
SELECT SUM(packets) AS packetcount,
       dstaddr
FROM test_table_vpclogs
WHERE dstport = 443
  AND day >= '2021/03/01'
  AND day < '2021/03/31'
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10
```

To return the logs that were created during a specified period of time, run a query like the following.

```
SELECT interface_id,
       srcaddr,
       action,
       protocol,
       to_iso8601(from_unixtime(start)) AS start_time,
       to_iso8601(from_unixtime("end")) AS end_time
FROM test_table_vpclogs
WHERE DAY >= '2021/04/01'
  AND DAY < '2021/04/30'
```

To return the access logs for a source IP address between specified time periods, run a query like the following.

```
SELECT *
FROM test_table_vpclogs
WHERE srcaddr = '10.117.1.22'
  AND day >= '2021/02/01'
  AND day < '2021/02/28'
```

To list rejected TCP connections, run a query like the following.

```
SELECT day,
       interface_id,
       srcaddr,
       action,
       protocol
FROM test_table_vpclogs
WHERE action = 'REJECT' AND protocol = 6 AND day >= '2021/02/01' AND day < '2021/02/28'
LIMIT 10
```

To return the access logs for the IP address range that starts with `10.117`, run a query like the following.

```
SELECT *
FROM test_table_vpclogs
WHERE split_part(srcaddr,'.', 1)='10'
  AND split_part(srcaddr,'.', 2) ='117'
```

To return the access logs for a destination IP address between a certain time range, run a query like the following.

```
SELECT *
FROM test_table_vpclogs
WHERE dstaddr = '10.0.1.14'
  AND day >= '2021/01/01'
  AND day < '2021/01/31'
```

Create tables for flow logs in Apache Parquet format using partition projection

The following partition projection `CREATE TABLE` statement for VPC flow logs is in Apache Parquet format, not Hive compatible, and partitioned by hour and by date instead of day. Replace the table name `test_table_vpclogs_parquet` in the example with the name of your table. Edit the `LOCATION` clause to specify the Amazon S3 bucket that contains your Amazon VPC log data.

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs_parquet (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
```

```
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (region string, date string, hour string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/prefix/AWSLogs/{account_id}/vpcflowlogs/'
TBLPROPERTIES (
"EXTERNAL"="true",
"skip.header.line.count" = "1",
"projection.enabled" = "true",
"projection.region.type" = "enum",
"projection.region.values" = "us-east-1,us-west-2,ap-south-1,eu-west-1",
"projection.date.type" = "date",
"projection.date.range" = "2021/01/01,NOW",
"projection.date.format" = "yyyy/MM/dd",
"projection.hour.type" = "integer",
"projection.hour.range" = "00,23",
"projection.hour.digits" = "2",
```

```
"storage.location.template" = "s3://amzn-s3-demo-bucket/prefix/AWSLogs/${account_id}/
vpcflowlogs/${region}/${date}/${hour}"
)
```

Additional resources

For more information about using Athena to analyze VPC flow logs, see the following AWS Big Data blog posts:

- Analyze VPC Flow Logs with point-and-click Amazon Athena integration
- Analyzing VPC flow logs using Amazon Athena and QuickSight
- Optimize performance and reduce costs for network analytics with VPC Flow Logs in Apache Parquet format

Query AWS WAF logs

AWS WAF is a web application firewall that lets you monitor and control the HTTP and HTTPS requests that your protected web applications receive from clients. You define how to handle the web requests by configuring rules inside an AWS WAF web access control list (ACL). You then protect a web application by associating a web ACL to it. Examples of web application resources that you can protect with AWS WAF include Amazon CloudFront distributions, Amazon API Gateway REST APIs, and Application Load Balancers. For more information about AWS WAF, see AWS WAF in the *AWS WAF developer guide*.

AWS WAF logs include information about the traffic that is analyzed by your web ACL, such as the time that AWS WAF received the request from your AWS resource, detailed information about the request, and the action for the rule that each request matched.

You can configure an AWS WAF web ACL to publish logs to one of several destinations, where you can query and view them. For more information about configuring web ACL logging and the contents of the AWS WAF logs, see Logging AWS WAF web ACL traffic in the *AWS WAF developer guide*.

For information on how to use Athena to analyze AWS WAF logs for insights into threat detection and potential security attacks, see the AWS Networking & Content Delivery Blog post How to use Amazon Athena queries to analyze AWS WAF logs and provide the visibility needed for threat detection.

For an example of how to aggregate AWS WAF logs into a central data lake repository and query them with Athena, see the AWS Big Data Blog post [Analyzing AWS WAF logs with OpenSearch Service, Amazon Athena, and QuickSight](#).

This topic provides example `CREATE TABLE` statements for partition projection, manual partitioning, and one that does not uses any partitioning.

> **Note**
>
> The `CREATE TABLE` statements in this topic can be used for both v1 and v2 AWS WAF logs. In v1, the `webaclid` field contains an ID. In v2, the `webaclid` field contains a full ARN. The `CREATE TABLE` statements here treat this content agnostically by using the `string` data type.

Topics

Create a table for AWS WAF S3 logs in Athena using partition projection

Because AWS WAF logs have a known structure whose partition scheme you can specify in advance, you can reduce query runtime and automate partition management by using the Athena partition projection feature. Partition projection automatically adds new partitions as new data is added. This removes the need for you to manually add partitions by using `ALTER TABLE ADD PARTITION`.

The following example `CREATE TABLE` statement automatically uses partition projection on AWS WAF logs from a specified date until the present for four different AWS regions. The `PARTITION BY` clause in this example partitions by region and by date, but you can modify this according to your requirements. Modify the fields as necessary to match your log output. In the `LOCATION` and `storage.location.template` clauses, replace the *`amzn-s3-demo-bucket`* and *`AWS_ACCOUNT_NUMBER`* placeholders with values that identify the Amazon S3 bucket location of your AWS WAF logs. For `projection.day.range`, replace *`2021`*/*`01`*/*`01`* with the starting date that you want to use. After you run the query successfully, you can query the table. You do not have to run `ALTER TABLE ADD PARTITION` to load the partitions.

```
CREATE EXTERNAL TABLE `waf_logs_partition_projection`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails`
 array<struct<conditiontype:string,sensitivitylevel:string,location:string,matcheddata:array<st
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist`
 array<struct<rulegroupid:string,terminatingrule:struct<ruleid:string,action:string,rulematchde
  `ratebasedrulelist`
 array<struct<ratebasedruleid:string,limitkey:string,maxrateallowed:int>>,
  `nonterminatingmatchingrules`
 array<struct<ruleid:string,action:string,rulematchdetails:array<struct<conditiontype:string,se
  `requestheadersinserted` array<struct<name:string,value:string>>,
  `responsecodesent` string,
  `httprequest`
 struct<clientip:string,country:string,headers:array<struct<name:string,value:string>>,uri:stri
  `labels` array<struct<name:string>>,
  `captcharesponse`
 struct<responsecode:string,solvetimestamp:string,failurereason:string>,
  `challengeresponse`
 struct<responsecode:string,solvetimestamp:string,failurereason:string>,
  `ja3fingerprint` string,
  `ja4fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int)
  PARTITIONED BY (
   `log_time` string)
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
```

```
  's3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_NUMBER/WAFLogs/cloudfront/testui/'
TBLPROPERTIES (
 'projection.enabled'='true',
  'projection.log_time.format'='yyyy/MM/dd/HH/mm',
  'projection.log_time.interval'='1',
  'projection.log_time.interval.unit'='minutes',
  'projection.log_time.range'='2025/01/01/00/00,NOW',
  'projection.log_time.type'='date',
  'storage.location.template'='s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_NUMBER/
WAFLogs/cloudfront/testui/${log_time}')
```

> **Note**
>
> The format of the path in the `LOCATION` clause in the example is standard but can vary based on the AWS WAF configuration that you have implemented. For example, the following example AWS WAF logs path is for a CloudFront distribution:
>
> ```
> s3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_NUMBER/WAFLogs/cloudfront/
> cloudfronyt/2025/01/01/00/00/
> ```
>
> If you experience issues while creating or querying your AWS WAF logs table, confirm the location of your log data or [contact Support](#).

For more information about partition projection, see [Use partition projection with Amazon Athena](#).

Create a table for AWS WAF S3 logs in Athena using manual partition

This section describes how to create a table for AWS WAF logs using manual partition.

In the `LOCATION` and `storage.location.template` clauses, replace the *`amzn-s3-demo-bucket`* and *`AWS_ACCOUNT_NUMBER`* placeholders with values that identify the Amazon S3 bucket location of your AWS WAF logs.

```
CREATE EXTERNAL TABLE `waf_logs_manual_partition`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
```

```
  `action` string,
  `terminatingrulematchdetails`
 array<struct<conditiontype:string,sensitivitylevel:string,location:string,matcheddata:array<st
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist`
 array<struct<rulegroupid:string,terminatingrule:struct<ruleid:string,action:string,rulematchde
  `ratebasedrulelist`
 array<struct<ratebasedruleid:string,limitkey:string,maxrateallowed:int>>,
  `nonterminatingmatchingrules`
 array<struct<ruleid:string,action:string,rulematchdetails:array<struct<conditiontype:string,se
  `requestheadersinserted` array<struct<name:string,value:string>>,
  `responsecodesent` string,
  `httprequest`
 struct<clientip:string,country:string,headers:array<struct<name:string,value:string>>,uri:stri
  `labels` array<struct<name:string>>,
  `captcharesponse`
 struct<responsecode:string,solvetimestamp:string,failurereason:string>,
  `challengeresponse`
 struct<responsecode:string,solvetimestamp:string,failurereason:string>,
  `ja3fingerprint` string,
  `ja4fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int)
  PARTITIONED BY ( `year` string, `month` string, `day` string, `hour` string, `min`
 string)
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/AWSLogs/AWS_ACCOUNT_NUMBER/WAFLogs/cloudfront/webacl/'
```

Create a table for AWS WAF logs without partitioning

This section describes how to create a table for AWS WAF logs without partitioning or partition projection.

Note

For performance and cost reasons, we do not recommend using non-partitioned schema for queries. For more information, see Top 10 Performance Tuning Tips for Amazon Athena in the AWS Big Data Blog.

To create the AWS WAF table

1. Copy and paste the following DDL statement into the Athena console. Modify the fields as necessary to match your log output. Modify the `LOCATION` for the Amazon S3 bucket to correspond to the one that stores your logs.

   This query uses the OpenX JSON SerDe.

   Note

   The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON Data Files in the OpenX SerDe documentation on GitHub.

```
CREATE EXTERNAL TABLE `waf_logs`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails` array <
                                    struct <
                                        conditiontype: string,
                                        sensitivitylevel: string,
                                        location: string,
                                        matcheddata: array < string >
```

```
                                        >
                                    >,
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist` array <
                      struct <
                          rulegroupid: string,
                          terminatingrule: struct <
                                              ruleid: string,
                                              action: string,
                                              rulematchdetails: array <
                                                                   struct <

conditiontype: string,

sensitivitylevel: string,
                                                                       location:
string,
                                                                       matcheddata:
array < string >
                                                                          >
                                                                    >
                                                >,
                          nonterminatingmatchingrules: array <
                                                             struct <
                                                                 ruleid: string,
                                                                 action: string,
                                                                 overriddenaction:
string,
                                                                 rulematchdetails:
array <

   struct <

       conditiontype: string,

       sensitivitylevel: string,

       location: string,

       matcheddata: array < string >

          >
                                                                   >,
```

```
challengeresponse: struct <

responsecode: string,

solvetimestamp: string
                                                                             >,
                                                                  captcharesponse: 
struct <

responsecode: string,

solvetimestamp: string
                                                                             >
                                                                   >
                                                           >,
                         excludedrules: string
                           >
                      >,
`ratebasedrulelist` array <
                        struct <
                            ratebasedruleid: string,
                            limitkey: string,
                            maxrateallowed: int
                              >
                         >,
  `nonterminatingmatchingrules` array <
                                   struct <
                                       ruleid: string,
                                       action: string,
                                       rulematchdetails: array <
                                                           struct <
                                                               conditiontype: 
string,

                                                               sensitivitylevel: 
string,

                                                               location: string,
                                                               matcheddata: array 
< string >

                                                                   >
                                                           >,
                                       challengeresponse: struct <
                                                          responsecode: string,
                                                          solvetimestamp: string
```

```
                                                         >,
                                    captcharesponse: struct <
                                                        responsecode: string,
                                                        solvetimestamp: string
                                                         >
                                      >
                                >,
`requestheadersinserted` array <
                              struct <
                                  name: string,
                                  value: string
                                    >
                               >,
`responsecodesent` string,
`httprequest` struct <
                  clientip: string,
                  country: string,
                  headers: array <
                              struct <
                                  name: string,
                                  value: string
                                    >
                               >,
                  uri: string,
                  args: string,
                  httpversion: string,
                  httpmethod: string,
                  requestid: string
                    >,
`labels` array <
             struct <
                 name: string
                   >
              >,
`captcharesponse` struct <
                      responsecode: string,
                      solvetimestamp: string,
                      failureReason: string
                        >,
`challengeresponse` struct <
                      responsecode: string,
                      solvetimestamp: string,
                      failureReason: string
                      >,
```

```
  `ja3Fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/prefix/'
```

2. Run the `CREATE EXTERNAL TABLE` statement in the Athena console query editor. This registers the `waf_logs` table and makes the data in it available for queries from Athena.

Example queries for AWS WAF logs

Many of the example queries in this section use the partition projection table created previously. Modify the table name, column values, and other variables in the examples according to your requirements. To improve the performance of your queries and reduce cost, add the partition column in the filter condition.

Topics

- Count referrers, IP addresses, or matched rules
- Query using date and time
- Query for blocked requests or addresses

Count referrers, IP addresses, or matched rules

The examples in this section query for counts of log items of interest.

- Count the number of referrers that contain a specified term
- Count all matched IP addresses in the last 10 days that have matched excluded rules
- Group all counted managed rules by the number of times matched
- Group all counted custom rules by number of times matched

Example – Count the number of referrers that contain a specified term

The following query counts the number of referrers that contain the term "amazon" for the specified date range.

```
WITH test_dataset AS
  (SELECT header FROM waf_logs
    CROSS JOIN UNNEST(httprequest.headers) AS t(header) WHERE "date" >= '2021/03/01'
    AND "date" < '2021/03/31')
SELECT COUNT(*) referer_count
FROM test_dataset
WHERE LOWER(header.name)='referer' AND header.value LIKE '%amazon%'
```

Example – Count all matched IP addresses in the last 10 days that have matched excluded rules

The following query counts the number of times in the last 10 days that the IP address matched the excluded rule in the rule group.

```
WITH test_dataset AS
  (SELECT * FROM waf_logs
    CROSS JOIN UNNEST(rulegrouplist) AS t(allrulegroups))
SELECT
  COUNT(*) AS count,
  "httprequest"."clientip",
  "allrulegroups"."excludedrules",
  "allrulegroups"."ruleGroupId"
FROM test_dataset
WHERE allrulegroups.excludedrules IS NOT NULL AND from_unixtime(timestamp/1000) > now()
 - interval '10' day
GROUP BY "httprequest"."clientip", "allrulegroups"."ruleGroupId",
 "allrulegroups"."excludedrules"
ORDER BY count DESC
```

Example – Group all counted managed rules by the number of times matched

If you set rule group rule actions to Count in your web ACL configuration before October 27, 2022, AWS WAF saved your overrides in the web ACL JSON as `excludedRules`. Now, the JSON setting for overriding a rule to Count is in the `ruleActionOverrides` settings. For more information, see Action overrides in rule groups in the *AWS WAF Developer Guide*. To extract managed rules in Count mode from the new log structure, query the `nonTerminatingMatchingRules` in the `ruleGroupList` section instead of the `excludedRules` field, as in the following example.

```
SELECT
 count(*) AS count,
 httpsourceid,
 httprequest.clientip,
```

```
 t.rulegroupid,
 t.nonTerminatingMatchingRules
FROM "waf_logs"
CROSS JOIN UNNEST(rulegrouplist) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(t.nonTerminatingMatchingRules) > 0
GROUP BY t.nonTerminatingMatchingRules, action, httpsourceid, httprequest.clientip,
 t.rulegroupid
ORDER BY "count" DESC
Limit 50
```

Example – Group all counted custom rules by number of times matched

The following query groups all counted custom rules by the number of times matched.

```
SELECT
  count(*) AS count,
         httpsourceid,
         httprequest.clientip,
         t.ruleid,
         t.action
FROM "waf_logs"
CROSS JOIN UNNEST(nonterminatingmatchingrules) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(nonTerminatingMatchingRules) > 0
GROUP BY t.ruleid, t.action, httpsourceid, httprequest.clientip
ORDER BY "count" DESC
Limit 50
```

For information about the log locations for custom rules and managed rule groups, see Monitoring and tuning in the *AWS WAF Developer Guide*.

Query using date and time

The examples in this section include queries that use date and time values.

- Return the timestamp field in human-readable ISO 8601 format
- Return records from the last 24 hours
- Return records for a specified date range and IP address
- For a specified date range, count the number of IP addresses in five minute intervals
- Count the number of X-Forwarded-For IP in the last 10 days

Example – Return the timestamp field in human-readable ISO 8601 format

The following query uses the `from_unixtime` and `to_iso8601` functions to return the `timestamp` field in human-readable ISO 8601 format (for example, `2019-12-13T23:40:12.000Z` instead of `1576280412771`). The query also returns the HTTP source name, source ID, and request.

```
SELECT to_iso8601(from_unixtime(timestamp / 1000)) as time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
LIMIT 10;
```

Example – Return records from the last 24 hours

The following query uses a filter in the `WHERE` clause to return the HTTP source name, HTTP source ID, and HTTP request fields for records from the last 24 hours.

```
SELECT to_iso8601(from_unixtime(timestamp/1000)) AS time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
WHERE from_unixtime(timestamp/1000) > now() - interval '1' day
LIMIT 10;
```

Example – Return records for a specified date range and IP address

The following query lists the records in a specified date range for a specified client IP address.

```
SELECT *
FROM waf_logs
WHERE httprequest.clientip='53.21.198.66' AND "date" >= '2021/03/01' AND "date" <
 '2021/03/31'
```

Example – For a specified date range, count the number of IP addresses in five minute intervals

The following query counts, for a particular date range, the number of IP addresses in five minute intervals.

```
WITH test_dataset AS
  (SELECT
     format_datetime(from_unixtime((timestamp/1000) -
 ((minute(from_unixtime(timestamp / 1000))%5) * 60)),'yyyy-MM-dd HH:mm') AS
 five_minutes_ts,
     "httprequest"."clientip"
     FROM waf_logs
     WHERE "date" >= '2021/03/01' AND "date" < '2021/03/31')
SELECT five_minutes_ts,"clientip",count(*) ip_count
FROM test_dataset
GROUP BY five_minutes_ts,"clientip"
```

Example – Count the number of X-Forwarded-For IP in the last 10 days

The following query filters the request headers and counts the number of X-Forwarded-For IP in the last 10 days.

```
WITH test_dataset AS
  (SELECT header
   FROM waf_logs
   CROSS JOIN UNNEST (httprequest.headers) AS t(header)
   WHERE from_unixtime("timestamp"/1000) > now() - interval '10' DAY)
SELECT header.value AS ip,
       count(*) AS COUNT
FROM test_dataset
WHERE header.name='X-Forwarded-For'
GROUP BY header.value
ORDER BY COUNT DESC
```

For more information about date and time functions, see Date and time functions and operators in the Trino documentation.

Query for blocked requests or addresses

The examples in this section query for blocked requests or addresses.

- Extract the top 100 IP addresses blocked by a specified rule type
- Count the number of times a request from a specified country has been blocked
- Count the number of times a request has been blocked, grouping by specific attributes
- Count the number of times a specific terminating rule ID has been matched
- Retrieve the top 100 IP addresses blocked during a specified date range

Example – Extract the top 100 IP addresses blocked by a specified rule type

The following query extracts and counts the top 100 IP addresses that have been blocked by the RATE_BASED terminating rule during the specified date range.

```
SELECT COUNT(httpRequest.clientIp) as count,
httpRequest.clientIp
FROM waf_logs
WHERE terminatingruletype='RATE_BASED' AND action='BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY httpRequest.clientIp
ORDER BY count DESC
LIMIT 100
```

Example – Count the number of times a request from a specified country has been blocked

The following query counts the number of times the request has arrived from an IP address that belongs to Ireland (IE) and has been blocked by the RATE_BASED terminating rule.

```
SELECT
  COUNT(httpRequest.country) as count,
  httpRequest.country
FROM waf_logs
WHERE
  terminatingruletype='RATE_BASED' AND
  httpRequest.country='IE'
GROUP BY httpRequest.country
ORDER BY count
LIMIT 100;
```

Example – Count the number of times a request has been blocked, grouping by specific attributes

The following query counts the number of times the request has been blocked, with results grouped by WebACL, RuleId, ClientIP, and HTTP Request URI.

```
SELECT
  COUNT(*) AS count,
  webaclid,
  terminatingruleid,
  httprequest.clientip,
  httprequest.uri
```

```
FROM waf_logs
WHERE action='BLOCK'
GROUP BY webaclid, terminatingruleid, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example – Count the number of times a specific terminating rule ID has been matched

The following query counts the number of times a specific terminating rule ID has been matched (`WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e'`). The query then groups the results by WebACL, Action, ClientIP, and HTTP Request URI.

```
SELECT
  COUNT(*) AS count,
  webaclid,
  action,
  httprequest.clientip,
  httprequest.uri
FROM waf_logs
WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e'
GROUP BY webaclid, action, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example – Retrieve the top 100 IP addresses blocked during a specified date range

The following query extracts the top 100 IP addresses that have been blocked for a specified date range. The query also lists the number of times the IP addresses have been blocked.

```
SELECT "httprequest"."clientip", "count"(*) "ipcount", "httprequest"."country"
FROM waf_logs
WHERE "action" = 'BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY "httprequest"."clientip", "httprequest"."country"
ORDER BY "ipcount" DESC limit 100
```

For information about querying Amazon S3 logs, see the following topics:

- How do I analyze my Amazon S3 server access logs using Athena? in the AWS Knowledge Center
- Querying Amazon S3 access logs for requests using Amazon Athena in the Amazon Simple Storage Service User Guide

- Using AWS CloudTrail to identify Amazon S3 requests in the Amazon Simple Storage Service User Guide

Query web server logs stored in Amazon S3

You can use Athena to query Web server logs stored in Amazon S3. The topics in this section show you how to create tables in Athena to query Web server logs in a variety of formats.

Topics

Query Apache logs stored in Amazon S3

You can use Amazon Athena to query Apache HTTP Server log files stored in your Amazon S3 account. This topic shows you how to create table schemas to query Apache Access log files in the common log format.

Fields in the common log format include the client IP address, client ID, user ID, request received timestamp, text of the client request, server status code, and size of the object returned to the client.

The following example data shows the Apache common log format.

```
198.51.100.7 - Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200 232
198.51.100.14 - Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html HTTP/1.1" 200 2165
198.51.100.22 - Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html HTTP/1.1" 200 1287
198.51.100.9 - Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1" 404 230
198.51.100.2 - Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1" 404 30
198.51.100.13 - Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html HTTP/1.1" 200 1608
198.51.100.11 - Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html HTTP/1.1"
 200 1344
```

Create a table in Athena for Apache logs

Before you can query Apache logs stored in Amazon S3, you must create a table schema for Athena so that it can read the log data. To create an Athena table for Apache logs, you can use the Grok SerDe. For more information about using the Grok SerDe, see Writing grok custom classifiers in the *AWS Glue Developer Guide*.

To create a table in Athena for Apache web server logs

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Paste the following DDL statement into the Athena Query Editor. Modify the values in `LOCATION 's3://amzn-s3-demo-bucket/apache-log-folder/'` to point to your Apache logs in Amazon S3.

```
CREATE EXTERNAL TABLE apache_logs (
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{USERNAME:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://amzn-s3-demo-bucket/apache-log-folder/';
```

3. Run the query in the Athena console to register the `apache_logs` table. When the query completes, the logs are ready for you to query from Athena.

Example queries

Example – Filter for 404 errors

The following example query selects the request received time, text of the client request, and server status code from the `apache_logs` table. The `WHERE` clause filters for HTTP status code `404` (page not found).

```
SELECT request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '404'
```

The following image shows the results of the query in the Athena Query Editor.

Results

	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

Example – Filter for successful requests

The following example query selects the user ID, request received time, text of the client request, and server status code from the `apache_logs` table. The `WHERE` clause filters for HTTP status code `200` (successful).

```
SELECT user_id, request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '200'
```

The following image shows the results of the query in the Athena Query Editor.

Results

	user_id	request_received_time	client_request	server_status
1	Li	[10/Oct/2019:13:55:36 -0700]	GET /logo.gif HTTP/1.0	200
2	Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
3	Mateo	[27/Dec/2019:11:38:12 -0700]	GET /about.html HTTP/1.1	200
4	Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200
5	Xiulan	[22/Apr/2019:10:51:34 -0700]	GET /group/index.html HTTP/1.1	200

Example – Filter by timestamp

The following example queries for records whose request received time is greater than the specified timestamp.

```
SELECT * FROM apache_logs WHERE request_received_time > 10/Oct/2023:00:00:00
```

Query internet information server (IIS) logs stored in Amazon S3

You can use Amazon Athena to query Microsoft Internet Information Services (IIS) web server logs stored in your Amazon S3 account. While IIS uses a variety of log file formats, this topic shows you how to create table schemas to query W3C extended and IIS log file format logs from Athena.

Because the W3C extended and IIS log file formats use single character delimiters (spaces and commas, respectively) and do not have values enclosed in quotation marks, you can use the LazySimpleSerDe to create Athena tables for them.

Topics

Query W3C extended log file format

The W3C extended log file data format has space-separated fields. The fields that appear in W3C extended logs are determined by a web server administrator who chooses which log fields to include. The following example log data has the fields `date, time`, `c-ip`, `s-ip`, `cs-method`, `cs-uri-stem`, `sc-status`, `sc-bytes`, `cs-bytes`, `time-taken`, and `cs-version`.

```
2020-01-19 22:48:39 203.0.113.5 198.51.100.2 GET /default.html 200 540 524 157 HTTP/1.0
2020-01-19 22:49:40 203.0.113.10 198.51.100.12 GET /index.html 200 420 324 164 HTTP/1.0
2020-01-19 22:50:12 203.0.113.12 198.51.100.4 GET /image.gif 200 324 320 358 HTTP/1.0
2020-01-19 22:51:44 203.0.113.15 198.51.100.16 GET /faq.html 200 330 324 288 HTTP/1.0
```

Create a table in Athena for W3C extended logs

Before you can query your W3C extended logs, you must create a table schema so that Athena can read the log data.

To create a table in Athena for W3C extended logs

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Paste a DDL statement like the following into the Athena console, noting the following points:

   a. Add or remove the columns in the example to correspond to the fields in the logs that you want to query.

   b. Column names in the W3C extended log file format contain hyphens (-). However, in accordance with [Athena naming conventions](#), the example `CREATE TABLE` statement replaces them with underscores (_).

   c. To specify the space delimiter, use `FIELDS TERMINATED BY ' '`.

   d. Modify the values in `LOCATION 's3://amzn-s3-demo-bucket/w3c-log-folder/'` to point to your W3C extended logs in Amazon S3.

```
CREATE EXTERNAL TABLE `iis_w3c_logs`(
  date_col string,
  time_col string,
  c_ip string,
  s_ip string,
  cs_method string,
  cs_uri_stem string,
  sc_status string,
  sc_bytes string,
  cs_bytes string,
  time_taken string,
  cs_version string
  )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ' '
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION   's3://amzn-s3-demo-bucket/w3c-log-folder/'
```

3. Run the query in the Athena console to register the `iis_w3c_logs` table. When the query completes, the logs are ready for you to query from Athena.

Example W3C extended log select query

The following example query selects the date, time, request target, and time taken for the request from the table `iis_w3c_logs`. The `WHERE` clause filters for cases in which the HTTP method is `GET` and the HTTP status code is `200` (successful).

```
SELECT date_col, time_col, cs_uri_stem, time_taken
FROM iis_w3c_logs
WHERE cs_method = 'GET' AND sc_status = '200'
```

The following image shows the results of the query in the Athena Query Editor.

Results

	date_col	time_col	cs_uri_stem	time_taken
1	2020-01-19	22:48:39	/default.html	157
2	2020-01-19	22:49:40	/index.html	164
3	2020-01-19	22:50:12	/image.gif	358
4	2020-01-19	22:51:44	/faq.html	288

Combine the date and time fields

The space delimited `date` and `time` fields are separate entries in the log source data, but you can combine them into a timestamp if you want. Use the [concat()](#) and [date_parse()](#) functions in a [SELECT](#) or [CREATE TABLE AS SELECT](#) query to concatenate and convert the date and time columns into timestamp format. The following example uses a CTAS query to create a new table with a `derived_timestamp` column.

```
CREATE TABLE iis_w3c_logs_w_timestamp AS
SELECT
  date_parse(concat(date_col,' ', time_col),'%Y-%m-%d %H:%i:%s') as derived_timestamp,
  c_ip,
  s_ip,
  cs_method,
  cs_uri_stem,
  sc_status,
  sc_bytes,
```

```
  cs_bytes,
  time_taken,
  cs_version
FROM iis_w3c_logs
```

After the table is created, you can query the new timestamp column directly, as in the following example.

```
SELECT derived_timestamp, cs_uri_stem, time_taken
FROM iis_w3c_logs_w_timestamp
WHERE cs_method = 'GET' AND sc_status = '200'
```

The following image shows the results of the query.

Results

	derived_timestamp	cs_uri_stem	time_taken
1	2020-01-19 22:48:39.000	/default.html	157
2	2020-01-19 22:49:40.000	/index.html	164
3	2020-01-19 22:50:12.000	/image.gif	358
4	2020-01-19 22:51:44.000	/faq.html	288

Query IIS log file format

Unlike the W3C extended format, the IIS log file format has a fixed set of fields and includes a comma as a delimiter. The LazySimpleSerDe treats the comma as the delimiter and the space after the comma as the beginning of the next field.

The following example shows sample data in the IIS log file format.

```
203.0.113.15, -, 2020-02-24, 22:48:38, W3SVC2, SERVER5, 198.51.100.4, 254, 501, 488,
 200, 0, GET, /index.htm, -,
203.0.113.4, -, 2020-02-24, 22:48:39, W3SVC2, SERVER6, 198.51.100.6, 147, 411, 388,
 200, 0, GET, /about.html, -,
203.0.113.11, -, 2020-02-24, 22:48:40, W3SVC2, SERVER7, 198.51.100.18, 170, 531, 468,
 200, 0, GET, /image.png, -,
203.0.113.8, -, 2020-02-24, 22:48:41, W3SVC2, SERVER8, 198.51.100.14, 125, 711, 868,
 200, 0, GET, /intro.htm, -,
```

Create a table in Athena for IIS log files

To query your IIS log file format logs in Amazon S3, you first create a table schema so that Athena can read the log data.

To create a table in Athena for IIS log file format logs

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Paste the following DDL statement into the Athena console, noting the following points:
   a. To specify the comma delimiter, use `FIELDS TERMINATED BY ','`.
   b. Modify the values in LOCATION 's3://amzn-s3-demo-bucket/*iis-log-file-folder*/' to point to your IIS log format log files in Amazon S3.

```
CREATE EXTERNAL TABLE `iis_format_logs`(
client_ip_address string,
user_name string,
request_date string,
request_time string,
service_and_instance string,
server_name string,
server_ip_address string,
time_taken_millisec string,
client_bytes_sent string,
server_bytes_sent string,
service_status_code string,
windows_status_code string,
request_type string,
target_of_operation string,
script_parameters string
   )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/iis-log-file-folder/'
```

3. Run the query in the Athena console to register the `iis_format_logs` table. When the query completes, the logs are ready for you to query from Athena.

Example IIS log format select query

The following example query selects the request date, request time, request target, and time taken in milliseconds from the table `iis_format_logs`. The `WHERE` clause filters for cases in which the request type is `GET` and the HTTP status code is `200` (successful). In the query, note that the leading spaces in `' GET'` and `' 200'` are required to make the query successful.

```
SELECT request_date, request_time, target_of_operation, time_taken_millisec
FROM iis_format_logs
WHERE request_type = ' GET' AND service_status_code = ' 200'
```

The following image shows the results of the query of the sample data.

Results

	request_date	request_time	target_of_operation	time_taken_millisec
1	2020-02-24	22:48:38	/index.htm	254
2	2020-02-24	22:48:39	/about.html	147
3	2020-02-24	22:48:40	/image.png	170
4	2020-02-24	22:48:41	/intro.htm	125

Query NCSA log file format

IIS also uses the NCSA logging format, which has a fixed number of fields in ASCII text format separated by spaces. The structure is similar to the common log format used for Apache access logs. Fields in the NCSA common log data format include the client IP address, client ID (not typically used), domain\user ID, request received timestamp, text of the client request, server status code, and size of the object returned to the client.

The following example shows data in the NCSA common log format as documented for IIS.

```
198.51.100.7 - ExampleCorp\Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200
 232
198.51.100.14 - AnyCompany\Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html
 HTTP/1.1" 200 2165
```

```
198.51.100.22 - ExampleCorp\Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html
 HTTP/1.1" 200 1287
198.51.100.9 - AnyCompany\Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1"
 404 230
198.51.100.2 - ExampleCorp\Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1"
 404 30
198.51.100.13 - AnyCompany\Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html
 HTTP/1.1" 200 1608
198.51.100.11 - ExampleCorp\Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html
 HTTP/1.1" 200 1344
```

Create a table in Athena for IIS NCSA logs

For your `CREATE TABLE` statement, you can use the Grok SerDe and a grok pattern similar to the one for Apache web server logs. Unlike Apache logs, the grok pattern uses `%{DATA:user_id}` for the third field instead of `%{USERNAME:user_id}` to account for the presence of the backslash in `domain\user_id`. For more information about using the Grok SerDe, see Writing grok custom classifiers in the *AWS Glue Developer Guide*.

To create a table in Athena for IIS NCSA web server logs

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. Paste the following DDL statement into the Athena Query Editor. Modify the values in `LOCATION 's3://amzn-s3-demo-bucket/iis-ncsa-logs/'` to point to your IIS NCSA logs in Amazon S3.

```
CREATE EXTERNAL TABLE iis_ncsa_logs(
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{DATA:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
```

```
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://amzn-s3-demo-bucket/iis-ncsa-logs/';
```

3. Run the query in the Athena console to register the `iis_ncsa_logs` table. When the query completes, the logs are ready for you to query from Athena.

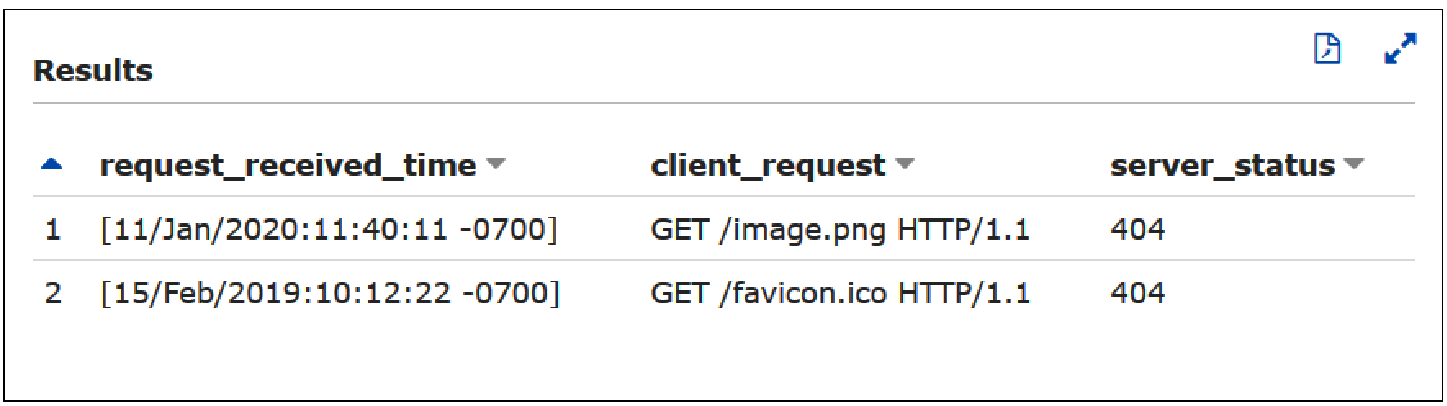
Example select queries for IIS NCSA logs

Example – Filtering for 404 errors

The following example query selects the request received time, text of the client request, and server status code from the `iis_ncsa_logs` table. The `WHERE` clause filters for HTTP status code `404` (page not found).

```
SELECT request_received_time, client_request, server_status
FROM iis_ncsa_logs
WHERE server_status = '404'
```

The following image shows the results of the query in the Athena Query Editor.



Results

	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

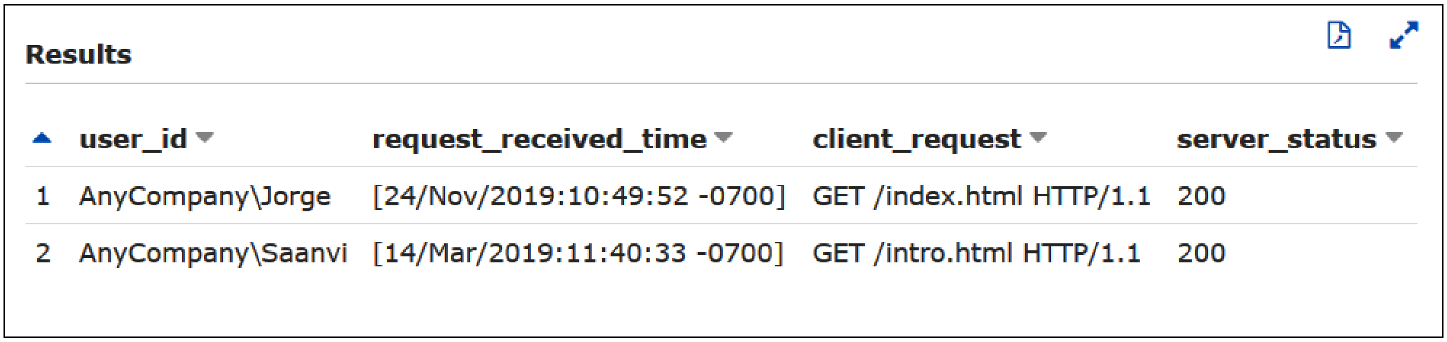
Example – Filtering for successful requests from a particular domain

The following example query selects the user ID, request received time, text of the client request, and server status code from the `iis_ncsa_logs` table. The `WHERE` clause filters for requests with HTTP status code `200` (successful) from users in the `AnyCompany` domain.

```
SELECT user_id, request_received_time, client_request, server_status
```

```
FROM iis_ncsa_logs
WHERE server_status = '200' AND user_id LIKE 'AnyCompany%'
```

The following image shows the results of the query in the Athena Query Editor.



Results

	user_id	request_received_time	client_request	server_status
1	AnyCompany\Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
2	AnyCompany\Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200

Use Athena ACID transactions

The term "ACID transactions" refers to a set of properties (atomicity, consistency, isolation, and durability) that ensure data integrity in database transactions. ACID transactions enable multiple users to concurrently and reliably add and delete Amazon S3 objects in an atomic manner, while isolating any existing queries by maintaining read consistency for queries against the data lake. Athena ACID transactions add single-table support for insert, delete, update, and time travel operations to the Athena SQL data manipulation language (DML). You and multiple concurrent users can use Athena ACID transactions to make reliable, row-level modifications to Amazon S3 data. Athena transactions automatically manage locking semantics and coordination and do not require a custom record locking solution.

Athena ACID transactions and familiar SQL syntax simplify updates to your business and regulatory data. For example, to respond to a data erasure request, you can perform a SQL `DELETE` operation. To make manual record corrections, you can use a single `UPDATE` statement. To recover data that was recently deleted, you can issue time travel queries using a `SELECT` statement.

Because they are built on shared table formats, Athena ACID transactions are compatible with other services and engines such as Amazon EMR and Apache Spark that also support shared table formats.

Athena transactions are available through the Athena console, API operations, and ODBC and JDBC drivers.

Topics

- Query Linux Foundation Delta Lake tables
- Query Apache Hudi datasets
- Query Apache Iceberg tables

Query Linux Foundation Delta Lake tables

Linux Foundation Delta Lake is a table format for big data analytics. You can use Amazon Athena to read Delta Lake tables stored in Amazon S3 directly without having to generate manifest files or run the `MSCK REPAIR` statement.

The Delta Lake format stores the minimum and maximum values per column of each data file. The Athena implementation makes use of this information to enable file-skipping on predicates to eliminate unwanted files from consideration.

Considerations and limitations

Delta Lake support in Athena has the following considerations and limitations:

- **Tables with AWS Glue catalog only** – Native Delta Lake support is supported only through tables registered with AWS Glue. If you have a Delta Lake table that is registered with another metastore, you can still keep it and treat it as your primary metastore. Because Delta Lake metadata is stored in the file system (for example, in Amazon S3) rather than in the metastore, Athena requires only the location property in AWS Glue to read from your Delta Lake tables.
- **V3 engine only** – Delta Lake queries are supported only on Athena engine version 3. You must ensure that the workgroup you create is configured to use Athena engine version 3.
- **No time travel support** – There is no support for queries that use Delta Lake's time travel capabilities.
- **Read only** – Write DML statements like `UPDATE`, `INSERT`, or `DELETE` are not supported.
- **Lake Formation support** – Lake Formation integration is available for Delta Lake tables with their schema in sync with AWS Glue. For more information, see Using AWS Lake Formation with Amazon Athena and Set up permissions for a Delta Lake table in the *AWS Lake Formation Developer Guide*.
- **Limited DDL support** – The following DDL statements are supported: `CREATE EXTERNAL TABLE`, `SHOW COLUMNS`, `SHOW TBLPROPERTIES`, `SHOW PARTITIONS`, `SHOW CREATE TABLE`, and `DESCRIBE`. For information on using the `CREATE EXTERNAL TABLE` statement, see the Get started section.

- **Skipping S3 Glacier objects not supported** – If objects in the Linux Foundation Delta Lake table are in an Amazon S3 Glacier storage class, setting the `read_restored_glacier_objects` table property to `false` has no effect.

  For example, suppose you issue the following command:

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  For Iceberg and Delta Lake tables, the command produces the error Unsupported table property key: read_restored_glacier_objects. For Hudi tables, the `ALTER TABLE` command does not produce an error, but Amazon S3 Glacier objects are still not skipped. Running `SELECT` queries after the `ALTER TABLE` command continues to return all objects.

Delta Lake versioning and Athena

Athena does not use the versioning listed in the Delta Lake documentation. To determine whether your Delta Lake tables are compatible with Athena, consider the following two characteristics:

- **Reader version** – Every Delta Lake table has a reader version. Currently, this is a number between 1 and 3. Queries that include a table with a reader version that Athena does not support will fail.
- **Table features** – Every Delta Lake table also can declare a set of reader/writer features. Because Athena's support of Delta Lake is read-only, table writer feature compatibility does not apply. However, queries on tables with unsupported table reader features will fail.

The following table shows the Delta Lake reader versions and Delta Lake table reader features that Athena supports.

Query type	Supported reader versions	Supported reader features
DQL (SELECT statements)	<= 3	Column mapping, timestamp Ntz, deletion vectors
DDL	<= 1	Not applicable. Reader features can be declared only on tables with a reader version of 2 or greater.

- For a list of Delta Lake table features, see Valid feature names in table features on GitHub.com
- For a list of Delta Lake features by protocol version, see Features by protocol version on GitHub.com.

To create a Delta Lake table in Athena with a reader version greater than 1, see Synchronize Delta Lake metadata.

Topics

Supported column data types

This section describes the supported data types for non-partition and partition columns.

Supported non-partition column data types

For non-partition columns, all data types that Athena supports except CHAR are supported (CHAR is not supported in the Delta Lake protocol itself). Supported data types include:

```
boolean
tinyint
smallint
integer
bigint
double
float
decimal
varchar
string
binary
date
timestamp
array
map
```

```
struct
```

Supported partition column data types

For partition columns, Athena supports tables with the following data types:

```
boolean
integer
smallint
tinyint
bigint
decimal
float
double
date
timestamp
varchar
```

For more information about the data types in Athena, see Data types in Amazon Athena.

Get started

To be queryable, your Delta Lake table must exist in AWS Glue. If your table is in Amazon S3 but not in AWS Glue, run a `CREATE EXTERNAL TABLE` statement using the following syntax. If your table already exists in AWS Glue (for example, because you are using Apache Spark or another engine with AWS Glue), you can skip this step.

```
CREATE EXTERNAL TABLE
  [db_name.]table_name
  LOCATION 's3://amzn-s3-demo-bucket/your-folder/'
  TBLPROPERTIES ('table_type' = 'DELTA')
```

Note the omission of column definitions, SerDe library, and other table properties. Unlike traditional Hive tables, Delta Lake table metadata are inferred from the Delta Lake transaction log and synchronized directly to AWS Glue.

> **Note**
>
> For Delta Lake tables, `CREATE TABLE` statements that include more than the `LOCATION` and `table_type` property are not allowed.

Query Delta Lake tables with SQL

To query a Delta Lake table, use standard SQL SELECT syntax:

```
[ WITH with_query [, ...] ]SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
[ LIMIT [ count | ALL ] ]
```

For more information about SELECT syntax, see SELECT in the Athena documentation.

The Delta Lake format stores the minimum and maximum values per column of each data file. Athena makes use of this information to enable file skipping on predicates to eliminate unnecessary files from consideration.

Synchronize Delta Lake metadata

Athena synchronizes table metadata, including schema, partition columns, and table properties, to AWS Glue if you use Athena to create your Delta Lake table. As time passes, this metadata can lose its synchronization with the underlying table metadata in the transaction log. To keep your table up to date, you can choose one of the following options:

- Use the AWS Glue crawler for Delta Lake tables. For more information, see Introducing native Delta Lake table support with AWS Glue crawlers in the *AWS Big Data Blog* and Scheduling an AWS Glue crawler in the AWS Glue Developer Guide.
- Drop and recreate the table in Athena.
- Use the SDK, CLI, or AWS Glue console to manually update the schema in AWS Glue.

Note that the following features require your AWS Glue schema to always have the same schema as the transaction log:

- Lake Formation
- Views

- Row and column filters

If your workflow does not require any of this functionality, and you prefer not to maintain this compatibility, you can use `CREATE TABLE` DDL in Athena and then add the Amazon S3 path as a SerDe parameter in AWS Glue.

Create a Delta Lake table using the Athena and AWS Glue consoles

You can use the following procedure to create a Delta Lake table with the Athena and AWS Glue consoles.

To create a Delta Lake table using the Athena and AWS Glue consoles

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena query editor, use the following DDL to create your Delta Lake table. Note that when using this method, the value for `TBLPROPERTIES` must be `'spark.sql.sources.provider' = 'delta'` and not `'table_type' = 'delta'`.

   Note that this same schema (with a single of column named `col` of type `array<string>`) is inserted when you use Apache Spark (Athena for Apache Spark) or most other engines to create your table.

   ```
   CREATE EXTERNAL TABLE
      [db_name.]table_name(col array<string>)
      LOCATION 's3://amzn-s3-demo-bucket/your-folder/'
      TBLPROPERTIES ('spark.sql.sources.provider' = 'delta')
   ```
3. Open the AWS Glue console at https://console.aws.amazon.com/glue/.
4. In the navigation pane, choose **Data Catalog**, **Tables**.
5. In the list of tables, choose the link for your table.
6. On the page for the table, choose **Actions**, **Edit table**.
7. In the **Serde parameters** section, add the key **`path`** with the value **`s3://amzn-s3-demo-bucket/`*`your-folder`*`/`**.
8. Choose **Save**.

Create a Delta Lake table using the AWS CLI

To create a Delta Lake table using the AWS CLI, enter a command like the following.

```
aws glue create-table --database-name dbname \
    --table-input '{"Name" : "tablename", "StorageDescriptor":{
            "Columns" : [
                {
                    "Name": "col",
                    "Type": "array<string>"
                }
            ],
            "Location" : "s3://amzn-s3-demo-bucket/<prefix>/",
            "SerdeInfo" : {
                "Parameters" : {
                    "serialization.format" : "1",
                    "path" : "s3://amzn-s3-demo-bucket/<prefix>/"
                }
            }
        },
        "PartitionKeys": [],
        "TableType": "EXTERNAL_TABLE",
        "Parameters": {
            "EXTERNAL": "TRUE",
            "spark.sql.sources.provider": "delta"
        }
    }'
```

Additional resources

For a discussion of using Delta Lake tables with AWS Glue and querying them with Athena, see Handle UPSERT data operations using open-source Delta Lake and AWS Glue in the *AWS Big Data Blog*.

Query Apache Hudi datasets

Apache Hudi is an open-source data management framework that simplifies incremental data processing. Record-level insert, update, upsert, and delete actions are processed much more granularly, reducing overhead. `Upsert` refers to the ability to insert records into an existing dataset if they do not already exist or to update them if they do.

Hudi handles data insertion and update events without creating many small files that can cause performance issues for analytics. Apache Hudi automatically tracks changes and merges files so that they remain optimally sized. This avoids the need to build custom solutions that monitor and re-write many small files into fewer large files.

Hudi datasets are suitable for the following use cases:

- Complying with privacy regulations like General data protection regulation (GDPR) and California consumer privacy act (CCPA) that enforce people's right to remove personal information or change how their data is used.
- Working with streaming data from sensors and other Internet of Things (IoT) devices that require specific data insertion and update events.
- Implementing a change data capture (CDC) system.

Data sets managed by Hudi are stored in Amazon S3 using open storage formats. Currently, Athena can read compacted Hudi datasets but not write Hudi data. Athena supports Hudi version 0.14.0 with Athena engine version 3. This is subject to change. Athena cannot guarantee read compatibility with tables that are created with later versions of Hudi. For information about Athena engine versioning, see Athena engine versioning. For more information about Hudi features and versioning, see the Hudi documentation on the Apache website.

A Hudi dataset can be one of the following types:

- **Copy on Write (CoW)** – Data is stored in a columnar format (Parquet), and each update creates a new version of files during a write.
- **Merge on Read (MoR)** – Data is stored using a combination of columnar (Parquet) and row-based (Avro) formats. Updates are logged to row-based `delta` files and are compacted as needed to create new versions of the columnar files.

With CoW datasets, each time there is an update to a record, the file that contains the record is rewritten with the updated values. With a MoR dataset, each time there is an update, Hudi writes only the row for the changed record. MoR is better suited for write- or change-heavy workloads with fewer reads. CoW is better suited for read-heavy workloads on data that change less frequently.

Hudi provides three query types for accessing the data:

- **Snapshot queries** – Queries that see the latest snapshot of the table as of a given commit or compaction action. For MoR tables, snapshot queries expose the most recent state of the table by merging the base and delta files of the latest file slice at the time of the query.
- **Incremental queries** – Queries only see new data written to the table, since a given commit/compaction. This effectively provides change streams to enable incremental data pipelines.

- **Read optimized queries –** For MoR tables, queries see the latest data compacted. For CoW tables, queries see the latest data committed.

The following table shows the possible Hudi query types for each table type.

Table type	Possible Hudi query types
Copy On Write	snapshot, incremental
Merge On Read	snapshot, incremental, read optimized

Currently, Athena supports snapshot queries and read optimized queries, but not incremental queries. On MoR tables, all data exposed to read optimized queries are compacted. This provides good performance but does not include the latest delta commits. Snapshot queries contain the freshest data but incur some computational overhead, which makes these queries less performant.

For more information about the tradeoffs between table and query types, see Table & Query Types in the Apache Hudi documentation.

Hudi terminology change: Views are now queries

Starting in Apache Hudi release version 0.5.1, what were formerly called views are now called queries. The following table summarizes the changes between the old and new terms.

Old term	New term
CoW: read optimized view MoR: realtime view	Snapshot queries
Incremental view	Incremental query
MoR read optimized view	Read optimized query

Topics

Considerations and limitations

When you use Athena to read Apache Hudi tables, consider the following points.

- **Incremental queries** – Athena does not support incremental queries.
- **CTAS** – Athena does not support CTAS or INSERT INTO on Hudi data. If you would like Athena support for writing Hudi datasets, send feedback to `<athena-feedback@amazon.com>`.

  For more information about writing Hudi data, see the following resources:

  - Working with a Hudi dataset in the Amazon EMR Release Guide.
  - Writing Data in the Apache Hudi documentation.
- **MSCK REPAIR TABLE** – Using MSCK REPAIR TABLE on Hudi tables in Athena is not supported. If you need to load a Hudi table not created in AWS Glue, use ALTER TABLE ADD PARTITION.
- **Skipping S3 Glacier objects not supported** – If objects in the Apache Hudi table are in an Amazon S3 Glacier storage class, setting the `read_restored_glacier_objects` table property to `false` has no effect.

  For example, suppose you issue the following command:

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  For Iceberg and Delta Lake tables, the command produces the error Unsupported table property key: read_restored_glacier_objects. For Hudi tables, the `ALTER TABLE` command does not produce an error, but Amazon S3 Glacier objects are still not skipped. Running `SELECT` queries after the `ALTER TABLE` command continues to return all objects.
- **Timestamp queries** – Currently, queries that attempt to read timestamp columns in Hudi real time tables either fail or produce empty results. This limitation applies only to queries that read

a timestamp column. Queries that include only non-timestamp columns from the same table succeed.

Failed queries return a message similar to the following:

GENERIC_INTERNAL_ERROR: class org.apache.hadoop.io.ArrayWritable cannot be cast to class org.apache.hadoop.hive.serde2.io.TimestampWritableV2 (org.apache.hadoop.io.ArrayWritable and org.apache.hadoop.hive.serde2.io.TimestampWritableV2 are in unnamed module of loader io.trino.server.PluginClassLoader @75c67992)

Copy on write (CoW) create table examples

If you have Hudi tables already created in AWS Glue, you can query them directly in Athena. When you create partitioned Hudi tables in Athena, you must run `ALTER TABLE ADD PARTITION` to load the Hudi data before you can query it.

Nonpartitioned CoW table

The following example creates a nonpartitioned CoW table in Athena.

```
CREATE EXTERNAL TABLE `non_partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/non_partition_cow/'
```

Partitioned CoW table

The following example creates a partitioned CoW table in Athena.

```
CREATE EXTERNAL TABLE `partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/partition_cow/'
```

The following `ALTER TABLE ADD PARTITION` example adds two partitions to the example `partition_cow` table.

```
ALTER TABLE partition_cow ADD
  PARTITION (event_type = 'one') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_cow/one/'
  PARTITION (event_type = 'two') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_cow/two/'
```

Merge on read (MoR) create table examples

Hudi creates two tables in the metastore for MoR: a table for snapshot queries, and a table for read optimized queries. Both tables are queryable. In Hudi versions prior to 0.5.1, the table for read optimized queries had the name that you specified when you created the table. Starting in Hudi version 0.5.1, the table name is suffixed with `_ro` by default. The name of the table for snapshot queries is the name that you specified appended with `_rt`.

Nonpartitioned merge on read (MoR) table

The following example creates a nonpartitioned MoR table in Athena for read optimized queries. Note that read optimized queries use the input format `HoodieParquetInputFormat`.

```
CREATE EXTERNAL TABLE `nonpartition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/nonpartition_mor/'
```

The following example creates a nonpartitioned MoR table in Athena for snapshot queries. For snapshot queries, use the input format `HoodieParquetRealtimeInputFormat`.

```
CREATE EXTERNAL TABLE `nonpartition_mor_rt`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
```

```
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/nonpartition_mor/'
```

Partitioned merge on read (MoR) table

The following example creates a partitioned MoR table in Athena for read optimized queries.

```
CREATE EXTERNAL TABLE `partition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/partition_mor/'
```

The following `ALTER TABLE ADD PARTITION` example adds two partitions to the example `partition_mor` table.

```
ALTER TABLE partition_mor ADD
  PARTITION (event_type = 'one') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_mor/two/'
```

The following example creates a partitioned MoR table in Athena for snapshot queries.

```
CREATE EXTERNAL TABLE `partition_mor_rt`(
```

```
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/folder/partition_mor/'
```

Similarly, the following `ALTER TABLE ADD PARTITION` example adds two partitions to the example `partition_mor_rt` table.

```
ALTER TABLE partition_mor_rt ADD
  PARTITION (event_type = 'one') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://amzn-s3-demo-bucket/folder/
partition_mor/two/'
```

Use Hudi metadata for improved performance

The Apache Hudi has a metadata table that contains indexing features for improved performance like file listing, data skipping using column statistics, and a bloom filter based index.

Of these features, Athena currently supports only the file listing index. The file listing index eliminates file system calls like "list files" by fetching the information from an index which maintains a partition to files mapping. This removes the need to recursively list each and every partition under the table path to get a view of the file system. When you work with large datasets, this indexing drastically reduces the latency that would otherwise occur when getting the list of files during writes and queries. It also avoids bottlenecks like request limits throttling on Amazon S3 `LIST` calls.

Note

Athena does not support data skipping or bloom filter indexing at this time.

Enabling the Hudi metadata table

Metadata table based file listing is disabled by default. To enable the Hudi metadata table and the related file listing functionality, set the `hudi.metadata-listing-enabled` table property to `TRUE`.

Example

The following `ALTER TABLE SET TBLPROPERTIES` example enables the metadata table on the example `partition_cow` table.

```
ALTER TABLE partition_cow SET TBLPROPERTIES('hudi.metadata-listing-enabled'='TRUE')
```

Use bootstrap generated metadata

Starting in Apache Hudi version 0.6.0, the bootstrap operation feature provides better performance with existing Parquet datasets. Instead of rewriting the dataset, a bootstrap operation can generate metadata only, leaving the dataset in place.

You can use Athena to query tables from a bootstrap operation just like other tables based on data in Amazon S3. In your `CREATE TABLE` statement, specify the Hudi table path in your `LOCATION` clause.

For more information about creating Hudi tables using the bootstrap operation in Amazon EMR, see the article [New features from Apache Hudi available in Amazon EMR] in the AWS Big Data Blog.

Additional resources

For additional resources on using Apache Hudi with Athena, see the following resources.

Video

The following video shows how you can use Amazon Athena to query a read-optimized Apache Hudi dataset in your Amazon S3-based data lake.

Query Apache Hudi datasets using Amazon Athena

Blog posts

The following AWS Big Data Blog posts include descriptions of how you can use Apache Hudi with Athena.

- Use AWS Data Exchange to seamlessly share Apache Hudi datasets
- Create an Apache Hudi-based near-real-time transactional data lake using AWS DMS, Amazon Kinesis, AWS Glue streaming ETL, and data visualization using QuickSight
- For information about using AWS Glue custom connectors and AWS Glue 2.0 jobs to create an Apache Hudi table that you can query with Athena, see Writing to Apache Hudi tables using AWS Glue custom connector.
- For an article about using Apache Hudi, AWS Glue, and Amazon Athena to build a data processing framework for a data lake, see Simplify operational data processing in data lakes using AWS Glue and Apache Hudi.

Query Apache Iceberg tables

You can use Athena to perform read, time travel, write, and DDL queries on Apache Iceberg tables.

Apache Iceberg is an open table format for very large analytic datasets. Iceberg manages large collections of files as tables, and it supports modern analytical data lake operations such as record-level insert, update, delete, and time travel queries. The Iceberg specification allows seamless table evolution such as schema and partition evolution and is designed for optimized usage on Amazon S3. Iceberg also helps guarantee data correctness under concurrent write scenarios.

For more information about Apache Iceberg, see https://iceberg.apache.org/.

Considerations and limitations

Athena support for Iceberg tables has the following considerations and limitations:

- **Iceberg version support** – Athena supports Apache Iceberg version 1.4.2.
- **Tables with AWS Glue catalog only** – Only Iceberg tables created against the AWS Glue catalog based on specifications defined by the open source glue catalog implementation are supported from Athena.

- **Table locking support by AWS Glue only –** Unlike the open source Glue catalog implementation, which supports plug-in custom locking, Athena supports AWS Glue optimistic locking only. Using Athena to modify an Iceberg table with any other lock implementation will cause potential data loss and break transactions.
- **Supported file formats –** Athena engine version 3 supports the following Iceberg file formats.
  - Parquet
  - ORC
  - Avro
- **Iceberg restricted metadata –** Lake Formation does not evaluate the Iceberg metadata tables. Hence, the Iceberg metadata tables are restricted if there are any Lake Formation row or cell filters present on the base table or if you do not have permissions to view all columns in the base table. For such cases, when you query the `$partitions`, `$files`, `$manifests`, and `$snapshots` Iceberg metadata tables, it fails and you get an `AccessDeniedException` error. Additionally, the metadata column `$path` has the same Lake Formation restrictions and fails when selected by the query. All other metadata tables can be queried regardless of the Lake Formation filters. For more information, see Metadata tables.
- **Iceberg v2 tables –** Athena only creates and operates on Iceberg v2 tables. For the difference between v1 and v2 tables, see Format version changes in the Apache Iceberg documentation.
- **Display of time types without time zone –** The time and timestamp without time zone types are displayed in UTC. If the time zone is unspecified in a filter expression on a time column, UTC is used.
- **Timestamp related data precision –** Although Iceberg supports microsecond precision for the timestamp data type, Athena supports only millisecond precision for timestamps in both reads and writes. For data in time related columns that is rewritten during manual compaction operations, Athena retains only millisecond precision.
- **Unsupported operations –** The following Athena operations are not supported for Iceberg tables.
  - ALTER TABLE SET LOCATION
- **Views –** Use `CREATE VIEW` to create Athena views as described in Work with views. If you are interested in using the Iceberg view specification to create views, contact athena-feedback@amazon.com.
- **TTF management commands not supported in AWS Lake Formation –** Although you can use Lake Formation to manage read access permissions for TransactionTable Formats (TTFs) like Apache Iceberg, Apache Hudi, and Linux Foundation Delta Lake, you cannot use Lake Formation

to manage permissions for operations like VACUUM, MERGE, UPDATE or OPTIMIZE with these table formats. For more information about Lake Formation integration with Athena, see Using AWS Lake Formation with Amazon Athena in the *AWS Lake Formation Developer Guide*.

- **Partitioning by nested fields** – Partitioning by nested fields is not supported. Attempting to do so produces the message NOT_SUPPORTED: Partitioning by nested field is unsupported: *column_name*.*nested_field_name*.
- **Skipping S3 Glacier objects not supported** – If objects in the Apache Iceberg table are in an Amazon S3 Glacier storage class, setting the `read_restored_glacier_objects` table property to `false` has no effect.

  For example, suppose you issue the following command:

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  For Iceberg and Delta Lake tables, the command produces the error Unsupported table property key: read_restored_glacier_objects. For Hudi tables, the ALTER TABLE command does not produce an error, but Amazon S3 Glacier objects are still not skipped. Running SELECT queries after the ALTER TABLE command continues to return all objects.

If you would like Athena to support a particular feature, send feedback to athena-feedback@amazon.com.

Topics

Create Iceberg tables

To create an Iceberg table for use in Athena, you can use a CREATE TABLE statement as documented on this page, or you can use an AWS Glue crawler.

Use a CREATE TABLE statement

Athena creates Iceberg v2 tables. For the difference between v1 and v2 tables, see Format version changes in the Apache Iceberg documentation.

Athena CREATE TABLE creates an Iceberg table with no data. You can query a table from external systems such as Apache Spark directly if the table uses the Iceberg open source glue catalog. You do not have to create an external table.

> **Warning**
>
> Running CREATE EXTERNAL TABLE results in the error message External keyword not supported for table type ICEBERG.

To create an Iceberg table from Athena, set the 'table_type' table property to 'ICEBERG' in the TBLPROPERTIES clause, as in the following syntax summary.

```
CREATE TABLE
  [db_name.]table_name (col_name data_type [COMMENT col_comment] [, ...] )
  [PARTITIONED BY (col_name | transform, ... )]
  LOCATION 's3://amzn-s3-demo-bucket/your-folder/'
  TBLPROPERTIES ( 'table_type' ='ICEBERG' [, property_name=property_value] )
```

For information about the data types that you can query in Iceberg tables, see Supported data types for Iceberg tables in Athena.

Use partitions

To create Iceberg tables with partitions, use PARTITIONED BY syntax. Columns used for partitioning must be specified in the columns declarations first. Within the PARTITIONED BY clause, the column type must not be included. You can also define partition transforms in CREATE TABLE syntax. To specify multiple columns for partitioning, separate the columns with the comma (,) character, as in the following example.

```
CREATE TABLE iceberg_table (id bigint, data string, category string)
```

```
PARTITIONED BY (category, bucket(16, id))
LOCATION 's3://amzn-s3-demo-bucket/your-folder/'
TBLPROPERTIES ( 'table_type' = 'ICEBERG' )
```

The following table shows the available partition transform functions.

Function	Description	Supported types
`year(ts)`	Partition by year	`date`, `timestamp`
`month(ts)`	Partition by month	`date`, `timestamp`
`day(ts)`	Partition by day	`date`, `timestamp`
`hour(ts)`	Partition by hour	`timestamp`
`bucket(N, col)`	Partition by hashed value mod *N* buckets. This is the same concept as hash bucketing for Hive tables.	`int`, `long`, `decimal`, `date`, `timestamp` , `string`, `binary`
`truncate( L, col)`	Partition by value truncated to *L*	`int`, `long`, `decimal`, `string`

Athena supports Iceberg's hidden partitioning. For more information, see Iceberg's hidden partitioning in the Apache Iceberg documentation.

Specify table properties

This section describes table properties that you can specify as key-value pairs in the `TBLPROPERTIES` clause of the `CREATE TABLE` statement. Athena allows only a predefined list of key-value pairs in the table properties for creating or altering Iceberg tables. The following tables show the table properties that you can specify. For more information about the compaction options, see Optimize Iceberg tables in this documentation. If you would like Athena to support a specific open source table configuration property, send feedback to athena-feedback@amazon.com.

format

Description	File data format
Allowed property values	Supported file format and compression combinations vary by Athena engine version. For more information, see Use Iceberg table compression.
Default value	parquet

write_compression

Description	File compression codec
Allowed property values	Supported file format and compression combinations vary by Athena engine version. For more information, see Use Iceberg table compression.
Default value	Default write compression varies by Athena engine version. For more information, see Use Iceberg table compression.

optimize_rewrite_data_file_threshold

Description	Data optimization specific configuration. If there are fewer data files that require optimization than the given threshold, the files are not rewritten. This allows the accumulation of more data files to produce files closer to the target size and skip unnecessary computation for cost saving.
Allowed property values	A positive number. Must be less than 50.
Default value	5

optimize_rewrite_delete_file_threshold

Description	Data optimization specific configuration. If there are fewer delete files associated with a data file than the threshold, the data file is not rewritten. This allows the accumulation of more delete files for each data file for cost saving.
Allowed property values	A positive number. Must be less than 50.
Default value	2

vacuum_min_snapshots_to_keep

Description	Minimum number of snapshots to retain on a table's main branch. This value takes precedence over the `vacuum_max_snapshot_age_seconds` property. If the minimum remaining snapshots are older than the age specified by `vacuum_max_snapshot_age_seconds`, the snapshots are kept, and the value of `vacuum_max_snapshot_age_seconds` is ignored.
Allowed property values	A positive number.
Default value	1

vacuum_max_snapshot_age_seconds

Description	Maximum age of the snapshots to retain on the main branch. This value is ignored if the remaining minimum of snapshots specified by `vacuum_min_snapshots_to_keep` are older than the age specified. This table behavior property corresponds to the `history.expire.max-snapshot-age-ms` property in Apache Iceberg configuration.
Allowed property values	A positive number.

Default value	432000 seconds (5 days)

vacuum_max_metadata_files_to_keep

Description	The maximum number of previous metadata files to retain on the table's main branch.
Allowed property values	A positive number.
Default value	100

Example CREATE TABLE statement

The following example creates an Iceberg table that has three columns.

```
CREATE TABLE iceberg_table (
  id int,
  data string,
  category string)
PARTITIONED BY (category, bucket(16,id))
LOCATION 's3://amzn-s3-demo-bucket/iceberg-folder'
TBLPROPERTIES (
  'table_type'='ICEBERG',
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

Use CREATE TABLE AS SELECT (CTAS)

For information about creating an Iceberg table using the `CREATE TABLE AS` statement, see [CREATE TABLE AS](), with particular attention to the [CTAS table properties]() section.

Use an AWS Glue crawler

You can use an AWS Glue crawler to automatically register your Iceberg tables into the AWS Glue Data Catalog. If you want to migrate from another Iceberg catalog, you can create and schedule an AWS Glue crawler and provide the Amazon S3 paths where the Iceberg tables are located. You can specify the maximum depth of the Amazon S3 paths that the AWS Glue crawler can traverse. After you schedule an AWS Glue crawler, the crawler extracts schema information and updates the AWS Glue Data Catalog with the schema changes every time it runs. The AWS Glue crawler supports schema merging across snapshots and updates the latest metadata file location in the AWS Glue Data Catalog. For more information, see Data Catalog and crawlers in AWS Glue.

Query Iceberg table data

To query an Iceberg dataset, use a standard `SELECT` statement like the following. Queries follow the Apache Iceberg format v2 spec and perform merge-on-read of both position and equality deletes.

```
SELECT * FROM [db_name.]table_name [WHERE predicate]
```

To optimize query times, all predicates are pushed down to where the data lives.

For information about time travel and version travel queries, see Perform time travel and version travel queries.

Create and query views with Iceberg tables

To create and query Athena views on Iceberg tables, use `CREATE VIEW` views as described in Work with views.

Example:

```
CREATE VIEW view1 AS SELECT * FROM iceberg_table
```

```
SELECT * FROM view1
```

If you are interested in using the Iceberg view specification to create views, contact athena-feedback@amazon.com.

Query Iceberg table metadata

In a `SELECT` query, you can use the following properties after *`table_name`*to query Iceberg table metadata:

- **$files** – Shows a table's current data files.
- **$manifests** – Shows a table's current file manifests.

- **$history** – Shows a table's history.
- **$partitions** – Shows a table's current partitions.
- **$snapshots** – Shows a table's snapshots.
- **$refs** – Shows a table's references.

Examples

The following statement lists the files for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$files"
```

The following statement lists the manifests for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$manifests"
```

The following statement shows the history for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$history"
```

The following example shows the partitions for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$partitions"
```

The following example lists the snapshots for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$snapshots"
```

The following example shows the references for an Iceberg table.

```
SELECT * FROM "dbname"."tablename$refs"
```

Use Lake Formation fine-grained access control

Athena engine version 3 supports Lake Formation fine-grained access control with Iceberg tables, including column level and row level security access control. This access control works with time travel queries and with tables that have performed schema evolution. For more information, see Lake Formation fine-grained access control and Athena workgroups.

If you created your Iceberg table outside of Athena, use Apache Iceberg SDK version 0.13.0 or higher so that your Iceberg table column information is populated in the AWS Glue Data Catalog. If your Iceberg table does not contain column information in AWS Glue, you can use the Athena ALTER TABLE SET TBLPROPERTIES statement or the latest Iceberg SDK to fix the table and update the column information in AWS Glue.

Perform time travel and version travel queries

Each Apache Iceberg table maintains a versioned manifest of the Amazon S3 objects that it contains. Previous versions of the manifest can be used for time travel and version travel queries.

Time travel queries in Athena query Amazon S3 for historical data from a consistent snapshot as of a specified date and time. Version travel queries in Athena query Amazon S3 for historical data as of a specified snapshot ID.

Time travel queries

To run a time travel query, use `FOR TIMESTAMP AS OF` *`timestamp`* after the table name in the `SELECT` statement, as in the following example.

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF timestamp
```

The system time to be specified for traveling is either a timestamp or timestamp with a time zone. If not specified, Athena considers the value to be a timestamp in UTC time.

The following example time travel queries select CloudTrail data for the specified date and time.

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF TIMESTAMP '2020-01-01 10:00:00 UTC'
```

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

Version travel queries

To execute a version travel query (that is, view a consistent snapshot as of a specified version), use `FOR VERSION AS OF` *`version`* after the table name in the `SELECT` statement, as in the following example.

```
SELECT * FROM [db_name.]table_name FOR VERSION AS OF version
```

The *`version`* parameter is the `bigint` snapshot ID associated with an Iceberg table version.

The following example version travel query selects data for the specified version.

```
SELECT * FROM iceberg_table FOR VERSION AS OF 949530903748831860
```

> **Note**
>
> The `FOR SYSTEM_TIME AS OF` and `FOR SYSTEM_VERSION AS OF` clauses in Athena engine version 2 have been replaced by the `FOR TIMESTAMP AS OF` and `FOR VERSION AS OF` clauses in Athena engine version 3.

Retrieve the snapshot ID

You can use the Java SnapshotUtil class provided by Iceberg to retrieve the Iceberg snapshot ID, as in the following example.

```
import org.apache.iceberg.Table;
import org.apache.iceberg.aws.glue.GlueCatalog;
import org.apache.iceberg.catalog.TableIdentifier;
import org.apache.iceberg.util.SnapshotUtil;

import java.text.SimpleDateFormat;
import java.util.Date;

Catalog catalog = new GlueCatalog();

Map<String, String> properties = new HashMap<String, String>();
properties.put("warehouse", "s3://amzn-s3-demo-bucket/my-folder");
catalog.initialize("my_catalog", properties);

Date date = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").parse("2022/01/01 00:00:00");
long millis = date.getTime();

TableIdentifier name = TableIdentifier.of("db", "table");
Table table = catalog.loadTable(name);
long oldestSnapshotIdAfter2022 = SnapshotUtil.oldestAncestorAfter(table, millis);
```

Combine time and version travel

You can use time travel and version travel syntax in the same query to specify different timing and versioning conditions, as in the following example.

```
SELECT table1.*, table2.* FROM
  [db_name.]table_name FOR TIMESTAMP AS OF (current_timestamp - interval '1' day) AS
 table1
  FULL JOIN
  [db_name.]table_name FOR VERSION AS OF 5487432386996890161 AS table2
  ON table1.ts = table2.ts
  WHERE (table1.id IS NULL OR table2.id IS NULL)
```

Update Iceberg table data

You can manage Iceberg table data directly on Athena by using INSERT, UPDATE, and DELETE queries. Each data management transaction produces a new snapshot, which can be queried using time travel. The UPDATE and DELETE statements follow the Iceberg format v2 row-level position delete specification and enforce snapshot isolation.

Use the following commands to perform data management operations on Iceberg tables.

Topics

INSERT INTO

Inserts data into an Iceberg table. Athena Iceberg INSERT INTO is charged the same as current INSERT INTO queries for external Hive tables by the amount of data scanned. To insert data into an Iceberg table, use the following syntax, where *query* can be either VALUES (val1, val2, ...) or SELECT (col1, col2, …) FROM [*db_name*.]*table_name* WHERE *predicate*. For SQL syntax and semantic details, see INSERT INTO.

```
INSERT INTO [db_name.]table_name [(col1, col2, …)] query
```

The following examples insert values into the table iceberg_table.

```
INSERT INTO iceberg_table VALUES (1,'a','c1')
```

```
INSERT INTO iceberg_table (col1, col2, ...) VALUES (val1, val2, ...)
```

```
INSERT INTO iceberg_table SELECT * FROM another_table
```

DELETE

Athena Iceberg DELETE writes Iceberg position delete files to a table. This is known as a *merge-on-read* delete. In contrast to a *copy-on-write* delete, a merge-on-read delete is more efficient because it does not rewrite file data. When Athena reads Iceberg data, it merges the Iceberg position delete files with data files to produce the latest view of a table. To remove these position delete files, you can run the REWRITE DATA compaction action. DELETE operations are charged by the amount of data scanned. For syntax, see DELETE.

The following example deletes rows from `iceberg_table` that have `c3` as the value for `category`.

```
DELETE FROM iceberg_table WHERE category='c3'
```

UPDATE

Athena Iceberg UPDATE writes Iceberg position delete files and newly updated rows as data files in the same transaction. UPDATE can be imagined as a combination of INSERT INTO and DELETE. UPDATE operations are charged by the amount of data scanned. For syntax, see UPDATE.

The following example updates the specified values in the table `iceberg_table`.

```
UPDATE iceberg_table SET category='c2' WHERE category='c1'
```

MERGE INTO

Conditionally updates, deletes, or inserts rows into an Iceberg table. A single statement can combine update, delete, and insert actions. For syntax, see MERGE INTO.

> **Note**
>
> MERGE INTO is transactional and is supported only for Apache Iceberg tables in Athena engine version 3.

The following example deletes all customers from table t that are in the source table s.

```
MERGE INTO accounts t USING monthly_accounts_update s
ON t.customer = s.customer
WHEN MATCHED
THEN DELETE
```

The following example updates target table t with customer information from source table s. For customer rows in table t that have matching customer rows in table s, the example increments the purchases in table t. If table t has no match for a customer row in table s, the example inserts the customer row from table s into table t.

```
MERGE INTO accounts t USING monthly_accounts_update s
    ON (t.customer = s.customer)
    WHEN MATCHED
        THEN UPDATE SET purchases = s.purchases + t.purchases
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
              VALUES(s.customer, s.purchases, s.address)
```

The following example conditionally updates target table t with information from the source table s. The example deletes any matching target row for which the source address is Centreville. For all other matching rows, the example adds the source purchases and sets the target address to the source address. If there is no match in the target table, the example inserts the row from the source table.

```
MERGE INTO accounts t USING monthly_accounts_update s
    ON (t.customer = s.customer)
    WHEN MATCHED AND s.address = 'Centreville'
        THEN DELETE
    WHEN MATCHED
        THEN UPDATE
            SET purchases = s.purchases + t.purchases, address = s.address
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
              VALUES(s.customer, s.purchases, s.address)
```

Manage Iceberg tables

Athena supports the following table DDL operations for Iceberg tables.

Topics

ALTER TABLE RENAME

Renames a table.

Because the table metadata of an Iceberg table is stored in Amazon S3, you can update the database and table name of an Iceberg managed table without affecting underlying table information.

Synopsis

```
ALTER TABLE [db_name.]table_name RENAME TO [new_db_name.]new_table_name
```

Example

```
ALTER TABLE my_db.my_table RENAME TO my_db2.my_table2
```

ALTER TABLE SET TBLPROPERTIES

Adds properties to an Iceberg table and sets their assigned values.

In accordance with Iceberg specifications, table properties are stored in the Iceberg table metadata file rather than in AWS Glue. Athena does not accept custom table properties. Refer to the Specify table properties section for allowed key-value pairs. If you would like Athena to support a specific open source table configuration property, send feedback to athena-feedback@amazon.com.

Synopsis

```
ALTER TABLE [db_name.]table_name SET TBLPROPERTIES ('property_name' =
 'property_value' [ , ... ])
```

Example

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

ALTER TABLE UNSET TBLPROPERTIES

Drops existing properties from an Iceberg table.

Synopsis

```
ALTER TABLE [db_name.]table_name UNSET TBLPROPERTIES ('property_name' [ , ... ])
```

Example

```
ALTER TABLE iceberg_table UNSET TBLPROPERTIES ('write_compression')
```

DESCRIBE

Describes table information.

Synopsis

```
DESCRIBE [FORMATTED] [db_name.]table_name
```

When the `FORMATTED` option is specified, the output displays additional information such as table location and properties.

Example

```
DESCRIBE iceberg_table
```

DROP TABLE

Drops an Iceberg table.

> **Warning**
>
> Because Iceberg tables are considered managed tables in Athena, dropping an Iceberg table also removes all the data in the table.

Synopsis

```
DROP TABLE [IF EXISTS] [db_name.]table_name
```

Example

```
DROP TABLE iceberg_table
```

SHOW CREATE TABLE

Displays a CREATE TABLE DDL statement that can be used to recreate the Iceberg table in Athena. If Athena cannot reproduce the table structure (for example, because custom table properties are specified in the table), an UNSUPPORTED error is thrown.

Synopsis

```
SHOW CREATE TABLE [db_name.]table_name
```

Example

```
SHOW CREATE TABLE iceberg_table
```

SHOW TBLPROPERTIES

Shows one or more table properties of an Iceberg table. Only Athena-supported table properties are shown.

Synopsis

```
SHOW TBLPROPERTIES [db_name.]table_name [('property_name')]
```

Example

```
SHOW TBLPROPERTIES iceberg_table
```

Evolve Iceberg table schema

Iceberg schema updates are metadata-only changes. No data files are changed when you perform a schema update.

The Iceberg format supports the following schema evolution changes:

- **Add** – Adds a new column to a table or to a nested `struct`.
- **Drop** – Removes an existing column from a table or nested `struct`.
- **Rename** – Renames an existing column or field in a nested `struct`.
- **Reorder** – Changes the order of columns.
- **Type promotion** – Widens the type of a column, `struct` field, `map` key, `map` value, or `list` element. Currently, the following cases are supported for Iceberg tables:
  - integer to big integer
  - float to double
  - increasing the precision of a decimal type

You can use the DDL statements in this section to modify Iceberg table schema.

Topics

ALTER TABLE ADD COLUMNS

Adds one or more columns to an existing Iceberg table.

Synopsis

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name data_type [,...])
```

Examples

The following example adds a `comment` column of type `string` to an Iceberg table.

```
ALTER TABLE iceberg_table ADD COLUMNS (comment string)
```

The following example adds a `point` column of type `struct` to an Iceberg table.

```
ALTER TABLE iceberg_table
ADD COLUMNS (point struct<x: double, y: double>)
```

The following example adds a `points` column that is an array of structs to an Iceberg table.

```
ALTER TABLE iceberg_table
ADD COLUMNS (points array<struct<x: double, y: double>>)
```

ALTER TABLE DROP COLUMN

Drops a column from an existing Iceberg table.

Synopsis

```
ALTER TABLE [db_name.]table_name DROP COLUMN col_name
```

Example

```
ALTER TABLE iceberg_table DROP COLUMN userid
```

ALTER TABLE CHANGE COLUMN

Changes the name, type, order or comment of a column in an Iceberg table.

> **Note**
>
> `ALTER TABLE REPLACE COLUMNS` is not supported. Because `REPLACE COLUMNS` removes all columns and then adds new ones, it is not supported for Iceberg. `CHANGE COLUMN` is the preferred syntax for schema evolution.

Synopsis

```
ALTER TABLE [db_name.]table_name
  CHANGE [COLUMN] col_old_name col_new_name column_type
  [COMMENT col_comment] [FIRST|AFTER column_name]
```

Example

```
ALTER TABLE iceberg_table CHANGE comment blog_comment string AFTER id
```

SHOW COLUMNS

Shows the columns in a table.

Synopsis

```
SHOW COLUMNS (FROM|IN) [db_name.]table_name
```

Example

```
SHOW COLUMNS FROM iceberg_table
```

Perform other DDL operations on Iceberg tables

In addition to the schema evolution operations described in Evolve Iceberg table schema, you can also perform the following DDL operations on Apache Iceberg tables in Athena.

Database level operations

When you use DROP DATABASE with the CASCADE option , any Iceberg table data is also removed. The following DDL operations have no effect on Iceberg tables.

- CREATE DATABASE
- ALTER DATABASE SET DBPROPERTIES
- SHOW DATABASES
- SHOW TABLES
- SHOW VIEWS

Partition related operations

Because Iceberg tables use hidden partitioning, you do not have to work with physical partitions directly. As a result, Iceberg tables in Athena do not support the following partition-related DDL operations:

- SHOW PARTITIONS

- ALTER TABLE ADD PARTITION
- ALTER TABLE DROP PARTITION
- ALTER TABLE RENAME PARTITION

If you would like to see Iceberg partition evolution in Athena, send feedback to athena-feedback@amazon.com.

Unload Iceberg tables

Iceberg tables can be unloaded to files in a folder on Amazon S3. For information, see UNLOAD.

MSCK REPAIR

Because Iceberg tables keep track of table layout information, running MSCK REPAIR TABLE as one does with Hive tables is not necessary and is not supported.

Optimize Iceberg tables

As data accumulates into an Iceberg table, queries gradually become less efficient because of the increased processing time required to open files. Additional computational cost is incurred if the table contains delete files. In Iceberg, delete files store row-level deletes, and the engine must apply the deleted rows to query results.

To help optimize the performance of queries on Iceberg tables, Athena supports manual compaction as a table maintenance command. Compactions optimize the structural layout of the table without altering table content.

OPTIMIZE

The `OPTIMIZE` *`table`* `REWRITE DATA` compaction action rewrites data files into a more optimized layout based on their size and number of associated delete files. For syntax and table property details, see OPTIMIZE.

Example

The following example merges delete files into data files and produces files near the targeted file size where the value of `category` is `c1`.

```
OPTIMIZE iceberg_table REWRITE DATA USING BIN_PACK
  WHERE category = 'c1'
```

VACUUM

VACUUM performs snapshot expiration and orphan file removal. These actions reduce metadata size and remove files not in the current table state that are also older than the retention period specified for the table. For syntax details, see VACUUM.

Example

The following example uses a table property to configure the table iceberg_table to retain the last three days of data, then uses VACUUM to expire the old snapshots and remove the orphan files from the table.

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'vacuum_max_snapshot_age_seconds'='259200'
)

VACUUM iceberg_table
```

Supported data types for Iceberg tables in Athena

Athena can query Iceberg tables that contain the following data types:

```
binary
boolean
date
decimal
double
float
int
list
long
map
string
struct
timestamp without time zone
```

For more information about Iceberg table types, see the schemas page for Iceberg in the Apache documentation.

The following table shows the relationship between Athena data types and Iceberg table data types.

Iceberg type	Athena type	Notes
boolean	boolean	
-	tinyint	Not supported for Iceberg tables in Athena.
-	smallint	Not supported for Iceberg tables in Athena.
int	int	In Athena DML statements, this type is INTEGER.
long	bigint	
double	double	
float	float	
decimal(P , S)	decimal(P , S)	P is precision, S is scale.
-	char	Not supported for Iceberg tables in Athena.
string	string	In Athena DML statements, this type is VARCHAR.
binary	binary	
date	date	
time	-	Only Iceberg timestamp (without time zone) is supported for Athena Iceberg DDL statements like CREATE TABLE, but all timestamp types can be queried through Athena.
timestamp	timestamp	
timestamptz	timestamptz	
list<E>	array	
map<K,V>	map	

Iceberg type	Athena type	Notes
`struct<...>`	`struct`	
`fixed(L)`	-	The `fixed(L)` type is not currently supported in Athena.

For more information about data types in Athena, see Data types in Amazon Athena.

Additional resources

The following article is in the AWS Prescriptive Guidance documentation.

- Working with Apache Iceberg tables by using Amazon Athena SQL

For in-depth articles on using Athena with Apache Iceberg tables, see the following posts in the *AWS Big Data Blog*.

- Implement a serverless CDC process with Apache Iceberg using Amazon DynamoDB and Amazon Athena
- Accelerate data science feature engineering on transactional data lakes using Amazon Athena with Apache Iceberg
- Build an Apache Iceberg data lake using Amazon Athena, Amazon EMR, and AWS Glue
- Perform upserts in a data lake using Amazon Athena and Apache Iceberg
- Build a transactional data lake using Apache Iceberg, AWS Glue, and cross-account data shares using AWS Lake Formation and Amazon Athena
- Use Apache Iceberg in a data lake to support incremental data processing
- Build a real-time GDPR-aligned Apache Iceberg data lake
- Automate replication of relational sources into a transactional data lake with Apache Iceberg and AWS Glue
- Interact with Apache Iceberg tables using Amazon Athena and cross account fine-grained permissions using AWS Lake Formation
- Build a serverless transactional data lake with Apache Iceberg, Amazon EMR Serverless, and Amazon Athena

Amazon Athena security

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

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

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS compliance programs. To learn about the compliance programs that apply to Athena, 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 organization's requirements, and applicable laws and regulations.

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

Topics

Data protection in Athena

The AWS shared responsibility model applies to data protection in Amazon Athena. 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 Athena 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.

As an additional security step, you can use the aws:CalledVia global condition context key to limit requests to only those made from Athena. For more information, see Use CalledVia context keys for Athena.

Protect multiple types of data

Multiple types of data are involved when you use Athena to create databases and tables. These data types include source data stored in Amazon S3, metadata for databases and tables that you create when you run queries or the AWS Glue Crawler to discover data, query results data, and query history. This section discusses each type of data and provides guidance about protecting it.

- **Source data** – You store the data for databases and tables in Amazon S3, and Athena doesn't modify it. For more information, see Data protection in Amazon S3 in the *Amazon Simple Storage Service User Guide*. You control access to your source data and can encrypt it in Amazon S3. You can use Athena to create tables based on encrypted datasets in Amazon S3.
- **Database and table metadata (schema)** – Athena uses schema-on-read technology, which means that your table definitions are applied to your data in Amazon S3 when Athena runs queries. Any schemas you define are automatically saved unless you explicitly delete them. In Athena, you can modify the Data Catalog metadata using DDL statements. You can also delete table definitions and schema without impacting the underlying data stored in Amazon S3. The metadata for databases and tables you use in Athena is stored in the AWS Glue Data Catalog.

  You can define fine-grained access policies to databases and tables registered in the AWS Glue Data Catalog using AWS Identity and Access Management (IAM). You can also encrypt metadata in the AWS Glue Data Catalog. If you encrypt the metadata, use permissions to encrypted metadata for access.
- **Query results and query history, including saved queries** – Query results are stored in a location in Amazon S3 that you can choose to specify globally, or for each workgroup. If not specified, Athena uses the default location in each case. You control access to Amazon S3 buckets where you store query results and saved queries. Additionally, you can choose to encrypt query results that you store in Amazon S3. Your users must have the appropriate permissions to access the Amazon S3 locations and decrypt files. For more information, see Encrypt Athena query results stored in Amazon S3 in this document.

  Athena retains query history for 45 days. You can view query history using Athena APIs, in the console, and with AWS CLI. To store the queries for longer than 45 days, save them. To protect access to saved queries, use workgroups in Athena, restricting access to saved queries only to users who are authorized to view them.

Topics

- Encryption at rest

- Encryption in transit
- Key management
- Internetwork traffic privacy

Encryption at rest

You can run queries in Amazon Athena on encrypted data in Amazon S3 in the same Region and across a limited number of Regions. You can also encrypt the query results in Amazon S3 and the data in the AWS Glue Data Catalog.

You can encrypt the following assets in Athena:

- The results of all queries in Amazon S3, which Athena stores in a location known as the Amazon S3 results location. You can encrypt query results stored in Amazon S3 whether the underlying dataset is encrypted in Amazon S3 or not. For information, see Encrypt Athena query results stored in Amazon S3.
- The data in the AWS Glue Data Catalog. For information, see Permissions to encrypted metadata in the AWS Glue Data Catalog.

> **Note**
>
> When you use Athena to read an encrypted table, Athena uses the encryption options specified for the table data, not the encryption option for the query results. If separate encryption methods or keys are configured for query results and table data, Athena reads the table data without using the encryption option and key used to encrypt or decrypt the query results.
> However, if you use Athena to insert data into a table that has encrypted data, Athena uses the encryption configuration that was specified for the query results to encrypt the inserted data. For example, if you specify `CSE_KMS` encryption for query results, Athena uses the same AWS KMS key ID that you used for query results encryption to encrypt the inserted table data with `CSE_KMS`.

Topics

- Supported Amazon S3 encryption options
- Permissions to encrypted data in Amazon S3

- Permissions to encrypted metadata in the AWS Glue Data Catalog
- Migrate from CSE-KMS to SSE-KMS
- Encrypt Athena query results stored in Amazon S3
- Create tables based on encrypted datasets in Amazon S3

Supported Amazon S3 encryption options

Athena supports the following encryption options for datasets and query results in Amazon S3.

Encryption type	Description	Cross-Region support
SSE-S3	Server side encryption (SSE) with an Amazon S3-managed key.	Yes
SSE-KMS (Recommended)	Server-side encryption (SSE) with an AWS Key Management Service customer managed key.	Yes
CSE-KMS	Client-side encryption (CSE) with a AWS KMS customer managed key. In Athena, this option requires that you use a `CREATE TABLE` statement with a `TBLPROPERTIES` clause that specifies `'has_encrypted_data'='true'`. For more information, see Create tables based on encrypted datasets in Amazon S3.	No

For more information about AWS KMS encryption with Amazon S3, see What is AWS Key Management Service and How Amazon Simple Storage Service (Amazon S3) uses AWS KMS in the *AWS Key Management Service Developer Guide*. For more information about using SSE-KMS or CSE-KMS with Athena, see Launch: Amazon Athena adds support for querying encrypted data from the *AWS Big Data Blog*.

Encryption recommendations

When you encrypt and decrypt table data and query results with customer-managed KMS keys, we recommend that you use SSE-KMS encryption over SSE-S3 or CSE-KMS encryption methods.

SSE-KMS provides a balance of control, simplicity, and performance that makes it a recommended method when you use managed KMS keys for data encryption.

Benefits of SSE-KMS over SSE-S3

- SSE-KMS allows you to specify and manage your own keys, providing greater control. You can define key policies, oversee key life cycles, and monitor key usage.

Benefits of SSE-KMS over CSE-KMS

- SSE-KMS eliminates the need for additional infrastructure to encrypt and decrypt data, unlike CSE-KMS which requires ongoing maintenance of an S3 encryption client.
- CSE-KMS might face compatibility issues between newer and older S3 encryption clients due to evolving encryption algorithms, a problem SSE-KMS avoids.
- SSE-KMS makes fewer API calls to the KMS service for key retrieval during the encryption and decryption processes, resulting in better performance compared to CSE-KMS.

Unsupported options

The following encryption options aren't supported:

- SSE with customer-provided keys (SSE-C).
- Client-side encryption using a client-side managed key.
- Asymmetric keys.

To compare Amazon S3 encryption options, see Protecting data using encryption in the *Amazon Simple Storage Service User Guide*.

Tools for client-side encryption

For client-side encryption, note that two tools are available:

- Amazon S3 encryption client – This encrypts data for Amazon S3 only and is supported by Athena.
- AWS Encryption SDK – The SDK can be used to encrypt data anywhere across AWS but is not directly supported by Athena.

These tools aren't compatible, and data encrypted using one tool cannot be decrypted by the other. Athena only supports the Amazon S3 Encryption Client directly. If you use the SDK to encrypt your data, you can run queries from Athena, but the data is returned as encrypted text.

If you want to use Athena to query data that has been encrypted with the AWS Encryption SDK, you must download and decrypt your data, and then encrypt it again using the Amazon S3 Encryption Client.

Permissions to encrypted data in Amazon S3

Depending on the type of encryption you use in Amazon S3, you might need to add permissions, also known as "Allow" actions, to your policies used in Athena:

- **SSE-S3** – If you use SSE-S3 for encryption, Athena users require no additional permissions in their policies. It is sufficient to have the appropriate Amazon S3 permissions for the appropriate Amazon S3 location and for Athena actions. For more information about policies that allow appropriate Athena and Amazon S3 permissions, see AWS managed policies for Amazon Athena and Control access to Amazon S3 from Athena.
- **AWS KMS** – If you use AWS KMS for encryption, Athena users must be allowed to perform particular AWS KMS actions in addition to Athena and Amazon S3 permissions. You allow these actions by editing the key policy for the customer managed key that are used to encrypt data in Amazon S3. To add key users to the appropriate AWS KMS key policies, you can use the AWS KMS console at https://console.aws.amazon.com/kms. For information about how to add a user to a AWS KMS key policy, see Allows key users to use the customer managed key in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> Advanced key policy administrators can adjust key policies. `kms:Decrypt` is the minimum allowed action for an Athena user to work with an encrypted dataset. To work with encrypted query results, the minimum allowed actions are `kms:GenerateDataKey` and `kms:Decrypt`.

When using Athena to query datasets in Amazon S3 with a large number of objects that are encrypted with AWS KMS, AWS KMS might throttle query results. This is more likely when there are a large number of small objects. Athena backs off retry requests, but a throttling error might still occur. If you are working with a large number of encrypted objects and experience this issue, one option is to enable Amazon S3 bucket keys to reduce the number of calls to KMS. For more information, see Reducing the cost of SSE-KMS with Amazon S3 Bucket keys in the *Amazon Simple Storage Service User Guide*. Another option is to increase your service quotas for AWS KMS. For more information, see Quotas in the *AWS Key Management Service Developer Guide*.

For troubleshooting information about permissions when using Amazon S3 with Athena, see the Permissions section of the Troubleshoot issues in Athena topic.

Permissions to encrypted metadata in the AWS Glue Data Catalog

If you encrypt metadata in the AWS Glue Data Catalog, you must add `"kms:GenerateDataKey"`, `"kms:Decrypt"`, and `"kms:Encrypt"` actions to the policies you use for accessing Athena. For information, see Configure access from Athena to encrypted metadata in the AWS Glue Data Catalog.

Migrate from CSE-KMS to SSE-KMS

You can specify CSE-KMS encryption in two ways – during the workgroup query results encryption configuration and in the client-side settings. For more information, see Encrypt Athena query results stored in Amazon S3. During the migration process, it's important to audit your existing workflows that read and write CSE-KMS data, identify workgroups where CSE-KMS is configured, and locate instances where CSE-KMS is set through client-side parameters.

Update workgroup query results encryption settings

Console

To update encryption settings in the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena console navigation pane, choose **Workgroups**.
3. On the **Workgroups** page, select the button for the workgroup that you want to edit.
4. Choose **Actions**, **Edit**.
5. Open **Query result configuration** and choose **Encrypt query results**.
6. For **Encryption type** section, choose **SSE_KMS** encryption option.
7. Enter your KMS key under **Choose a different AWS KMS key (advanced)**.
8. Choose **Save changes**. The updated workgroup appears in the list on the **Workgroups** page.

CLI

Run the following command to update your query results encryption configuration to SSE-KMS in your workgroup.

```
aws athena update-work-group \
    --work-group "my-workgroup" \
    --configuration-updates '{
        "ResultConfigurationUpdates": {
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_KMS",
                "KmsKey": "<my-kms-key>"
            }
        }
    }'
```

Update client-side query results encryption settings

Console

To update your client-side settings for query results encryption from CSE-KMS to SSE-KMS, see [Encrypt Athena query results stored in Amazon S3](#).

CLI

You can only specify query results encryption configuration in client-side settings with the `start-query-execution` command. If you run this CLI command and override the query results encryption configuration that you specified in your workgroup with CSE-KMS, change the command to encrypt query results using `SSE_KMS` as following.

```
aws athena start-query-execution \
    --query-string "SELECT * FROM <my-table>;" \
    --query-execution-context "Database=<my-database>,Catalog=<my-catalog>" \
    --result-configuration '{
        "EncryptionConfiguration": {
            "EncryptionOption": "SSE_KMS",
            "KmsKey": "<my-kms-key>"
        }
    }' \
    --work-group "<my-workgroup>"
```

> **Note**
>
> - After you update the workgroup or client-side settings, any new data that you insert by write queries uses the SSE-KMS encryption instead of CSE-KMS. This is because query results encryption configurations also apply to newly inserted table data. Athena query result, metadata, and manifest files are also encrypted with SSE-KMS.
> - Athena can still read tables with the `has_encrypted_data` table property even when there are a mix of CSE-KMS encrypted and SSE-S3/SSE-KMS objects.

Convert CSE-KMS table data to SSE-KMS

If your workflows currently use CSE-KMS for table data encryption, transition to SSE-KMS with the following steps.

Prerequisite

If you still write data using a CSE-KMS workgroup or client-side settings, follow the steps in ??? to update it to SSE-KMS. This prevents new CSE-KMS encrypted data from being added during the migration process from any other workflows that might write to the tables.

Data migration

1. Check if the table has the `has_encrypted_data` property set to `true`. This property specifies that the table might contain CSE-KMS encrypted data. However, it's important to note that this property could be present even on tables without any actual CSE-KMS encrypted data.

   Console

   a. Open the Athena console at https://console.aws.amazon.com/athena/.

   b. Choose **Launch query editor**.

   c. On the left side of the editor, under **Database**, choose the database that you want to query.

   d. In the Query editor, run the following query to see the value set to the `has_encrypted_data table` property.

   ```
   SHOW TBLPROPERTIES <table_name>('has_encrypted_data');
   ```

CLI

Start Athena query that shows the value of the `has_encrypted_data` property on the table as shown in the following example.

```
aws athena start-query-execution \
    --query-string "SHOW TBLPROPERTIES <table-name>('has_encrypted_data');" \
    --work-group "<my-workgroup>"
```

Fetch query results to check the value of `has_encrypted_data` table property for the table as shown in the following example.

```
aws athena get-query-results --query-execution-id <query-execution-id-from-
previous-step>
```

2. For each CSE-KMS encrypted object in the table.

   a. Download the object from S3 using the S3 encryption client and decrypt it. Here is an example with AWS Java SDK V2.

   Imports

```
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.encryption.s3.S3EncryptionClient;
import software.amazon.encryption.s3.materials.Keyring;
import software.amazon.encryption.s3.materials.KmsDiscoveryKeyring;
```

   Code

```
final Keyring kmsDiscoveryKeyRing = KmsDiscoveryKeyring.builder()
        .enableLegacyWrappingAlgorithms(true)
        .build();
final S3EncryptionClient s3EncryptionClient = S3EncryptionClient.builder()
        .enableLegacyUnauthenticatedModes(true)
        .keyring(kmsDiscoveryKeyRing)
        .build();

GetObjectRequest getObjectRequest = GetObjectRequest.builder()
        .bucket("amzn-s3-demo-bucket")
```

```
        .key("<my-key>")
        .build();

ResponseInputStream<GetObjectResponse> s3Object =
 s3EncryptionClient.getObject(getObjectRequest);
```

b. Upload the object to S3 with the same name and SSE-KMS encryption. Here is an example with AWS Java SDK V2.

Imports

```
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.ServerSideEncryption;
```

Code

```
final S3Client s3Client = S3Client.builder()
        .build();

PutObjectRequest putObjectRequest = PutObjectRequest.builder()
        .bucket("amzn-s3-demo-bucket")
        .key("<my-key>")
        .serverSideEncryption(ServerSideEncryption.AWS_KMS)
        .ssekmsKeyId("<my-kms-key>")
        .build();

s3Client.putObject(putObjectRequest,
 RequestBody.fromBytes(s3Object.readAllBytes()));
```

Post migration

After successfully re-encrypting all CSE-KMS files in the table, perform the following steps.

1. Remove the `has_encrypted_data` property from the table.

   Console

   1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. Choose **Launch query editor**.
3. On the left side of the editor, under **Database**, choose the database that you want to query.
4. In the Query editor, run the following query for your table.

```
ALTER TABLE <database-name>.<table-name> UNSET TBLPROPERTIES
 ('has_encrypted_data')
```

CLI

Run the following command to remove the `has_encrypted_data` property from your table.

```
aws athena start-query-execution \
    --query-string "ALTER TABLE <database-name>.<table-name> UNSET TBLPROPERTIES
 ('has_encrypted_data');" \
    --work-group "<my-workgroup>"
```

2. Update your workflows to use a basic S3 client instead of a S3 encryption client and then specify SSE-KMS encryption for data writes.

Encrypt Athena query results stored in Amazon S3

You set up query result encryption using the Athena console or when using JDBC or ODBC. Workgroups allow you to enforce the encryption of query results.

> **Note**
>
> When you encrypt query results, Athena encrypts all objects written by the query. This includes the results of statements like `INSERT INTO`, `UPDATE`, and queries of data in Iceberg or other formats.

In the console, you can configure the setting for encryption of query results in two ways:

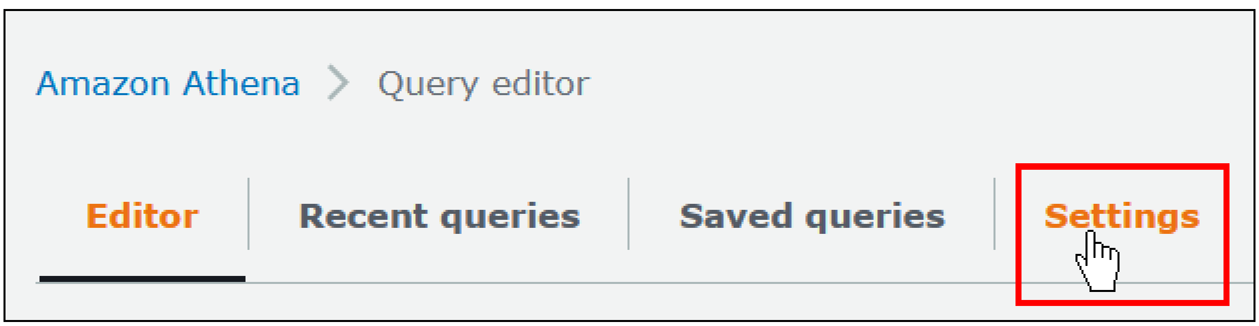
- **Client-side settings** – When you use **Settings** in the console or the API operations to indicate that you want to encrypt query results, this is known as using client-side settings. Client-side settings include query results location and encryption. If you specify them, they are used, unless they are overridden by the workgroup settings.

- **Workgroup settings** – When you create or edit a workgroup and select the **Override client-side settings** field, then all queries that run in this workgroup use the workgroup encryption and query results location settings. For more information, see Override client-side settings.

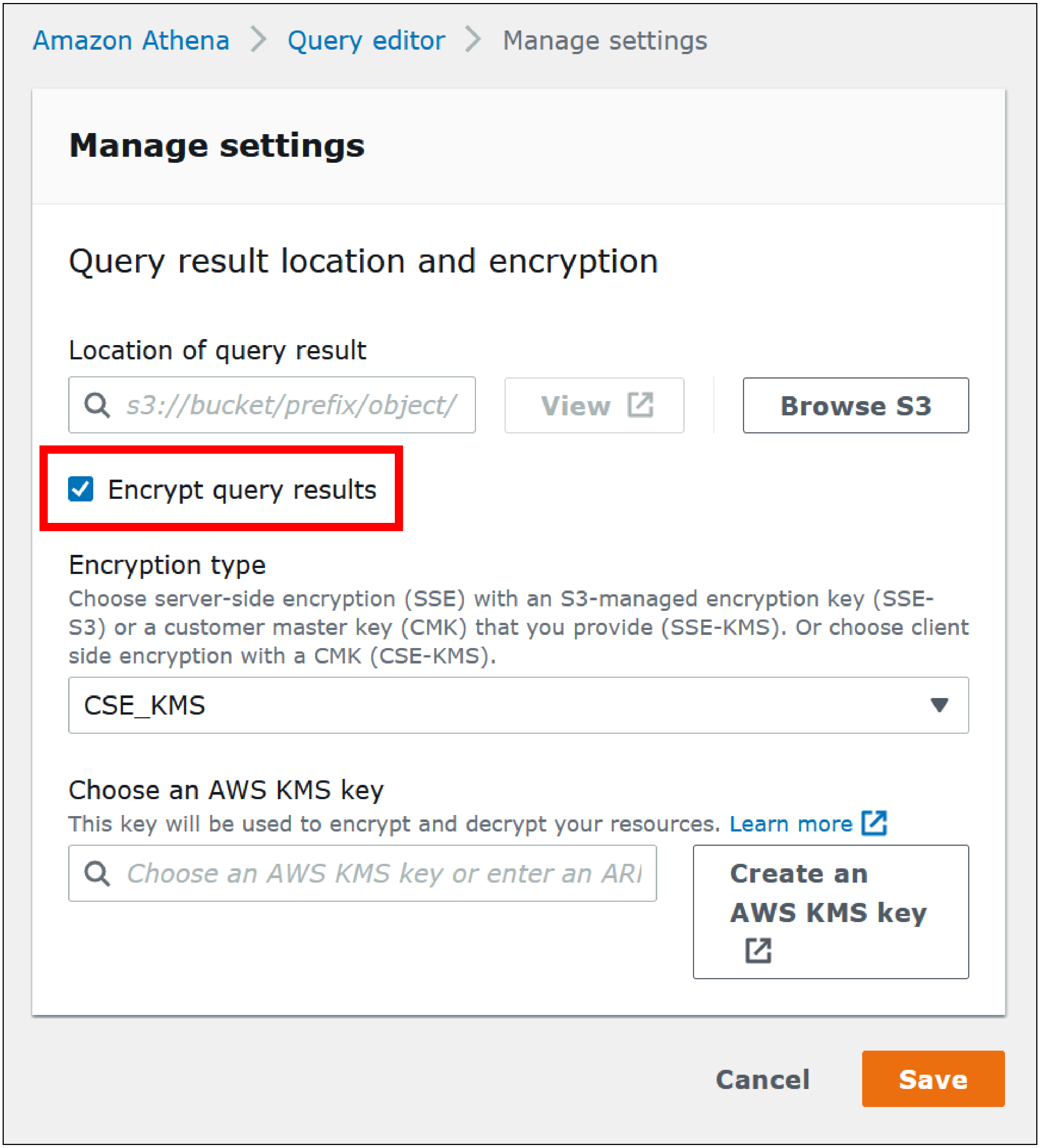
To encrypt query results stored in Amazon S3 using the console

> **Important**
>
> If your workgroup has the **Override client-side settings** field selected, then all queries in the workgroup use the workgroup settings. The encryption configuration and the query results location specified on the **Settings** tab in the Athena console, by API operations and by JDBC and ODBC drivers aren't used. For more information, see Override client-side settings.

1. In the Athena console, choose **Settings**.



2. Choose **Manage**.
3. For **Location of query result**, enter or choose an Amazon S3 path. This is the Amazon S3 location where query results are stored.
4. Choose **Encrypt query results**.



5. For **Encryption type**, choose **CSE-KMS**, **SSE-KMS**, or **SSE-S3**. Of these three, **CSE-KMS** offers the highest level of encryption and **SSE-S3** the lowest.
6. If you chose **SSE-KMS** or **CSE-KMS**, specify an AWS KMS key.

- For **Choose an AWS KMS key**, if your account has access to an existing AWS KMS customer managed key, choose its alias or enter an AWS KMS key ARN.
- If your account doesn't have access to an existing customer managed key, choose **Create an AWS KMS key**, and then open the AWS KMS console. For more information, see Creating keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> Athena supports only symmetric keys for reading and writing data.

7. Return to the Athena console and choose the key that you created by alias or ARN.
8. Choose **Save**.

Encrypt Athena query results when you use JDBC or ODBC

If you connect using a JDBC or ODBC driver, you configure driver options to specify the type of encryption to use and the Amazon S3 staging directory location. To configure a JDBC or ODBC driver to encrypt your query results using any of the encryption protocols that Athena supports, see Connect to Amazon Athena with ODBC and JDBC drivers.

Create tables based on encrypted datasets in Amazon S3

When you create a table, indicate to Athena that a dataset is encrypted in Amazon S3. This isn't required when using SSE-KMS. For both SSE-S3 and AWS KMS encryption, Athena determines how to decrypt the dataset and create the table, so mustn't provide key information.

Users that run queries, including the user who creates the table, must have the permissions described earlier in this topic.

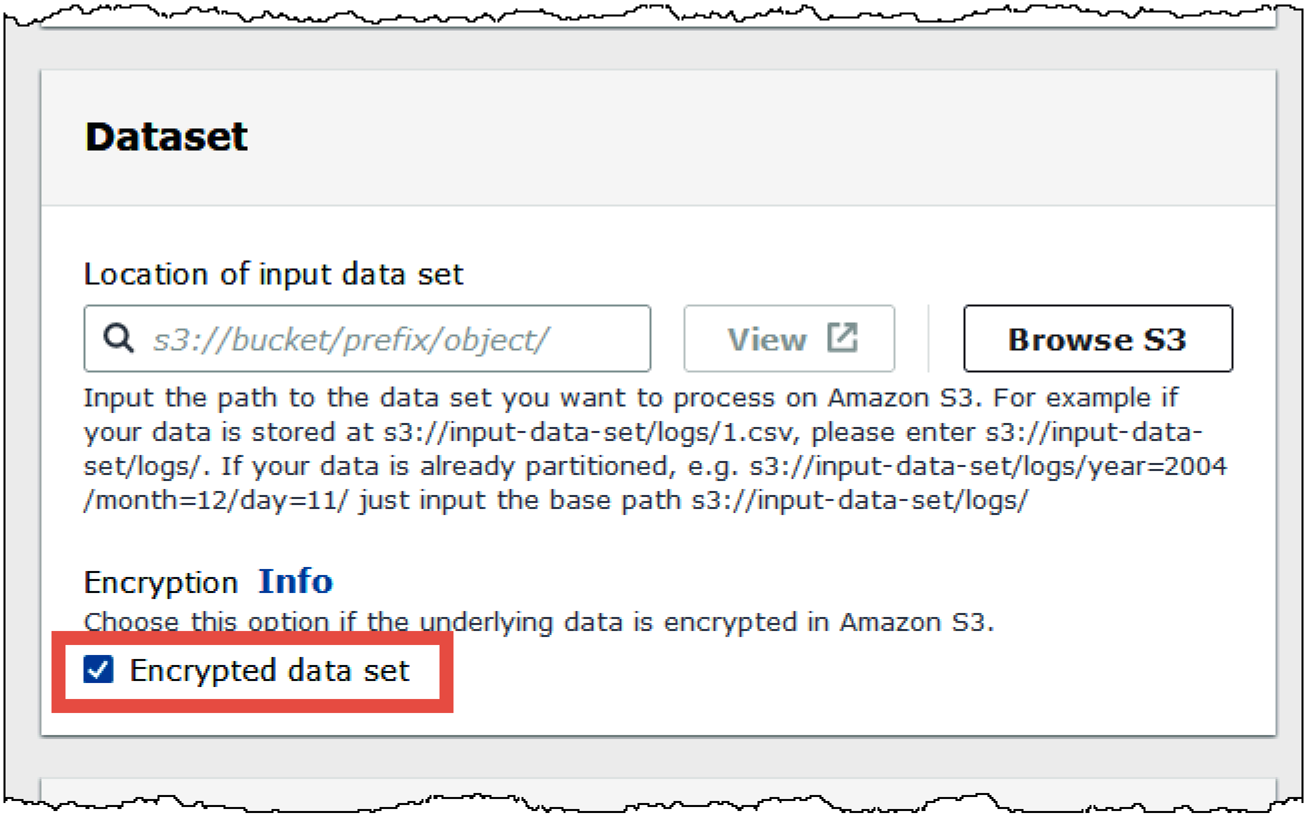
> **Important**
>
> If you use Amazon EMR along with EMRFS to upload encrypted Parquet files, you must disable multipart uploads by setting `fs.s3n.multipart.uploads.enabled` to `false`. If you don't do this, Athena is unable to determine the Parquet file length and a **HIVE_CANNOT_OPEN_SPLIT** error occurs. For more information, see Configure multipart upload for Amazon S3 in the *Amazon EMR Management Guide*.

To indicate that the dataset is encrypted in Amazon S3, perform one of the following steps. This step isn't required if SSE-KMS is used.

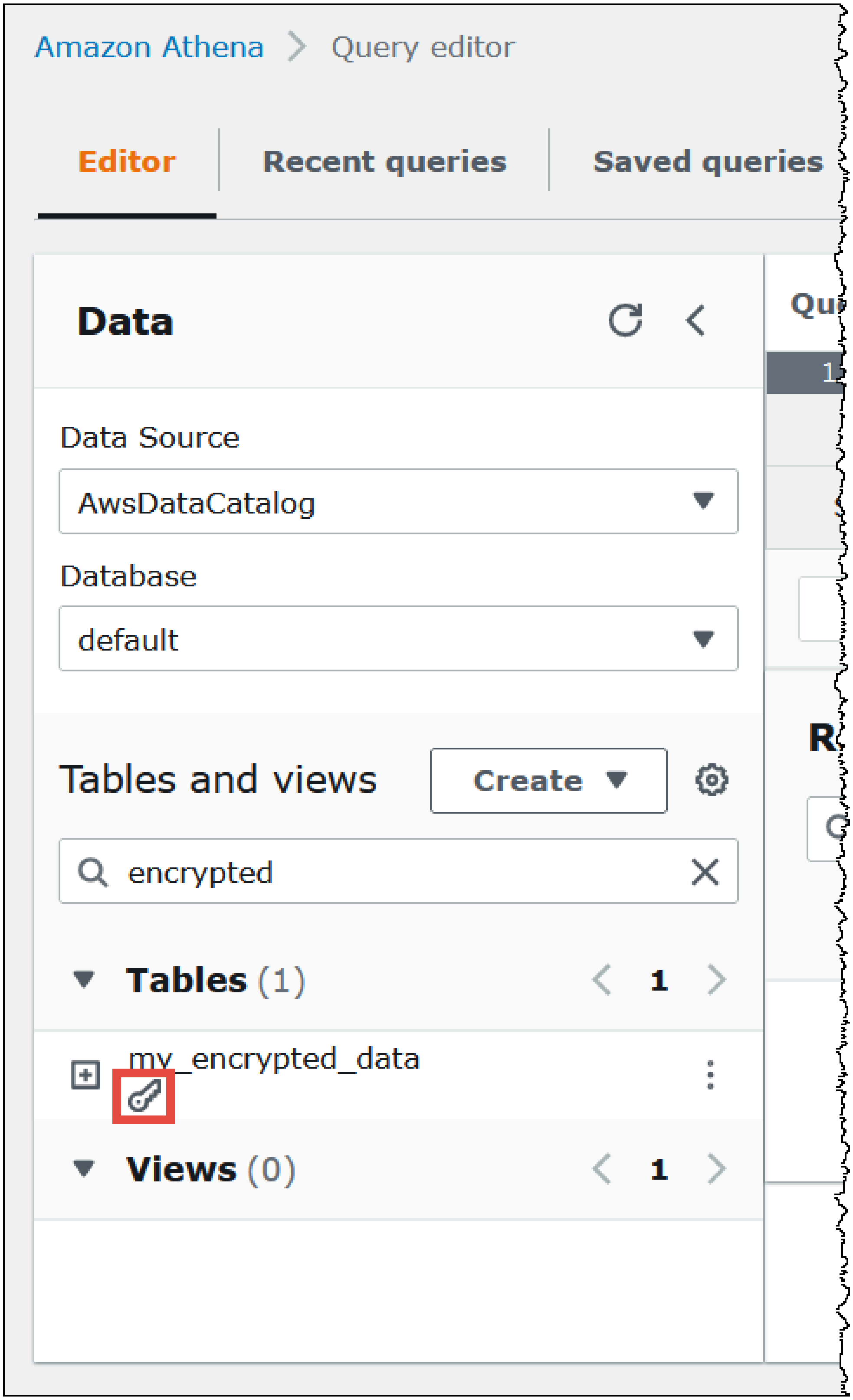
- In a CREATE TABLE statement, use a TBLPROPERTIES clause that specifies 'has_encrypted_data'='true', as in the following example.

```
CREATE EXTERNAL TABLE 'my_encrypted_data' (
   `n_nationkey` int,
   `n_name` string,
   `n_regionkey` int,
   `n_comment` string)
ROW FORMAT SERDE
   'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
   'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
LOCATION
   's3://amzn-s3-demo-bucket/folder_with_my_encrypted_data/'
TBLPROPERTIES (
   'has_encrypted_data'='true')
```

- Use the JDBC driver and set the TBLPROPERTIES value as shown in the previous example when you use statement.executeQuery() to run the CREATE TABLE statement.
- When you use the Athena console to create a table using a form and specify the table location, select the **Encrypted data set** option.



In the Athena console list of tables, encrypted tables display a key-shaped icon.


Amazon Athena > Query editor
Editor
Recent queries
Saved queries
Data
Data Source
AwsDataCatalog
Database
default
Tables and views
Create
encrypted
Tables (1)
1
my_encrypted_data
Views (0)
1

Encryption in transit

In addition to encrypting data at rest in Amazon S3, Amazon Athena uses Transport Layer Security (TLS) encryption for data in-transit between Athena and Amazon S3, and between Athena and customer applications accessing it.

You should allow only encrypted connections over HTTPS (TLS) using the `aws:SecureTransport condition` on Amazon S3 bucket IAM policies.

Query results that stream to JDBC or ODBC clients are encrypted using TLS. For information about the latest versions of the JDBC and ODBC drivers and their documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

For Athena federated data source connectors, support for encryption in transit using TLS depends on the individual connector. For information, see the documentation for the individual data source connectors.

Key management

Amazon Athena supports AWS Key Management Service (AWS KMS) to encrypt datasets in Amazon S3 and Athena query results. AWS KMS uses customer managed keys to encrypt your Amazon S3 objects and relies on envelope encryption.

In AWS KMS, you can perform the following actions:

- Create keys
- Import your own key material for new customer managed keys

> **Note**
>
> Athena supports only symmetric keys for reading and writing data.

For more information, see What is AWS Key Management Service in the *AWS Key Management Service Developer Guide*, and How Amazon Simple Storage Service uses AWS KMS. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.

If you are uploading or accessing objects encrypted by SSE-KMS, use AWS Signature Version 4 for added security. For more information, see Specifying the signature version in request authentication in the *Amazon Simple Storage Service User Guide*.

If your Athena workloads encrypt a large amount of data, you can use Amazon S3 Bucket Keys to reduce costs. For more information, see Reducing the cost of SSE-KMS with Amazon S3 Bucket keys in the *Amazon Simple Storage Service User Guide*.

Internetwork traffic privacy

Traffic is protected both between Athena and on-premises applications and between Athena and Amazon S3. Traffic between Athena and other services, such as AWS Glue and AWS Key Management Service, uses HTTPS by default.

- **For traffic between Athena and on-premises clients and applications**, query results that stream to JDBC or ODBC clients are encrypted using Transport Layer Security (TLS).

  You can use one of the connectivity options between your private network and AWS:

  - A Site-to-Site VPN AWS VPN connection. For more information, see What is Site-to-Site VPN AWS VPN in the *AWS Site-to-Site VPN User Guide*.
  - An AWS Direct Connect connection. For more information, see What is AWS Direct Connect in the *AWS Direct Connect User Guide*.
- **For traffic between Athena and Amazon S3 buckets**, Transport Layer Security (TLS) encrypts objects in-transit between Athena and Amazon S3, and between Athena and customer applications accessing it, you should allow only encrypted connections over HTTPS (TLS) using the `aws:SecureTransport condition` on Amazon S3 bucket IAM policies. Although Athena currently uses the public endpoint to access data in Amazon S3 buckets, this doesn't mean that the data traverses the public internet. All traffic between Athena and Amazon S3 is routed over the AWS network and is encrypted using TLS.
- **Compliance programs** – Amazon Athena complies with multiple AWS compliance programs, including SOC, PCI, FedRAMP, and others. For more information, see AWS services in scope by compliance program.

Identity and access management in Athena

Amazon Athena uses AWS Identity and Access Management (IAM) policies to restrict access to Athena operations. For a full list of permissions for Athena, see Actions, resources, and condition keys for Amazon Athena in the *Service Authorization Reference*.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

The permissions required to run Athena queries include the following:

- Amazon S3 locations where the underlying data to query is stored. For more information, see Identity and access management in Amazon S3 in the *Amazon Simple Storage Service User Guide*.
- Metadata and resources that you store in the AWS Glue Data Catalog, such as databases and tables, including additional actions for encrypted metadata. For more information, see Setting up IAM permissions for AWS Glue and Setting up encryption in AWS Glue in the *AWS Glue Developer Guide*.
- Athena API actions. For a list of API actions in Athena, see Actions in the *Amazon Athena API Reference*.

The following topics provide more information about permissions for specific areas of Athena.

Topics

- Configure access to prepared statements
- Use CalledVia context keys for Athena
- Allow access to the Athena Data Connector for External Hive Metastore
- Allow Lambda function access to external Hive metastores
- Permissions required to create connector and Athena catalog
- Allow access to Athena Federated Query: Example policies
- Allow access to Athena UDFs: Example policies
- Allow access for ML with Athena
- Enable federated access to the Athena API

AWS managed policies for Amazon Athena

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*.

Considerations when using managed policies with Athena

Managed policies are easy to use and are updated automatically with the required actions as the service evolves. When using managed policies with Athena, keep the following points in mind:

- To allow or deny Amazon Athena service actions for yourself or other users using AWS Identity and Access Management (IAM), you attach identity-based policies to principals, such as users or groups.

- Each identity-based policy consists of statements that define the actions that are allowed or denied. For more information and step-by-step instructions for attaching a policy to a user, see Attaching managed policies in the *IAM User Guide*. For a list of actions, see the Amazon Athena API Reference.
- *Customer-managed* and *inline* identity-based policies allow you to specify more detailed Athena actions within a policy to fine-tune access. We recommend that you use the `AmazonAthenaFullAccess` policy as a starting point and then allow or deny specific actions listed in the Amazon Athena API Reference. For more information about inline policies, see Managed policies and inline policies in the *IAM User Guide*.
- If you also have principals that connect using JDBC, you must provide the JDBC driver credentials to your application. For more information, see Control access through JDBC and ODBC connections.
- If you have encrypted the AWS Glue Data Catalog, you must specify additional actions in the identity-based IAM policies for Athena. For more information, see Configure access from Athena to encrypted metadata in the AWS Glue Data Catalog.
- If you create and use workgroups, make sure your policies include relevant access to workgroup actions. For detailed information, see the section called "Use IAM policies to control workgroup access" and the section called "Example workgroup policies".

AWS managed policy: AmazonAthenaFullAccess

The `AmazonAthenaFullAccess` managed policy grants full access to Athena.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.

- (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user (console)]() in the *IAM User Guide*.

Permissions groupings

The `AmazonAthenaFullAccess` policy is grouped into the following sets of permissions.

- **`athena`** – Allows principals access to Athena resources.
- **`glue`** – Allows principals access to AWS Glue Catalogs, databases, tables, and partitions. This is required so that the principal can use the AWS Glue Data Catalogs with Athena.
- **`s3`** – Allows the principal to write and read query results from Amazon S3, to read publically available Athena data examples that reside in Amazon S3, and to list buckets. This is required so that the principal can use Athena to work with Amazon S3.
- **`sns`** – Allows principals to list Amazon SNS topics and get topic attributes. This enables principals to use Amazon SNS topics with Athena for monitoring and alert purposes.
- **`cloudwatch`** – Allows principals to create, read, and delete CloudWatch alarms. For more information, see Use CloudWatch and EventBridge to monitor queries and control costs.
- **`lakeformation`** – Allows principals to request temporary credentials to access data in a data lake location that is registered with Lake Formation. For more information, see Underlying data access control in the *AWS Lake Formation Developer Guide*.
- **`datazone`** – Allows principals to list Amazon DataZone projects, domains, and environments. For information about using DataZone in Athena, see Use Amazon DataZone in Athena.
- **`pricing`** – Provides access to AWS Billing and Cost Management. For more information, see GetProducts in the *AWS Billing and Cost Management API Reference*.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Sid": "BaseAthenaPermissions",
            "Effect": "Allow",
            "Action": [
                "athena:*"
            ],
            "Resource": [
                "*"
            ]
```

```
        },
        {
            "Sid": "BaseGluePermissions",
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetCatalog",
                "glue:GetCatalogs",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:StartColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRuns",
                "glue:GetCatalogImportStatus"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseQueryResultsPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
```

```
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Sid": "BaseAthenaExamplesPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::athena-examples*"
            ]
        },
        {
            "Sid": "BaseS3BucketPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseSNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics",
                "sns:GetTopicAttributes"
            ],
            "Resource": [
                "*"
            ]
        },
        {
```

```
            "Sid": "BaseCloudWatchPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseLakeFormationPermissions",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseDataZonePermissions",
            "Effect": "Allow",
            "Action": [
                "datazone:ListDomains",
                "datazone:ListProjects",
                "datazone:ListAccountEnvironments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BasePricingPermissions",
            "Effect": "Allow",
            "Action": [
                "pricing:GetProducts"
            ],
            "Resource": [
                "*"
            ]
        }
```

```
    ]
}
```

Note

You must explicitly allow access to service-owned Amazon S3 buckets to store example queries and sample dataset. For more information, see ???.

AWS managed policy: AWSQuicksightAthenaAccess

`AWSQuicksightAthenaAccess` grants access to actions that QuickSight requires for integration with Athena. You can attach the `AWSQuicksightAthenaAccess` policy to your IAM identities. Attach this policy only to principals who use QuickSight with Athena. This policy includes some actions for Athena that are either deprecated and not included in the current public API, or that are used only with the JDBC and ODBC drivers.

Permissions groupings

The `AWSQuicksightAthenaAccess` policy is grouped into the following sets of permissions.

- **`athena`** – Allows the principal to run queries on Athena resources.
- **`glue`** – Allows principals access to AWS Glue Catalogs, databases, tables, and partitions. This is required so that the principal can use the AWS Glue Data Catalogs with Athena.
- **`s3`** – Allows the principal to write and read query results from Amazon S3.
- **`lakeformation`** – Allows principals to request temporary credentials to access data in a data lake location that is registered with Lake Formation. For more information, see Underlying data access control in the *AWS Lake Formation Developer Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:CancelQueryExecution",
                "athena:GetCatalogs",
                "athena:GetExecutionEngine",
                "athena:GetExecutionEngines",
```

```
                "athena:GetNamespace",
                "athena:GetNamespaces",
                "athena:GetQueryExecution",
                "athena:GetQueryExecutions",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:GetTable",
                "athena:GetTables",
                "athena:ListQueryExecutions",
                "athena:RunQuery",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:GetWorkGroup",
                "athena:GetDataCatalog",
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetCatalog",
                "glue:GetCatalogs",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
```

```
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Athena updates to AWS managed policies

View details about updates to AWS managed policies for Athena since this service began tracking these changes.

Change	Description	Date
AWSQuicksightAthenaAccess – Updates to existing policies	The `glue:GetCatalog` and `glue:GetCatalogs` permissions were added to enable Athena users to access to SageMaker AI Lakehouse catalogs.	January 02, 2025
AmazonAthenaFullAccess – Update to existing policy	The `glue:GetCatalog` and `glue:GetCatalogs` permissions were added to enable Athena users to access to SageMaker AI Lakehouse catalogs.	January 02, 2025
AmazonAthenaFullAccess – Update to existing policy	Enables Athena to use the publicly documented AWS Glue `GetCatalogImportStatus` API to retrieve catalog import status.	June 18, 2024
AmazonAthenaFullAccess – Update to existing policy	The `datazone:ListDomains`, `datazone:ListProjects`, and `datazone:ListAccountEnvironments` permissions were added to enable Athena users to work with Amazon DataZone domains, projects, and environments. For more information, see Use Amazon DataZone in Athena.	January 3, 2024
AmazonAthenaFullAccess – Update to existing policy	The `glue:StartColumnStatisticsTaskRun`,	January 3, 2024

Change	Description	Date
	`glue:GetColumnStatisticsTaskRun` , and `glue:GetColumnStatisticsTaskRuns` permissions were added to give Athena the right to call AWS Glue to retrieve statistics for the cost-based optimizer feature. For more information, see Use the cost-based optimizer.	
AmazonAthenaFullAccess – Update to existing policy	Athena added `pricing:GetProducts` to provide access to AWS Billing and Cost Management. For more information, see GetProducts in the *AWS Billing and Cost Management API Reference*.	January 25, 2023
AmazonAthenaFullAccess – Update to existing policy	Athena added `cloudwatch:GetMetricData` to retrieve CloudWatch metric values. For more information, see GetMetricData in the *Amazon CloudWatch API Reference*.	November 14, 2022
AmazonAthenaFullAccess and AWSQuicksightAthenaAccess – Updates to existing policies	Athena added `s3:PutBucketPublicAccessBlock` to enable the blocking of public access on the buckets created by Athena.	July 7, 2021
Athena started tracking changes	Athena started tracking changes for its AWS managed policies.	July 7, 2021

Data perimeters

A data perimeter is a set of permissions guardrails in your AWS environment you use to help ensure that only your trusted identities are accessing trusted resources from expected networks.

Amazon Athena uses service-owned Amazon S3 buckets to store example queries and sample datasets. If you are using data perimeters to control access in your environment, you must explicitly allow access to these service-owned resources to use the corresponding Athena features.

The following table lists the ARN of the Amazon S3 bucket that Athena needs to access, required permissions, identity used by Athena, and the features that rely on the S3 bucket. To allow access, replace `<region>` in the bucket ARN with your actual AWS Region and allowlist this bucket based on your Amazon S3 access controls.

Data perimeters that Athena uses

Resource ARN	Required permissions	Identity used for access	Access scenarios
`arn:aws:s3:::athena-examples-<region>`	s3:GetObject s3:ListBucket	The IAM principal accessing Athena.	• Running example queries in the Athena console • Exploring sample datasets that Athena provides

Control access through JDBC and ODBC connections

To gain access to AWS services and resources, such as Athena and the Amazon S3 buckets, provide the JDBC or ODBC driver credentials to your application. If you are using the JDBC or ODBC driver, ensure that the IAM permissions policy includes all of the actions listed in AWS managed policy: AWSQuicksightAthenaAccess.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Authentication methods

The Athena JDBC and ODBC drivers support SAML 2.0-based authentication, including the following identity providers:

- Active Directory Federation Services (AD FS)
- Azure Active Directory (AD)
- Okta
- PingFederate

For more information, see the installation and configuration guides for the respective drivers, downloadable in PDF format from the JDBC and ODBC driver pages. For additional related information, see the following:

- Enable federated access to the Athena API
- Use Lake Formation and JDBC or ODBC drivers for federated access to Athena
- Configure single sign-on using ODBC, SAML 2.0, and the Okta Identity Provider

For information about the latest versions of the JDBC and ODBC drivers and their documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

Control access to Amazon S3 from Athena

You can grant access to Amazon S3 locations using identity-based policies, bucket resource policies, access point policies, or any combination of the above. When actors interact with Athena, their permissions pass through Athena to determine what Athena can access. This means that users must have permission to access Amazon S3 buckets in order to query them with Athena.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Note that requests to Amazon S3 come from a private IPv4 address for Athena, not the source IP specified in `aws:SourceIp`. For this reason, you cannot use the `aws:SourceIp` condition to deny access to Amazon S3 actions in a given IAM policy. You also cannot restrict or allow access to Amazon S3 resources based on the `aws:SourceVpc` or `aws:SourceVpce` condition keys.

> **Note**
>
> Athena workgroups that use IAM Identity Center authentication require that S3 Access Grants be configured to use trusted identity propagation identities. For more information, see S3 Access Grants and directory identities in the *Amazon Simple Storage Service User Guide*.

Topics

- Use identity-based policies to control access to Amazon S3 buckets
- Use bucket resource policies to control access to Amazon S3 buckets
- Use Amazon S3 access points for more precise control over bucket access
- Use CalledVia context keys to allow only calls from Athena to another service
- Additional resources

Use identity-based policies to control access to Amazon S3 buckets

Identity-based policies are attached to an IAM user, group, or role. These policies let you specify what that identity can do (its permissions). You can use identity-based policies to control access to your Amazon S3 buckets.

The following identity-based policy allows `Read` and `Write` access to objects in a specific Amazon S3 bucket. To use this policy, replace the *`italicized placeholder text`* with your own values.

```
{
"Version": "2012-10-17",
"Statement":
    [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource":
                ["arn:aws:s3:::amzn-s3-demo-bucket"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource":
                ["arn:aws:s3:::amzn-s3-demo-bucket/*"]
        }
    ]
}
```

Use bucket resource policies to control access to Amazon S3 buckets

You can use Amazon S3 bucket policies to secure access to objects in your buckets so that only users with the appropriate permissions can access them. For guidance on creating your Amazon S3 policy, see [Adding a bucket policy by using the Amazon S3 console]() in the *Amazon S3 User Guide*.

The following example permissions policy limits a user to reading objects that have the `environment: production` tag key and value. The example policy uses the `s3:ExistingObjectTag` condition key to specify the tag key and value.

```
{
    "Version":"2012-10-17",
    "Statement":
    [
        {
            "Principal":{"AWS":"arn:aws:iam::111122223333:role/JohnDoe"
        },
            "Effect":"Allow",
            "Action": [ "s3:GetObject", "s3:GetObjectVersion" ],
            "Resource":"arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition":
                {
                    "StringEquals":{"s3:ExistingObjectTag/environment":"production"
                }
    ]
}
```

For more bucket policy examples, see [Examples of Amazon S3 bucket policies]() in the *Amazon S3 User Guide*.

Use Amazon S3 access points for more precise control over bucket access

If you have a shared dataset in an Amazon S3 bucket, maintaining a single bucket policy that manages access for hundreds of use cases can be challenging.

Amazon S3 bucket access points, policies, and aliases can help solve this issue. A bucket can have multiple access points, each with a policy that controls access to the bucket in a different way.

For each access point that you create, Amazon S3 generates an alias that represents the access point. Because the alias is in Amazon S3 bucket name format, you can use the alias in the `LOCATION` clause of your `CREATE TABLE` statements in Athena. Athena's access to the bucket is then controlled by the policy for the access point that the alias represents.

For more information, see Specify a table location in Amazon S3 and Using access points in the *Amazon S3 User Guide*.

Use CalledVia context keys to allow only calls from Athena to another service

For added security, you can use the aws:CalledVia global condition context key. The `aws:CalledVia` condition key contains a list of services that you allow to call another service. For example, you can allow `InvokeFunction` calls to AWS Lambda only if the calls come from Athena by specifying the Athena service principal name `athena.amazonaws.com` for the `aws:CalledVia` context key. For more information, see Use CalledVia context keys for Athena.

Additional resources

For detailed information and examples about how to grant Amazon S3 access, see the following resources:

- Example walkthroughs: Managing access in the *Amazon S3 User Guide*.
- How can I provide cross-account access to objects that are in Amazon S3 buckets? in the AWS Knowledge Center.
- Configure cross-account access in Athena to Amazon S3 buckets.

Configure cross-account access in Athena to Amazon S3 buckets

A common Amazon Athena scenario is granting access to users in an account different from the bucket owner so that they can perform queries. In this case, use a bucket policy to grant access.

> **Note**
>
> For information about cross-account access to AWS Glue data catalogs from Athena, see Configure cross-account access to AWS Glue data catalogs.

The following example bucket policy, created and applied to bucket `s3://amzn-s3-demo-bucket` by the bucket owner, grants access to all users in account `123456789123`, which is a different account.

```
{
   "Version": "2012-10-17",
   "Id": "MyPolicyID",
```

```
    "Statement": [
       {
           "Sid": "MyStatementSid",
           "Effect": "Allow",
           "Principal": {
              "AWS": "arn:aws:iam::123456789123:root"
           },
           "Action": [
              "s3:GetBucketLocation",
              "s3:GetObject",
              "s3:ListBucket",
              "s3:ListBucketMultipartUploads",
              "s3:ListMultipartUploadParts",
              "s3:AbortMultipartUpload"
           ],
           "Resource": [
              "arn:aws:s3:::amzn-s3-demo-bucket",
              "arn:aws:s3:::amzn-s3-demo-bucket/*"
           ]
        }
     ]
  }
```

To grant access to a particular user in an account, replace the `Principal` key with a key that specifies the user instead of `root`. For example, for user profile `Dave`, use `arn:aws:iam::123456789123:user/Dave`.

Configure cross-account access to a bucket encrypted with a custom AWS KMS key

If you have an Amazon S3 bucket that is encrypted with a custom AWS Key Management Service (AWS KMS) key, you might need to grant access to it to users from another Amazon Web Services account.

Granting access to an AWS KMS-encrypted bucket in Account A to a user in Account B requires the following permissions:

- The bucket policy in Account A must grant access to the role assumed by Account B.
- The AWS KMS key policy in Account A must grant access to the role assumed by the user in Account B.
- The AWS Identity and Access Management (IAM) role assumed by Account B must grant access to both the bucket and the key in Account A.

The following procedures describe how to grant each of these permissions.

To grant access to the bucket in account a to the user in account b

- From Account A, review the S3 bucket policy and confirm that there is a statement that allows access from the account ID of Account B.

  For example, the following bucket policy allows `s3:GetObject` access to the account ID `111122223333`:

```
{
  "Id": "ExamplePolicy1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt1",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
      "Principal": {
        "AWS": [
          "111122223333"
        ]
      }
    }
  ]
}
```

To grant access to the user in account b from the AWS KMS key policy in account a

1. In the AWS KMS key policy for Account A, grant the role assumed by Account B permissions to the following actions:

   - `kms:Encrypt`
   - `kms:Decrypt`
   - `kms:ReEncrypt*`
   - `kms:GenerateDataKey*`
   - `kms:DescribeKey`

The following example grants key access to only one IAM role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role_name"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

2. From Account A, review the key policy [using the AWS Management Console policy view](#).
3. In the key policy, verify that the following statement lists Account B as a principal.

```
"Sid": "Allow use of the key"
```

4. If the `"Sid": "Allow use of the key"` statement is not present, perform the following steps:

   a. Switch to view the key policy [using the console default view](#).

   b. Add Account B's account ID as an external account with access to the key.

To grant access to the bucket and the key in account a from the IAM role assumed by account b

1. From Account B, open the IAM console at https://console.aws.amazon.com/iam/.
2. Open the IAM role associated with the user in Account B.
3. Review the list of permissions policies applied to IAM role.

4. Ensure that a policy is applied that grants access to the bucket.

   The following example statement grants the IAM role access to the `s3:GetObject` and `s3:PutObject` operations on the bucket `amzn-s3-demo-bucket`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt2",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

5. Ensure that a policy is applied that grants access to the key.

   Note

   If the IAM role assumed by Account B already has administrator access, then you don't need to grant access to the key from the user's IAM policies.

   The following example statement grants the IAM role access to use the key `arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt3",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt",
```

```
        "kms:GenerateDataKey",
        "kms:ReEncrypt*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd"
    }
  ]
}
```

Configure cross-account access to bucket objects

Objects that are uploaded by an account (Account C) other than the bucket's owning account (Account A) might require explicit object-level ACLs that grant read access to the querying account (Account B). To avoid this requirement, Account C should assume a role in Account A before it places objects in Account A's bucket. For more information, see How can I provide cross-account access to objects that are in Amazon S3 buckets?.

Configure access to databases and tables in the AWS Glue Data Catalog

If you use the AWS Glue Data Catalog with Amazon Athena, you can define resource-level policies for the database and table Data Catalog objects that are used in Athena.

> **Note**
>
> This topic discusses database- and table-level security. For information about configuring column-, row-, and cell-level security, see Data filtering and cell-level security in Lake Formation.

You define resource-level permissions in IAM identity-based policies.

> **Important**
>
> This section discusses resource-level permissions in IAM identity-based policies. These are different from resource-based policies. For more information about the differences, see Identity-based policies and resource-based policies in the *IAM User Guide*.

See the following topics for these tasks:

To perform this task	See the following topic
Create an IAM policy that defines access to resources	Creating IAM policies in the *IAM User Guide*.
Learn about IAM identity-based policies used in AWS Glue	Identity-based policies (IAM policies) in the *AWS Glue Developer Guide*.

In this section

- Limitations
- Configure AWS Glue access to your catalog and database per AWS Region
- About access control for table partitions and versions in AWS Glue
- Examples of database and table-level permissions

Limitations

Consider the following limitations when you use database and table-level access control for the AWS Glue Data Catalog and Athena:

- IAM Identity Center enabled Athena workgroups require Lake Formation be configured to use IAM Identity Center identities. For more information, see Integrating IAM Identity Center in the *AWS Lake Formation Developer Guide*.
- You can limit access only to databases and tables. These controls apply at the table level. You cannot limit access to individual partitions within a table. For more information, see About access control for table partitions and versions in AWS Glue.
- The AWS Glue Data Catalog contains the following resources: `CATALOG`, `DATABASE`, `TABLE`, and `FUNCTION`.

> **Note**
>
> From this list, resources that are common between Athena and the AWS Glue Data Catalog are `TABLE`, `DATABASE`, and `CATALOG` for each account. `Function` is specific

to AWS Glue. For delete actions in Athena, you must include permissions to AWS Glue actions. See Examples of database and table-level permissions.

The hierarchy is as follows: `CATALOG` is an ancestor of all `DATABASES` in each account, and each `DATABASE` is an ancestor for all of its `TABLES` and `FUNCTIONS`. For example, for a table named `table_test` that belongs to a database db in the catalog in your account, its ancestors are db and the catalog in your account. For the db database, its ancestor is the catalog in your account, and its descendants are tables and functions. For more information about the hierarchical structure of resources, see List of ARNs in Data Catalog in the *AWS Glue Developer Guide*.

- For any non-delete Athena action on a resource, such as `CREATE DATABASE`, `CREATE TABLE`, `SHOW DATABASE`, `SHOW TABLE`, or `ALTER TABLE`, you need permissions to call this action on the resource (table or database) and all ancestors of the resource in the Data Catalog. For example, for a table, its ancestors are the database to which it belongs, and the catalog for the account. For a database, its ancestor is the catalog for the account. See Examples of database and table-level permissions.
- For a delete action in Athena, such as `DROP DATABASE` or `DROP TABLE`, you also need permissions to call the delete action on all ancestors and descendants of the resource in the Data Catalog. For example, to delete a database you need permissions on the database, the catalog, which is its ancestor, and all the tables and user defined functions, which are its descendents. A table does not have descendants. To run `DROP TABLE`, you need permissions to this action on the table, the database to which it belongs, and the catalog. See Examples of database and table-level permissions.

Configure AWS Glue access to your catalog and database per AWS Region

For Athena to work with the AWS Glue, a policy that grants access to your database and to the AWS Glue Data Catalog in your account per AWS Region is required. To create databases, the `CreateDatabase` permission is also required. In the following example policy, replace the AWS Region, AWS account ID, and database name with those of your own.

```
{
   "Sid": "DatabasePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetDatabases",
```

```
      "glue:CreateDatabase"
   ],
   "Resource": [
     "arn:aws:glue:us-east-1:123456789012:catalog",
     "arn:aws:glue:us-east-1:123456789012:database/default"
   ]
}
```

About access control for table partitions and versions in AWS Glue

In AWS Glue, tables can have partitions and versions. Table versions and partitions are not considered to be independent resources in AWS Glue. Access to table versions and partitions is given by granting access on the table and ancestor resources for the table.

For the purposes of access control, the following access permissions apply:

- Controls apply at the table level. You can limit access only to databases and tables. For example, if you allow access to a partitioned table, this access applies to all partitions in the table. You cannot limit access to individual partitions within a table.

> **Important**
>
> To run actions in AWS Glue on partitions, permissions for partition actions are required at the catalog, database, and table levels. Having access to partitions within a table is not sufficient. For example, to run `GetPartitions` on table `myTable` in the database `myDB`, you must grant `glue:GetPartitions` permissions on the catalog, `myDB` database, and `myTable` resources.

- Access controls do not apply to table versions. As with partitions, access to previous versions of a table is granted through access to the table version APIs in AWS Glue on the table, and to the table ancestors.

For information about permissions on AWS Glue actions, see [AWS Glue API permissions: Actions and resources reference]() in the *AWS Glue Developer Guide*.

Examples of database and table-level permissions

The following table lists examples of IAM identity-based policies that allow access to databases and tables in Athena. We recommend that you start with these examples and, depending on your needs, adjust them to allow or deny specific actions to particular databases and tables.

These examples include access to databases and catalogs so that Athena and AWS Glue can work together. For multiple AWS Regions, include similar policies for each of your databases and catalogs, one line for each Region.

In the examples, replace the `example_db` database and `test` table with your own database and table names.

<table>
<tr><th>DDL statement</th><th>Example of an IAM access policy granting access to the resource</th></tr>
<tr><td>ALTER DATABASE</td><td>Allows you to modify the properties for the <code>example_db</code> database.
<pre>{
    "Effect": "Allow",
    "Action": [
       "glue:GetDatabase",
       "glue:UpdateDatabase"
    ],
    "Resource": [
      "arn:aws:glue: us-east-1 :123456789012 :catalog",
      "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db "
    ]
}</pre></td></tr>
<tr><td>CREATE DATABASE</td><td>Allows you to create the database named <code>example_db</code> .
<pre>{
    "Effect": "Allow",
    "Action": [
       "glue:GetDatabase",
       "glue:CreateDatabase"
    ],
    "Resource": [
      "arn:aws:glue: us-east-1 :123456789012 :catalog",
      "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db "
    ]
}</pre></td></tr>
<tr><td>CREATE TABLE</td><td>Allows you to create a table named <code>test</code> in the <code>example_db</code> database.</td></tr>
</table>

DDL statement	Example of an IAM access policy granting access to the resource
	```{ "Sid": "DatabasePermissions", "Effect": "Allow", "Action": ["glue:GetDatabase", "glue:GetDatabases"], "Resource": ["arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db "] }, { "Sid": "TablePermissions", "Effect": "Allow", "Action": ["glue:GetTables", "glue:GetTable", "glue:GetPartitions", "glue:CreateTable"], "Resource": ["arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db ", "arn:aws:glue: us-east-1 :123456789012 :table/example_d b /test"] }```

<table>
<tr><th>DDL statement</th><th>Example of an IAM access policy granting access to the resource</th></tr>
<tr><td>DROP DATABASE</td><td>Allows you to drop the <code>example_db</code> database, including all tables in it.<pre>
{
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:DeleteDatabase",
      "glue:GetTables",
      "glue:GetTable",
      "glue:DeleteTable"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
     "arn:aws:glue: us-east-1 :123456789012 :table/example_d
b /*",
     "arn:aws:glue: us-east-1 :123456789012 :userDefi
nedFunction/ example_db /*"
   ]
 }
</pre></td></tr>
</table>

DDL statement	Example of an IAM access policy granting access to the resource
DROP TABLE	Allows you to drop a partitioned table named `test` in the `example_db` database. If your table does not have partitions, do not include partition actions. `{ "Effect": "Allow", "Action": ["glue:GetDatabase", "glue:GetTable", "glue:DeleteTable", "glue:GetPartitions", "glue:GetPartition", "glue:DeletePartition"], "Resource": ["arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database/ example_db ", "arn:aws:glue: us-east-1 :123456789012 :table/example_db /test"] }`

```
{
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetTable",
      "glue:DeleteTable",
      "glue:GetPartitions",
      "glue:GetPartition",
      "glue:DeletePartition"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
     "arn:aws:glue: us-east-1 :123456789012 :table/example_d
b /test"
   ]
 }
```

<table>
<tr><th>DDL statement</th><th>Example of an IAM access policy granting access to the resource</th></tr>
<tr><td>MSCK REPAIR TABLE</td><td>Allows you to update catalog metadata after you add Hive compatible partitions to the table named <code>test</code> in the <code>example_db</code> database.
<pre>
{
    "Effect": "Allow",
    "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetTable",
        "glue:GetPartitions",
        "glue:GetPartition",
        "glue:BatchCreatePartition"
    ],
    "Resource": [
      "arn:aws:glue: us-east-1 :123456789012 :catalog",
      "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
      "arn:aws:glue: us-east-1 :123456789
012 :table/example_db /test"
    ]
}
</pre></td></tr>
<tr><td>SHOW DATABASES</td><td>Allows you to list all databases in the AWS Glue Data Catalog.
<pre>
{
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetDatabases"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database/*"
   ]
 }
</pre></td></tr>
</table>

<table>
<tr><th>DDL statement</th><th>Example of an IAM access policy granting access to the resource</th></tr>
<tr><td>SHOW TABLES</td><td>Allows you to list all tables in the <code>example_db</code> database.
<pre>
{
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetTables"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
     "arn:aws:glue: us-east-1 :123456789012 :table/example_d
b /*"
   ]
}
</pre></td></tr>
</table>

Configure cross-account access to AWS Glue data catalogs

You can use Athena's cross-account AWS Glue catalog feature to register an AWS Glue catalog from an account other than your own. After you configure the required IAM permissions for AWS Glue and register the catalog as an Athena DataCatalog resource, you can use Athena to run cross-account queries. For information about using the Athena console to register a catalog from another account, see Register a Data Catalog from another account.

For more information about cross-account access in AWS Glue, see Granting cross-account access in the *AWS Glue Developer Guide*.

Before you start

Because this feature uses existing Athena `DataCatalog` resource APIs and functionality to enable cross-account access, we recommend that you read the following resources before you start:

- Connect to data sources - Contains topics on using Athena with AWS Glue, Hive, or Lambda data catalog sources.
- Data Catalog example policies - Shows how to write policies that control access to data catalogs.

- Use the AWS CLI with Hive metastores - Shows how to use the AWS CLI with Hive metastores, but contains use cases applicable to other data sources.

Considerations and limitations

Currently, Athena cross-account AWS Glue catalog access has the following limitations:

- The feature is available only in AWS Regions where Athena engine version 2 or later is supported. For information about Athena engine versions, see Athena engine versioning. To upgrade the engine version for a workgroup, see Change Athena engine versions.
- When you register another account's AWS Glue Data Catalog in your account, you create a regional `DataCatalog` resource that is linked to the other account's data in that particular Region only.
- Currently, `CREATE VIEW` statements that include a cross-account AWS Glue catalog are not supported.
- Catalogs encrypted using AWS managed keys cannot be queried across accounts. For catalogs that you want to query across accounts, use customer managed keys (`KMS_CMK`) instead. For information about the differences between customer managed keys and AWS managed keys, see Customer keys and AWS keys in the *AWS Key Management Service Developer Guide*.

Get started

In the following scenario, the "borrower" account (666666666666) wants to run a `SELECT` query that refers to the AWS Glue catalog that belongs to the "owner" account (999999999999), as in the following example:

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

In the following procedure, Steps 1a and 1b show how to give the borrower account access to the owner account's AWS Glue resources, from both the borrower and owner's side. The example grants access to the database `tpch1000` and the table `customer`. Change these example names to fit your requirements.

Step 1a: Create a borrower role with a policy to access the owner's AWS Glue resources

To create borrower account role with a policy to access to the owner account's AWS Glue resources, you can use the AWS Identity and Access Management (IAM) console or the IAM API. The following procedures use the IAM console.

To create a borrower role and policy to access the owner account's AWS Glue resources

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/ from the borrower account.
2. In the navigation pane, expand **Access management**, and then choose **Policies**.
3. Choose **Create policy**.
4. For **Policy editor**, choose **JSON**.
5. In the policy editor, enter the following policy, and then modify it according to your requirements:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

6. Choose **Next**.
7. On the **Review and create** page, for **Policy name**, enter a name for the policy (for example, **`CrossGluePolicyForBorrowerRole`**).
8. Choose **Create policy**.
9. In the navigation pane, choose **Roles**.
10. Choose **Create role**.
11. On the **Select trusted entity** page, choose **AWS account**, and then choose **Next**.
12. On the **Add permissions** page, enter the name of the policy that you created into the search box (for example, **`CrossGluePolicyForBorrowerRole`**).
13. Select the check box next to the policy name, and then choose **Next**.
14. On the **Name, review, and create** page, for **Role name**, enter a name for the role (for example, **`CrossGlueBorrowerRole`**).

15. Choose **Create role**.

Step 1b: Create an owner policy to grant AWS Glue access to the borrower

To grant AWS Glue access from the owner account (999999999999) to the borrower's role, you can use the AWS Glue console or the AWS Glue PutResourcePolicy API operation. The following procedure uses the AWS Glue console.

To grant AWS Glue access to the borrower account from the owner

1. Sign in to the AWS Glue console at https://console.aws.amazon.com/glue/ from the owner account.
2. In the navigation pane, expand **Data Catalog**, and then choose **Catalog settings**.
3. In the **Permissions** box, enter a policy like the following. For *rolename*, enter the role that the borrower created in Step 1a (for example, **CrossGlueBorrowerRole**). If you want to increase the permission scope, you can use the wild card character * for both the database and table resource types.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::666666666666:user/username",
                    "arn:aws:iam::666666666666:role/rolename"
                ]
            },
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

After you finish, we recommend that you use the AWS Glue API to make some test cross-account calls to confirm that permissions are configured as you expect.

Step 2: The borrower registers the AWS Glue Data Catalog that belongs to the owner account

The following procedure shows you how to use the Athena console to configure the AWS Glue Data Catalog in the owner Amazon Web Services account as a data source. For information about using API operations instead of the console to register the catalog, see (Optional) Use the API to register an Athena Data Catalog that belongs to the owner account.

To register an AWS Glue Data Catalog belonging to another account

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. Expand **Administration**, and then choose **Data sources**.
4. On the upper right, choose **Create data source**.
5. On the **Choose a data source** page, for **Data sources**, select **S3 - AWS Glue Data Catalog**, and then choose **Next**.
6. On the **Enter data source details** page, in the **AWS Glue Data Catalog** section, for **Choose an AWS Glue Data Catalog**, choose **AWS Glue Data Catalog in another account**.
7. For **Data source details**, enter the following information:
   - **Data source name** – Enter the name that you want to use in your SQL queries to refer to the data catalog in the other account.

- **Description –** (Optional) Enter a description of the data catalog in the other account.
- **Catalog ID –** Enter the 12-digit Amazon Web Services account ID of the account to which the data catalog belongs. The Amazon Web Services account ID is the catalog ID.

8. (Optional) Expand **Tags**, and then enter key-value pairs that you want to associate with the data source. For more information about tags, see Tag Athena resources.
9. Choose **Next**.
10. On the **Review and create** page, review the information that you provided, and then choose **Create data source**. The **Data source details** page lists the databases and tags for the data catalog that you registered.
11. Choose **Data sources and catalogs**. The data catalog that you registered is listed in the **Data source name** column.
12. To view or edit information about the data catalog, choose the catalog, and then choose **Actions**, **Edit**.
13. To delete the new data catalog, choose the catalog, and then choose **Actions**, **Delete**.

Step 3: The borrower submits a query

The borrower submits a query that references the catalog using the *catalog.database.table* syntax, as in the following example:

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

Instead of using the fully qualified syntax, the borrower can also specify the catalog contextually by passing it in through the QueryExecutionContext.

(Optional) Configure additional Amazon S3 permissions

- If the borrower account uses an Athena query to write new data to a table in the owner account, the owner will not automatically have access to this data in Amazon S3, even though the table exists in the owner's account. This is because the borrower is the object owner of the information in Amazon S3 unless otherwise configured. To grant the owner access to the data, set the permissions on the objects accordingly as an additional step.
- Certain cross-account DDL operations like MSCK REPAIR TABLE require Amazon S3 permissions. For example, if the borrower account is performing a cross-account `MSCK REPAIR` operation against a table in the owner account that has its data in an owner account S3 bucket, that bucket must grant permissions to the role assumed by the borrower for the query to succeed.

For information about granting bucket permissions, see How do I set ACL bucket permissions? in the *Amazon Simple Storage Service User Guide*.

(Optional) Use a catalog dynamically

In some cases you might want to quickly perform testing against a cross-account AWS Glue catalog without the prerequisite step of registering it. You can dynamically perform cross-account queries without creating the `DataCatalog` resource object if the required IAM and Amazon S3 permissions are correctly configured as described earlier in this document.

To explicitly reference a catalog without registration, use the syntax in the following example:

```
SELECT * FROM "glue:arn:aws:glue:us-east-1:999999999999:catalog".tpch1000.customer
```

Use the format "`glue:<arn>`", where *`<arn>`* is the AWS Glue Data Catalog ARN that you want to use. In the example, Athena uses this syntax to dynamically point to account 999999999999's AWS Glue data catalog as if you had separately created a `DataCatalog` object for it.

Notes for using dynamic catalogs

When you use dynamic catalogs, remember the following points.

- Use of a dynamic catalog requires the IAM permissions that you normally use for Athena Data Catalog API operations. The main difference is that the Data Catalog resource name follows the `glue:*` naming convention.
- The catalog ARN must belong to the same Region where the query is being run.
- When using a dynamic catalog in a DML query or view, surround it with escaped double quotation marks (\"). When using a dynamic catalog in a DDL query, surround it with backtick characters (`).

(Optional) Use the API to register an Athena Data Catalog that belongs to the owner account

Instead of using the Athena console as described in Step 2, it is possible to use API operations to register the Data Catalog that belongs to the owner account.

The creator of the Athena DataCatalog resource must have the necessary permissions to run the Athena CreateDataCatalog API operation. Depending on your requirements, access to additional API operations might be necessary. For more information, see Data Catalog example policies.

The following `CreateDataCatalog` request body registers an AWS Glue catalog for cross-account access:

```
# Example CreateDataCatalog request to register a cross-account Glue catalog:
{
    "Description": "Cross-account Glue catalog",
    "Name": "ownerCatalog",
    "Parameters": {"catalog-id" : "999999999999"  # Owner's account ID
    },
    "Type": "GLUE"
}
```

The following sample code uses a Java client to create the `DataCatalog` object.

```
# Sample code to create the DataCatalog through Java client
CreateDataCatalogRequest request = new CreateDataCatalogRequest()
    .withName("ownerCatalog")
    .withType(DataCatalogType.GLUE)
    .withParameters(ImmutableMap.of("catalog-id", "999999999999"));

athenaClient.createDataCatalog(request);
```

After these steps, the borrower should see `ownerCatalog` when it calls the ListDataCatalogs API operation.

Additional resources

- Register a Data Catalog from another account
- Configure cross-account access to a shared AWS Glue Data Catalog using Amazon Athena in the *AWS Prescriptive Guidance Patterns* guide.
- Query cross-account AWS Glue Data Catalogs using Amazon Athena in the *AWS Big Data Blog*
- Granting cross-account access in the *AWS Glue Developer Guide*

Configure access from Athena to encrypted metadata in the AWS Glue Data Catalog

If you use the AWS Glue Data Catalog with Amazon Athena, you can enable encryption in the AWS Glue Data Catalog using the AWS Glue console or the API. For information, see Encrypting your data catalog in the *AWS Glue Developer Guide*.

If the AWS Glue Data Catalog is encrypted, you must add the following actions to all policies that are used to access Athena:

```
{
 "Version": "2012-10-17",
 "Statement": {
 "Effect": "Allow",
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt",
           "kms:Encrypt"
      ],
     "Resource": "(arn of the key used to encrypt the catalog)"
   }
}
```

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Configure access to workgroups and tags

A workgroup is a resource managed by Athena. Therefore, if your workgroup policy uses actions that take `workgroup` as an input, you must specify the workgroup's ARN as follows, where *`workgroup-name`* is the name of your workgroup:

```
"Resource": [arn:aws:athena:region:AWSAcctID:workgroup/workgroup-name]
```

For example, for a workgroup named `test_workgroup` in the `us-west-2` region for Amazon Web Services account `123456789012`, specify the workgroup as a resource using the following ARN:

```
"Resource":["arn:aws:athena:us-east-2:123456789012:workgroup/test_workgroup"]
```

To access trusted identity propagation (TIP) enabled workgroups, IAM Identity Center users must be assigned to the `IdentityCenterApplicationArn` that is returned by the response of the Athena GetWorkGroup API action.

- For a list of workgroup policies, see the section called "Example workgroup policies".
- For a list of tag-based policies for workgroups, see Use tag-based IAM access control policies.
- For more information about creating IAM policies for workgroups, see Use IAM policies to control workgroup access.

- For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference.
- For more information about IAM policies, see Creating policies with the visual editor in the *IAM User Guide*.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Use IAM policies to control workgroup access

To control access to workgroups, use resource-level IAM permissions or identity-based IAM policies. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

> **Note**
>
> To access trusted identity propagation enabled workgroups, IAM Identity Center users must be assigned to the `IdentityCenterApplicationArn` that is returned by the response of the Athena GetWorkGroup API action.

The following procedure is specific to Athena.

For IAM-specific information, see the links listed at the end of this section. For information about example JSON workgroup policies, see Example workgroup policies.

To use the visual editor in the IAM console to create a workgroup policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**, and then choose **Create policy**.
3. On the **Visual editor** tab, choose **Choose a service**. Then choose Athena to add to the policy.
4. Choose **Select actions**, and then choose the actions to add to the policy. The visual editor shows the actions available in Athena. For more information, see Actions, resources, and condition keys for Amazon Athena in the *Service Authorization Reference*.
5. Choose **add actions** to type a specific action or use wildcards (*) to specify multiple actions.

By default, the policy that you are creating allows the actions that you choose. If you chose one or more actions that support resource-level permissions to the `workgroup` resource in Athena, then the editor lists the `workgroup` resource.

6. Choose **Resources** to specify the specific workgroups for your policy. For example JSON workgroup policies, see [Example workgroup policies](#).
7. Specify the `workgroup` resource as follows:

```
arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>
```

8. Choose **Review policy**, and then type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy summary to make sure that you granted the intended permissions.
9. Choose **Create policy** to save your new policy.
10. Attach this identity-based policy to a user, a group, or role.

For more information, see the following topics in the *Service Authorization Reference* and *IAM User Guide*:

- Actions, resources, and condition keys for Amazon Athena
- Creating policies with the visual editor
- Adding and removing IAM policies
- Controlling access to resources

For example JSON workgroup policies, see Example workgroup policies.

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference.

Example workgroup policies

This section includes example policies you can use to enable various actions on workgroups. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

A workgroup is an IAM resource managed by Athena. Therefore, if your workgroup policy uses actions that take `workgroup` as an input, you must specify the workgroup's ARN as follows:

```
"Resource": [arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>]
```

Where *<workgroup-name>* is the name of your workgroup. For example, for workgroup named `test_workgroup`, specify it as a resource as follows:

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"]
```

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference. For more information about IAM policies, see Creating policies with the visual editor in the *IAM User Guide*. For more information about creating IAM policies for workgroups, see Use IAM policies to control workgroup access.

- Example policy for full access to all workgroups
- Example policy for full access to a specified workgroup
- Example policy for running queries in a specified workgroup
- Example policy for running queries in the primary workgroup
- Example policy for management operations on a specified workgroup
- Example policy for listing workgroups
- Example policy for running and stopping queries in a specific workgroup
- Example policy for working with named queries in a specific workgroup
- Example policy for working with Spark notebooks

Example Example policy for full access to all workgroups

The following policy allows full access to all workgroup resources that might exist in the account. We recommend that you use this policy for those users in your account that must administer and manage workgroups for all other users.

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

```
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example Example policy for full access to a specified workgroup

The following policy allows full access to the single specific workgroup resource, named `workgroupA`. You could use this policy for users with full control over a particular workgroup.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
```

```
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:GetWorkGroup",
                "athena:CreateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
    ]
}
```

Example Example policy for running queries in a specified workgroup

In the following policy, a user is allowed to run queries in the specified `workgroupA`, and view them. The user is not allowed to perform management tasks for the workgroup itself, such as updating or deleting it. Note that the example policy does not limit users to only this workgroup or deny access to other workgroups.

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
```

```
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
    ]
}
```

Example Example policy for running queries in the primary workgroup

You can modify the preceding example to allow a particular user to also run queries in the primary workgroup.

> **Note**
>
> We recommend that you add the primary workgroup resource for all users who are otherwise configured to run queries in their designated workgroups. Adding this resource

to their workgroup user policies is useful in case their designated workgroup is deleted or is disabled. In this case, they can continue running queries in the primary workgroup.

To allow users in your account to run queries in the primary workgroup, add a line that contains the ARN of the primary workgroup to the resource section of the Example policy for running queries in a specified workgroup, as in the following example.

```
arn:aws:athena:us-east-1:123456789012:workgroup/primary"
```

Example Example policy for management operations on a specified workgroup

In the following policy, a user is allowed to create, delete, obtain details, and update a workgroup `test_workgroup`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:GetWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
        }
    ]
}
```

Example Example policy for listing workgroups

The following policy allows all users to list all workgroups:

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

Example Example policy for running and stopping queries in a specific workgroup

In this policy, a user is allowed to run queries in the workgroup:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
        }
    ]
}
```

Example Example policy for working with named queries in a specific workgroup

In the following policy, a user has permissions to create, delete, and obtain information about named queries in the specified workgroup:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
     ]
        }
    ]
}
```

Example Example policy for working with Spark notebooks in Athena

Use a policy like the following to work with Spark notebooks in Athena.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreatingWorkGroupWithDefaults",
            "Action": [
                "athena:CreateWorkGroup",
                "s3:CreateBucket",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:GetBucketLocation",
                "athena:ImportNotebook"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*",
                "arn:aws:s3:::123456789012-us-east-1-athena-results-bucket-*",
                "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-*",
                "arn:aws:iam::123456789012:policy/service-role/
AWSAthenaSparkRolePolicy-*"
            ]
        },
```

```
        {
            "Sid": "AllowRunningCalculations",
            "Action": [
                "athena:ListWorkGroups",
                "athena:GetWorkGroup",
                "athena:StartSession",
                "athena:CreateNotebook",
                "athena:ListNotebookMetadata",
                "athena:ListNotebookSessions",
                "athena:GetSessionStatus",
                "athena:GetSession",
                "athena:GetNotebookMetadata",
                "athena:CreatePresignedNotebookUrl"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*"
        },
        {
            "Sid": "AllowListWorkGroupAndEngineVersions",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Use IAM Identity Center enabled Athena workgroups

Trusted identity propagation is an AWS IAM Identity Center feature that administrators of connected AWS services can use to grant and audit access to service data. Access to this data is based on user attributes such as group associations. Setting up trusted identity propagation requires collaboration between the administrators of connected AWS services and the IAM Identity Center administrators. For more information, see Prerequisites and considerations.

With IAM Identity Center, you can manage sign-in security for your workforce identities, also known as workforce users. IAM Identity Center provides one place where you can create or connect workforce users and centrally manage their access across all their AWS accounts and applications. You can use multi-account permissions to assign these users access to AWS accounts. You can use application assignments to assign your users access to IAM Identity Center enabled applications, cloud applications, and customer Security Assertion Markup Language (SAML 2.0) applications. For more information, see Trusted identity propagation across applications in the *AWS IAM Identity Center User Guide*.

Athena SQL support for trusted identity propagation is available in both EMR Studio and SageMaker Unified Studio. Each platform provides specific interfaces for using TIP with Athena.

When using Athena SQL in EMR Studio with IAM Identity Center identities, you have two workgroup options:

- **Regular WorkGroups –** No user/group assignments needed.
- **IAM Identity Center enabled workgroups –** Requires assigning users/groups through IAM Identity Center console or API.

For both options, you can run queries using the Athena SQL interface in EMR Studio with IAM Identity Center enabled.

Considerations and limitations

When you use trusted identity propagation with Amazon Athena, consider the following points:

- You cannot change the authentication method for the workgroup after the workgroup is created.
  - Existing Athena SQL workgroups cannot be modified to support IAM Identity Center enabled workgroups. Existing Athena SQL workgroups can propagate identity to downstream services.
  - IAM Identity Center enabled workgroups cannot be modified to support resource-level IAM permissions or identity based IAM policies.
- To access trusted identity propagation enabled workgroups, IAM Identity Center users must be assigned to the `IdentityCenterApplicationArn` that is returned by the response of the Athena GetWorkGroup API action.
- Amazon S3 Access Grants must be configured to use trusted identity propagation identities. For more information, see S3 Access Grants and corporate directory identities in the *Amazon S3 User Guide*.
- IAM Identity Center enabled Athena workgroups require Lake Formation to be configured to use IAM Identity Center identities. For configuration information, see Integrating IAM Identity Center in the *AWS Lake Formation Developer Guide*.

- By default, queries time out after 30 minutes in IAM Identity Center enabled workgroups. You can request a query timeout increase, but the maximum a query can run in trusted identity propagation workgroups is one hour.
- User or group entitlement changes in trusted identity propagation workgroups can require up to an hour to take effect.
- Queries in an Athena workgroup that uses trusted identity propagation cannot be run directly from the Athena console. They must be run from the Athena interface in an EMR Studio that has IAM Identity Center enabled. For more information about using Athena in EMR Studio, see Use the Amazon Athena SQL editor in EMR Studio in the *Amazon EMR Management Guide*.
- Trusted identity propagation is not compatible with the following Athena features.
  - `aws:CalledVia` context keys for IAM Identity Center enabled workgroups.
  - Athena for Spark workgroups.
  - Federated access to the Athena API.
  - Federated access to Athena using Lake Formation and the Athena JDBC and ODBC drivers.
- You can use trusted identity propagation with Athena only in the following AWS Regions:
  - `us-east-2` – US East (Ohio)
  - `us-east-1` – US East (N. Virginia)
  - `us-west-1` – US West (N. California)
  - `us-west-2` – US West (Oregon)
  - `af-south-1` – Africa (Cape Town)
  - `ap-east-1` – Asia Pacific (Hong Kong)
  - `ap-southeast-3` – Asia Pacific (Jakarta)
  - `ap-south-1` – Asia Pacific (Mumbai)
  - `ap-northeast-3` – Asia Pacific (Osaka)
  - `ap-northeast-2` – Asia Pacific (Seoul)
  - `ap-southeast-1` – Asia Pacific (Singapore)
  - `ap-southeast-2` – Asia Pacific (Sydney)
  - `ap-northeast-1` – Asia Pacific (Tokyo)
  - `ca-central-1` – Canada (Central)
  - `eu-central-1` – Europe (Frankfurt)
  - `eu-central-2` – Europe (Zurich)

- `eu-west-1` – Europe (Ireland)
- `eu-west-2` – Europe (London)
- `eu-south-1` – Europe (Milan)
- `eu-west-3` – Europe (Paris)
- `eu-north-1` – Europe (Stockholm)
- `me-south-1` – Middle East (Bahrain)
- `sa-east-1` – South America (São Paulo)

Required permissions

The IAM user of the admin who creates the IAM Identity Center enabled workgroup in the Athena console must have the following policies attached.

- The `AmazonAthenaFullAccess` managed policy. For details, see [AWS managed policy: AmazonAthenaFullAccess](#).
- The following inline policy that allows IAM and IAM Identity Center actions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:createRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListRoles",
                "iam:PassRole",
                "identitystore:ListUsers",
                "identitystore:ListGroups",
                "identitystore:CreateUser",
                "identitystore:CreateGroup",
                "sso:ListInstances",
                "sso:CreateInstance",
                "sso:DeleteInstance",
                "sso:DescribeUser",
                "sso:DescribeGroup",
                "sso:ListTrustedTokenIssuers",
                "sso:DescribeTrustedTokenIssuer",
                "sso:ListApplicationAssignments",
```

```
                "sso:DescribeRegisteredRegions",
                "sso:GetManagedApplicationInstance",
                "sso:GetSharedSsoConfiguration",
                "sso:PutApplicationAssignmentConfiguration",
                "sso:CreateApplication",
                "sso:DeleteApplication",
                "sso:PutApplicationGrant",
                "sso:PutApplicationAuthenticationMethod",
                "sso:PutApplicationAccessScope",
                "sso:ListDirectoryAssociations",
                "sso:CreateApplicationAssignment",
                "sso:DeleteApplicationAssignment",
                "organizations:ListDelegatedAdministrators",
                "organizations:DescribeAccount",
                "organizations:DescribeOrganization",
                "organizations:CreateOrganization",
                "sso-directory:SearchUsers",
                "sso-directory:SearchGroups",
                "sso-directory:CreateUser"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Creating an IAM Identity Center enabled Athena workgroup

The following procedure shows the steps and options related to creating an IAM Identity Center enabled Athena workgroup. For a description of the other configuration options available for Athena workgroups, see Create a workgroup.

To create an SSO enabled workgroup in the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena console navigation pane, choose **Workgroups**.
3. On the **Workgroups** page, choose **Create workgroup**.
4. On the **Create workgroup** page, for **Workgroup name**, enter a name for the workgroup.
5. For **Analytics engine**, use the **Athena SQL** default.

6. For **Authentication**, choose **IAM Identity Center**.
7. For **Service role for IAM Identity Center access**, choose an existing service role, or create a new one.

   Athena requires permissions to access IAM Identity Center for you. A service role is required for Athena to do this. A service role is an IAM role that you manage that authorizes an AWS service to access other AWS services on your behalf. To query federated catalogs or run UDF, update service role with corresponding Lambda permissions. For more information, see [Creating a role to delegate permissions to an AWS service]() in the *IAM User Guide*.
8. Expand **Query result configuration**, and then enter or choose an Amazon S3 path for **Location of query result**.
9. (Optional) Choose **Encrypt query results**. By default, SSE-S3 is supported. To use SSE-KMS and CSE-KMS with query result location, provide grants to the **Service role for IAM Identity Center** from Amazon S3 Access Grants. For more information, see [Sample role policy]().
10. (Optional) Choose **Create user identity based S3 prefix**.

    When you create an IAM Identity Center enabled workgroup, the **Enable S3 Access Grants** option is selected by default. You can use Amazon S3 Access Grants to control access to Athena query results locations (prefixes) in Amazon S3. For more information about Amazon S3 Access Grants, see [Managing access with Amazon S3 Access Grants]().

    In Athena workgroups that use IAM Identity Center authentication, you can enable the creation of identity based query result locations that are governed by Amazon S3 Access Grants. These user identity based Amazon S3 prefixes let users in an Athena workgroup keep their query results isolated from other users in the same workgroup.

    When you enable the user prefix option, Athena appends the user ID as an Amazon S3 path prefix to the query result output location for the workgroup (for example, `s3://amzn-s3-demo-bucket/${user_id}`). To use this feature, you must configure Access Grants to allow only the user permission to the location that has the `user_id` prefix. For a sample Amazon S3 Access Grants location role policy that restricts access to Athena query results, see [Sample role policy]().

> **Note**
>
> Selecting the user identity S3 prefix option automatically enables the override client-side settings option for the workgroup, as described in the next step. The override client-side settings option is a requirement for the user identity prefix feature.

11. Expand **Settings**, and then confirm that **Override client-side settings** is selected.

    When you select **Override client-side settings**, workgroup settings are enforced at the workgroup level for all clients in the workgroup. For more information, see Override client-side settings.
12. (Optional) Make any other configuration settings that you require as described in Create a workgroup.
13. Choose **Create workgroup**.
14. Use the **Workgroups** section of the Athena console to assign users or groups from your IAM Identity Center directory to your IAM Identity Center enabled Athena workgroup.

Sample role policy

The following sample shows a policy for a role to attach to an Amazon S3 Access Grant location that restricts access to Athena query results.

```
{
    "Statement": [{
        "Action": ["s3:*"],
        "Condition": {
            "ArnNotEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-grants/default"
            },
            "StringNotEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Deny",
        "Resource": "*",
        "Sid": "ExplicitDenyS3"
    }, {
        "Action": ["kms:*"],
```

```
        "Effect": "Deny",
        "NotResource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "ExplictDenyKMS"
    }, {
        "Action": ["s3:ListMultipartUploadParts", "s3:GetObject"],
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelReadPermissions"
    }, {
        "Action": ["s3:PutObject", "s3:AbortMultipartUpload"],
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelWritePermissions"
    }, {
        "Action": "s3:ListBucket",
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            },
            "StringLikeIfExists": {
```

```
                "s3:prefix": ["${identitystore:UserId}", "${identitystore:UserId}/*"]
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION",
        "Sid": "BucketLevelReadPermissions"
    }, {
        "Action": ["kms:GenerateDataKey", "kms:Decrypt"],
        "Effect": "Allow",
        "Resource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "KMSPermissions"
    }],
    "Version": "2012-10-17"
}
```

Configure minimum encryption for a workgroup

As an administrator of an Athena SQL workgroup, you can enforce a minimal level of encryption in Amazon S3 for all query results from the workgroup. You can use this feature to ensure that query results are never stored in an Amazon S3 bucket in an unencrypted state.

When users in a workgroup with minimum encryption enabled submit a query, they can only set the encryption to the minimum level that you configure, or to a higher level if one is available. Athena encrypts query results at either the level specified when the user runs the query or at the level set in the workgroup.

The following levels are available:

- **Basic** – Amazon S3 server side encryption with Amazon S3 managed keys (**SSE_S3**).
- **Intermediate** – Server Side encryption with KMS managed keys (**SSE_KMS**).
- **Advanced** – Client side encryption with KMS managed keys (**CSE_KMS**).

Considerations and limitations

- The minimum encryption feature is not available for Apache Spark enabled workgroups.
- The minimum encryption feature is functional only when the workgroup does not enable the [Override client-side settings] option.
- If the workgroup has the **Override client-side settings** option enabled, the workgroup encryption setting prevails, and the minimum encryption setting has no effect.

- There is no cost to enable this feature.

Enable minimum encryption for a workgroup

You can enable a minimum encryption level for the query results from your Athena SQL workgroup when you create or update the workgroup. To do this, you can use the Athena console, Athena API, or AWS CLI.

Use the Athena console to enable minimum encryption

To get started creating or editing your workgroup using the Athena console, see Create a workgroup or Edit a workgroup. When configuring your workgroup, use the following steps to enable minimum encryption.

To configure the minimum encryption level for workgroup query results

1. Clear the **Override client-side settings** option, or verify that it is not selected.
2. Select the **Encrypt query results** option.
3. For **Encryption type**, select the encryption method that you want Athena to use for your workgroup's query results (**SSE_S3**, **SSE_KMS**, or **CSE_KMS**). These encryption types correspond to basic, intermediate, and advanced security levels.
4. To enforce the encryption method that you chose as the minimum level of encryption for all users, select **Set *`encryption_method`* as minimum encryption**.

   When you select this option, a table shows the encryption hierarchy and encryption levels that users will be allowed when the encryption type that you choose becomes the minimum.
5. After you create your workgroup or update your workgroup configuration, choose **Create workgroup** or **Save changes**.

Use the Athena API or AWS CLI to enable minimum encryption

When you use the CreateWorkGroup or UpdateWorkGroup API to create or update an Athena SQL workgroup, set EnforceWorkGroupConfiguration to `false`, EnableMinimumEncryptionConfiguration to `true`, and use the EncryptionOption to specify the type of encryption.

In the AWS CLI, use the `create-work-group` or `update-work-group` command with the `--configuration` or `--configuration-updates` parameters and specify the options corresponding to those for the API.

Configure access to prepared statements

This topic covers IAM permissions for prepared statements in Amazon Athena. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

For more information about prepared statements, see Use parameterized queries.

The following IAM permissions are required for creating, managing, and executing prepared statements.

```
athena:CreatePreparedStatement
athena:UpdatePreparedStatement
athena:GetPreparedStatement
athena:ListPreparedStatements
athena:DeletePreparedStatement
```

Use these permissions as shown in the following table.

To do this	Use these permissions
Run a `PREPARE` query	`athena:StartQueryExecution` `athena:CreatePreparedStatement`
Re-run a `PREPARE` query to update an existing prepared statement	`athena:StartQueryExecution` `athena:UpdatePreparedStatement`
Run an `EXECUTE` query	`athena:StartQueryExecution` `athena:GetPreparedStatement`
Run a `DEALLOCATE PREPARE` query	`athena:StartQueryExecution` `athena:DeletePreparedStatement`

Example

The following example IAM policy grants permissions to manage and run prepared statements on a specified account ID and workgroup.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:CreatePreparedStatement",
                "athena:UpdatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:DeletePreparedStatement",
                "athena:ListPreparedStatements"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/<workgroup-name>"
            ]
        }
    ]
}
```

Use CalledVia context keys for Athena

When a principal makes a request to AWS, AWS gathers the request information into a *request context* that evaluates and authorizes the request. You can use the `Condition` element of a JSON policy to compare keys in the request context with key values that you specify in your policy. *Global condition context keys* are condition keys with an `aws:` prefix.

About the aws:CalledVia context key

You can use the aws:CalledVia global condition context key to compare the services in the policy with the services that made requests on behalf of the IAM principal (user or role). When a principal makes a request to an AWS service, that service might use the principal's credentials to make subsequent requests to other services. The `aws:CalledVia` key contains an ordered list of each service in the chain that made requests on the principal's behalf.

By specifying a service principal name for the `aws:CalledVia` context key, you can make the context key AWS service-specific. For example, you can use the `aws:CalledVia` condition key to limit requests to only those made from Athena. To use the `aws:CalledVia` condition key in a policy with Athena, you specify the Athena service principal name `athena.amazonaws.com`, as in the following example.

```
 ...
    "Condition": {
        "ForAnyValue:StringEquals": {
            "aws:CalledVia": "athena.amazonaws.com"
        }
    }
...
```

You can use the `aws:CalledVia` context key to ensure that callers only have access to a resource (like a Lambda function) if they call the resource from Athena.

> **Note**
>
> The `aws:CalledVia` context key is not compatible with the trusted identity propagation feature.

Add a CalledVia context key for access to Lambda functions

Athena requires the caller to have `lambda:InvokeFunction` permissions in order to invoke the Lambda function associated with the query. The following statement specifies that the user can invoke Lambda functions only from Athena.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:OneAthenaLambdaFunction",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
```

```
            }
        }
    ]
}
```

The following example shows the addition of the previous statement to a policy that allows a user to run and read a federated query. Principals who are allowed to perform these actions can run queries that specify Athena catalogs associated with a federated data source. However, the principal cannot access the associated Lambda function unless the function is invoked through Athena.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "s3:PutObject",
                "s3:GetObject",
                "athena:StartQueryExecution",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkGroupName",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillPrefix*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
```

```
            "Action":
                [
                "s3:ListBucket",
                "s3:GetBucketLocation"
                ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket"
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
            }
        }
    ]
}
```

For more information about `CalledVia` condition keys, see AWS global condition context keys in the *IAM User Guide*.

Allow access to the Athena Data Connector for External Hive Metastore

The permission policy examples in this topic demonstrate required allowed actions and the resources for which they are allowed. Examine these policies carefully and modify them according to your requirements before you attach similar permissions policies to IAM identities.

- Example Policy to Allow an IAM Principal to Query Data Using Athena Data Connector for External Hive Metastore
- Example Policy to Allow an IAM Principal to Create an Athena Data Connector for External Hive Metastore

Example – Allow an IAM principal to query data using Athena Data Connector for External Hive Metastore

The following policy is attached to IAM principals in addition to the AWS managed policy: AmazonAthenaFullAccess, which grants full access to Athena actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:layer:MyAthenaLambdaLayer:*"
            ]
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillLocation"
        }
    ]
}
```

Explanation of permissions

Allowed actions	Explanation

`"s3:GetBucketLocation",`
`"s3:GetObject",`
`"s3:ListBucket",`
`"s3:PutObject",`
`"s3:ListMultipartUploadParts",`
`"s3:AbortMultipartUpload"`
	`s3` actions allow reading from and writing to the resource specified as `"arn:aws:s3:::` *`MyLambdaSpillBucket`* `/`*`MyLambdaSpillLocation`* `"`, where *`MyLambdaSpillLocation`* identifies the spill bucket that is specified in the configuration of the Lambda function or functions being invoked. The *`arn:aws:lambda:*:`* *`MyAWSAcctId`* *`:layer:MyAthenaLambdaLayer`* *`:*`* resource identifier is required only if you use a Lambda layer to create custom runtime dependencies to reduce function artifact size at deployment time. The `*` in the last position is a wildcard for layer version.

`"lambda:GetFunction",`
`"lambda:GetLayerVersion",`
`"lambda:InvokeFunction"`
	Allows queries to invoke the AWS Lambda functions specified in the `Resource` block. For example, `arn:aws:lambda:*:` *`MyAWSAcctId`* `:function:` *`MyAthenaLambdaFunction`* , where *`MyAthenaLambdaFunction`* specifies the name of a Lambda function to be invoked. Multiple functions can be specified as shown in the example.

Example – Allow an IAM principal to create an Athena Data Connector for External Hive Metastore

The following policy is attached to IAM principals in addition to the AWS managed policy: AmazonAthenaFullAccess, which grants full access to Athena actions.

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

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:ListFunctions",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion",
                "lambda:UpdateFunctionConfiguration",
                "lambda:PutFunctionConcurrency",
                "lambda:DeleteFunctionConcurrency"
            ],
            "Resource": "arn:aws:lambda:*:111122223333:
 function: MyAthenaLambdaFunctionsPrefix*"
        }
    ]
}
```

Explanation of Permissions

Allows queries to invoke the AWS Lambda functions for the AWS Lambda functions specified in the `Resource` block. For example, `arn:aws:lambda:*:MyAWSAcctId:function:MyAthenaLambdaFunction`, where *`MyAthenaLambdaFunction`* specifies the name of a Lambda function to be invoked. Multiple functions can be specified as shown in the example.

Allow Lambda function access to external Hive metastores

To invoke a Lambda function in your account, you must create a role that has the following permissions:

- `AWSLambdaVPCAccessExecutionRole` – An [AWS Lambda execution role](#) permission to manage elastic network interfaces that connect your function to a VPC. Ensure that you have a sufficient number of network interfaces and IP addresses available.
- `AmazonAthenaFullAccess` – The [AmazonAthenaFullAccess](#) managed policy grants full access to Athena.

- An Amazon S3 policy to allow the Lambda function to write to S3 and to allow Athena to read from S3.

For example, the following policy defines the permission for the spill location `s3:\\mybucket\spill`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/spill"
            ]
        }
    ]
}
```

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Create Lambda functions

To create a Lambda function in your account, function development permissions or the `AWSLambdaFullAccess` role are required. For more information, see Identity-based IAM policies for AWS Lambda.

Because Athena uses the AWS Serverless Application Repository to create Lambda functions, the superuser or administrator who creates Lambda functions should also have IAM policies to allow Athena federated queries.

Configure permissions for catalog registration and metadata API operations

For API access to catalog registration and metadata operations, you can use the [AmazonAthenaFullAccess managed policy](). If you do not use the `AmazonAthenaFullAccess` policy, add the following API operations to your Athena policies:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListDataCatalogs",
                "athena:GetDataCatalog",
                "athena:CreateDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
                "athena:GetTableMetadata",
                "athena:ListTableMetadata"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Call a Lambda function across regions

By default, Athena invokes Lambda functions defined in the same region. To invoke a Lambda function in an AWS Region other than the region in which you are running Athena queries, use the full ARN of the Lambda function.

The following example shows how a catalog in the Europe (Frankfurt) Region can specify a Lambda function in the US East (N. Virginia) Region to fetch data from the Hive metastore in the Europe (Frankfurt) Region.

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

When you specify the full ARN in this way, Athena can call the `external-hms-service-new` Lambda function on `us-east-1` to fetch the Hive metastore data from `eu-central-1`.

> **Note**
>
> The catalog should be registered in the same AWS Region that you use to run Athena queries.

Call a Lambda function across accounts

Sometimes you might require access to a Hive metastore from a different account. For example, to run a Hive metastore, you might use an account that is different from the one that you use for Athena queries. Different groups or teams might run Hive metastore with different accounts inside their VPC. Or you might want to access metadata from different Hive metastores from different groups or teams.

Athena uses the [AWS Lambda support for cross account access](https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html#permissions-resource-xaccountinvoke) to enable cross account access for Hive Metastores.

> **Note**
>
> Note that cross account access for Athena normally implies cross account access for both metadata and data in Amazon S3.

Imagine the following scenario:

- Account `111122223333` sets up the Lambda function `external-hms-service-new` on us-east-1 in Athena to access a Hive Metastore running on an EMR cluster.
- Account `111122223333` wants to allow account 444455556666 to access the Hive Metastore data.

To grant account `444455556666` access to the Lambda function `external-hms-service-new`, account `111122223333` uses the following AWS CLI `add-permission` command. The command has been formatted for readability.

```
$ aws --profile perf-test lambda add-permission
```

```
      --function-name external-hms-service-new
      --region us-east-1
      --statement-id Id-ehms-invocation2
      --action "lambda:InvokeFunction"
      --principal arn:aws:iam::444455556666:user/perf1-test
{
    "Statement": "{\"Sid\":\"Id-ehms-invocation2\",
                   \"Effect\":\"Allow\",
                   \"Principal\":{\"AWS\":\"arn:aws:iam::444455556666:user/perf1-test
\"},
                   \"Action\":\"lambda:InvokeFunction\",
                   \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}"
}
```

To check the Lambda permission, use the `get-policy` command, as in the following example. The command has been formatted for readability.

```
$ aws --profile perf-test lambda get-policy
      --function-name arn:aws:lambda:us-east-1:111122223333:function:external-hms-
service-new
      --region us-east-1
{
    "RevisionId": "711e93ea-9851-44c8-a09f-5f2a2829d40f",
    "Policy": "{\"Version\":\"2012-10-17\",
                \"Id\":\"default\",
                \"Statement\":[{\"Sid\":\"Id-ehms-invocation2\",
                                \"Effect\":\"Allow\",
                                \"Principal\":{\"AWS\":
\"arn:aws:iam::444455556666:user/perf1-test\"},
                                \"Action\":\"lambda:InvokeFunction\",
                                \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}]}"
}
```

After adding the permission, you can use a full ARN of the Lambda function on `us-east-1` like the following when you define catalog `ehms`:

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

For information about cross region invocation, see Call a Lambda function across regions earlier in this topic.

Grant cross-account access to data

Before you can run Athena queries, you must grant cross account access to the data in Amazon S3. You can do this in one of the following ways:

- Update the access control list policy of the Amazon S3 bucket with a canonical user ID.
- Add cross account access to the Amazon S3 bucket policy.

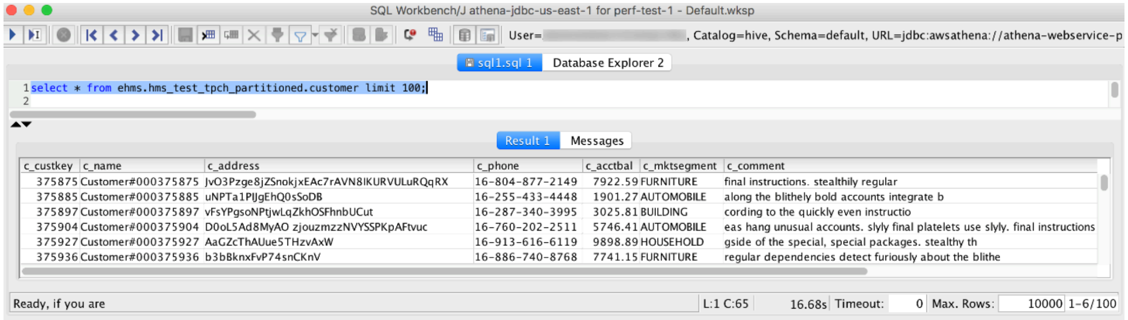
For example, add the following policy to the Amazon S3 bucket policy in the account 111122223333 to allow account 444455556666 to read data from the Amazon S3 location specified.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234567890123",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:user/perf1-test"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::athena-test/lambda/dataset/*"
        }
    ]
}
```

> **Note**
>
> You might need to grant cross account access to Amazon S3 not only to your data, but also to your Amazon S3 spill location. Your Lambda function spills extra data to the spill location when the size of the response object exceeds a given threshold. See the beginning of this topic for a sample policy.

In the current example, after cross account access is granted to `444455556666,` `444455556666` can use catalog `ehms` in its own `account` to query tables that are defined in account `111122223333`.

In the following example, the SQL Workbench profile `perf-test-1` is for account `444455556666`. The query uses catalog `ehms` to access the Hive metastore and the Amazon S3 data in account `111122223333`.



Permissions required to create connector and Athena catalog

To invoke Athena `CreateDataCatalog` you must create a role that has the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ECR",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "arn:aws:ecr:*:*:repository/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "glue:TagResource",
        "glue:GetConnection",
        "glue:CreateConnection",
        "glue:DeleteConnection",
        "glue:UpdateConnection",
        "serverlessrepo:CreateCloudFormationTemplate",
        "serverlessrepo:GetCloudFormationTemplate",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStacks",
        "cloudformation:CreateChangeSet",
```

```
            "cloudformation:DescribeAccountLimits",
            "cloudformation:CreateStackSet",
            "cloudformation:ValidateTemplate",
            "cloudformation:CreateUploadBucket",
            "cloudformation:DescribeStackDriftDetectionStatus",
            "cloudformation:ListExports",
            "cloudformation:ListStacks",
            "cloudformation:EstimateTemplateCost",
            "cloudformation:ListImports",
            "iam:AttachRolePolicy",
            "iam:DetachRolePolicy",
            "iam:DeleteRolePolicy",
            "iam:PutRolePolicy",
            "iam:GetRolePolicy",
            "iam:CreateRole",
            "iam:TagRole",
            "iam:DeleteRole",
            "iam:GetRole",
            "iam:PassRole",
            "iam:ListRoles",
            "iam:ListAttachedRolePolicies",
            "iam:ListRolePolicies",
            "iam:GetPolicy",
            "iam:UpdateRole",
            "lambda:InvokeFunction",
            "lambda:GetFunction",
            "lambda:DeleteFunction",
            "lambda:CreateFunction",
            "lambda:TagResource",
            "lambda:ListFunctions",
            "lambda:GetAccountSettings",
            "lambda:ListEventSourceMappings",
            "lambda:ListVersionsByFunction",
            "lambda:GetFunctionConfiguration",
            "lambda:PutFunctionConcurrency",
            "lambda:UpdateFunctionConfiguration",
            "lambda:UpdateFunctionCode",
            "lambda:DeleteFunctionConcurrency",
            "lambda:RemovePermission",
            "lambda:AddPermission",
            "lambda:ListTags",
            "lambda:GetAlias",
            "lambda:GetPolicy",
            "lambda:ListAliases",
```

```
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "secretsmanager:ListSecrets",
        "glue:GetCatalogs"
      ],
      "Resource": "*"
    }
  ]
}
```

Allow access to Athena Federated Query: Example policies

The permission policy examples in this topic demonstrate required allowed actions and the resources for which they are allowed. Examine these policies carefully and modify them according to your requirements before attaching them to IAM identities.

For information about attaching policies to IAM identities, see Adding and removing IAM identity permissions in the IAM User Guide.

- Example policy to allow an IAM principal to run and return results using Athena Federated Query
- Example Policy to Allow an IAM Principal to Create a Data Source Connector

Example – Allow an IAM principal to run and return results using Athena Federated Query

The following identity-based permissions policy allows actions that a user or other IAM principal requires to use Athena Federated Query. Principals who are allowed to perform these actions are able to run queries that specify Athena catalogs associated with a federated data source.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Athena",
            "Effect": "Allow",
            "Action": [
                "athena:GetDataCatalog",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "athena:GetWorkGroup",
                "athena:StartQueryExecution",
```

```
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkgroupName",
                "arn:aws:athena:aws_region:111122223333:datacatalog/DataCatalogName"
            ]
        },
        {
            "Sid": "ListAthenaWorkGroups",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "Lambda",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListMultipartUploadParts",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::MyLambdaSpillBucket",
                "arn:aws:s3:::MyLambdaSpillBucket/*",
                "arn:aws:s3:::MyQueryResultsBucket",
                "arn:aws:s3:::MyQueryResultsBucket/*"
            ]
        }
    ]
}
```

Explanation of permissions

Allowed actions	Explanation
`"athena:GetQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	Athena permissions that are required to run federated queries.
`"athena:GetDataCatalog", "athena:GetQueryExecution," "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	Athena permissions that are required to run federated view queries. The `GetDataCatalog` action is required for views.
`"lambda:InvokeFunction"`	Allows queries to invoke the AWS Lambda functions for the AWS Lambda functions specified in the `Resource` block. For example, `arn:aws:lambda:*:` *`MyAWSAcctId`* `:function:` *`MyAthenaLambdaFunction`* , where *`MyAthenaLambdaFunction`* specifies the name of a Lambda function to be invoked. As shown in the example, multiple functions can be specified.
`"s3:AbortMultipartUpload", "s3:GetBucketLocation", "s3:GetObject", "s3:ListBucket", "s3:ListMultipartUploadParts", "s3:PutObject"`	The `s3:ListBucket` and `s3:GetBucketLocation` permissions are required to access the query output bucket for IAM principals that run `StartQueryExecution` . `s3:PutObject` , `s3:ListMultipartUploadParts` , and `s3:AbortMultipartUpload` allow writing query results to all sub-folders of the query results bucket as specified by the `arn:aws:s3:::` *`MyQueryRe`*

Allowed actions	Explanation
	sultsBucket /* resource identifier, where *MyQueryResultsBucket* is the Athena query results bucket. For more information, see [Work with query results and recent queries](). `s3:GetObject` allows reading of query results and query history for the resource specified as `arn:aws:s3:::` *MyQueryResultsBucket*, where *MyQueryResultsBucket* is the Athena query results bucket. `s3:GetObject` also allows reading from the resource specified as `"arn:aws:s3:::` *MyLambdaSpillBucket* /*MyLambdaSpillPrefix* `*"`, where *MyLambdaSpillPrefix* is specified in the configuration of the Lambda function or functions being invoked.

Example – Allow an IAM principal to create a data source connector

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
```

```
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

 "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateUploadBucket",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
```

```
                "ec2:DescribeSecurityGroups",
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
        },
        {
            "Sid": "ECR",
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "arn:aws:ecr:*:*:repository/*"
        }
    ]
```

```
}
```

Explanation of permissions

Allowed actions	Explanation
`"lambda:CreateFunction",` `"lambda:ListVersionsByFunction",` `"lambda:GetFunctionConfiguration",` `"lambda:PutFunctionConcurrency",` `"lambda:ListTags",` `"lambda:DeleteFunction",` `"lambda:GetAlias",` `"lambda:InvokeFunction",` `"lambda:GetFunction",` `"lambda:ListAliases",` `"lambda:UpdateFunctionConfiguration",` `"lambda:UpdateFunctionCode",` `"lambda:AddPermission",` `"lambda:DeleteFunctionConcurrency",` `"lambda:RemovePermission",` `"lambda:GetPolicy"` `"lambda:GetAccountSettings",` `"lambda:ListFunctions",` `"lambda:ListEventSourceMappings",`	Allow the creation and management of Lambda functions listed as resources. In the example, a name prefix is used in the resource identifier `arn:aws:lambda:*:` *`MyAWSAcctId`* `:function:` *`MyAthenaLambdaFunctionsPrefix`* `*`, where *`MyAthenaLambdaFunctionsPrefix`* is a shared prefix used in the name of a group of Lambda functions so that they don't need to be specified individually as resources. You can specify one or more Lambda function resources.
`"s3:GetObject"`	Allows reading of a bucket that AWS Serverless Application Repository requires as specified by the resource identifier `arn:aws:s3:::awsserverlessrepo-changesets-` *`1iiv3xa62ln3m`* `/*`. This bucket may be specific to your account.
`"cloudformation:*"`	Allows the creation and management of AWS CloudFormation stacks specified by the resource *`MyCFStackPrefix`* . These stacks and stacksets are how AWS Serverless Application Repository deploys connectors and UDFs.

Allowed actions	Explanation
`"serverlessrepo:*"`	Allows searching, viewing, publishing, and updating applications in the AWS Serverles s Application Repository, specified by the resource identifier `arn:aws:serverless repo:*:*:applications/*` .
`"ecr:BatchGetImage", "ecr:GetDownloadUrlForLayer"`	Allows the created Lambda function to access the federation connector ECR image.

Allow access to Athena UDFs: Example policies

The permission policy examples in this topic demonstrate required allowed actions and the resources for which they are allowed. Examine these policies carefully and modify them according to your requirements before you attach similar permissions policies to IAM identities.

- Example Policy to Allow an IAM Principal to Run and Return Queries that Contain an Athena UDF Statement
- Example Policy to Allow an IAM Principal to Create an Athena UDF

Example – Allow an IAM principal to run and return queries that contain an Athena UDF statement

The following identity-based permissions policy allows actions that a user or other IAM principal requires to run queries that use Athena UDF statements.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "lambda:InvokeFunction",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts",
```

```
                "athena:GetWorkGroup",
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:athena:*:MyAWSAcctId:workgroup/MyAthenaWorkGroup",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:lambda:*:MyAWSAcctId:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:MyAWSAcctId:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        }
    ]
}
```

Explanation of permissions

Allowed actions	Explanation
`"athena:StartQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StopQueryExecution", "athena:GetQueryExecution",`	Athena permissions that are required to run queries in the `MyAthenaWorkGroup` work group.
`"s3:PutObject", "s3:GetObject", "s3:AbortMultipartUpload"`	`s3:PutObject` and `s3:AbortMultipartUpload` allow writing query results to all sub-folders of the query results bucket as specified by the `arn:aws:s3:::MyQueryResultsBucket/*` resource identifier, where *`MyQueryResultsBucket`* is the Athena query results

Allowed actions	Explanation
	bucket. For more information, see [Work with query results and recent queries](). `s3:GetObject` allows reading of query results and query history for the resource specified as `arn:aws:s3::: MyQueryResultsBucket` , where `MyQueryResultsBucket` is the Athena query results bucket. For more information, see [Work with query results and recent queries](). `s3:GetObject` also allows reading from the resource specified as `"arn:aws:s3::: MyLambdaSpillBucket /MyLambdaSpillPrefix *"`, where `MyLambdaSpillPrefix` is specified in the configuration of the Lambda function or functions being invoked.
`"lambda:InvokeFunction"`	Allows queries to invoke the AWS Lambda functions specified in the `Resource` block. For example, `arn:aws:lambda:*: MyAWSAcctId :function: MyAthenaLambdaFunction` , where `MyAthenaLambdaFunction` specifies the name of a Lambda function to be invoked. Multiple functions can be specified as shown in the example.

Example – Allow an IAM principal to create an Athena UDF

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

 "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
```

```
                "cloudformation:CreateUploadBucket",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
                "ec2:DescribeSecurityGroups",
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
        },
        {
            "Sid": "ECR",
            "Effect": "Allow",
```

```
                "Action": [
                    "ecr:BatchGetImage",
                    "ecr:GetDownloadUrlForLayer"
                ],
                "Resource": "arn:aws:ecr:*:*:repository/*"
            }
        ]
}
```

Explanation of permissions

Allowed actions	Explanation
`"lambda:CreateFunction", "lambda:ListVersionsByFunction", "lambda:GetFunctionConfiguration", "lambda:PutFunctionConcurrency", "lambda:ListTags", "lambda:DeleteFunction", "lambda:GetAlias", "lambda:InvokeFunction", "lambda:GetFunction", "lambda:ListAliases", "lambda:UpdateFunctionConfiguration", "lambda:UpdateFunctionCode", "lambda:AddPermission", "lambda:DeleteFunctionConcurrency", "lambda:RemovePermission", "lambda:GetPolicy" "lambda:GetAccountSettings", "lambda:ListFunctions", "lambda:ListEventSourceMappings",`	Allow the creation and management of Lambda functions listed as resources. In the example, a name prefix is used in the resource identifier `arn:aws:lambda:*:` *`MyAWSAcctId`* `:function:` *`MyAthenaLambdaFunctionsPrefix`* `*`, where *`MyAthenaLambdaFunctionsPrefix`* is a shared prefix used in the name of a group of Lambda functions so that they don't need to be specified individually as resources. You can specify one or more Lambda function resources.
`"s3:GetObject"`	Allows reading of a bucket that AWS Serverless Application Repository requires as specified by the resource identifier `arn:aws:s3:::awsserverlessrepo-changesets-` *`1iiv3xa62ln3m`* `/*`.

Allowed actions	Explanation
`"cloudformation:*"`	Allows the creation and management of AWS CloudFormation stacks specified by the resource *MyCFStackPrefix* . These stacks and stacksets are how AWS Serverless Application Repository deploys connectors and UDFs.
`"serverlessrepo:*"`	Allows searching, viewing, publishing, and updating applications in the AWS Serverless Application Repository, specified by the resource identifier `arn:aws:serverlessrepo:*:*:applications/*` .

Allow access for ML with Athena

IAM principals who run Athena ML queries must be allowed to perform the `sagemaker:invokeEndpoint` action for Sagemaker endpoints that they use. Include a policy statement similar to the following in identity-based permissions policies attached to user identities. In addition, attach the AWS managed policy: AmazonAthenaFullAccess, which grants full access to Athena actions, or a modified inline policy that allows a subset of actions.

Replace `arn:aws:sagemaker:region:AWSAcctID:ModelEndpoint` in the example with the ARN or ARNs of model endpoints to be used in queries. For more information, see Actions, resources, and condition keys for SageMaker AI in the *Service Authorization Reference*.

```
{
            "Effect": "Allow",
            "Action": [
                "sagemaker:invokeEndpoint"
            ],
            "Resource": "arn:aws:sagemaker:us-west-2:123456789012:workteam/public-crowd/default"
}
```

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Enable federated access to the Athena API

This section discusses federated access that allows a user or client application in your organization to call Amazon Athena API operations. In this case, your organization's users don't have direct access to Athena. Instead, you manage user credentials outside of AWS in Microsoft Active Directory. Active Directory supports SAML 2.0 (Security Assertion Markup Language 2.0).

To authenticate users in this scenario, use the JDBC or ODBC driver with SAML.2.0 support to access Active Directory Federation Services (ADFS) 3.0 and enable a client application to call Athena API operations.

For more information about SAML 2.0 support on AWS, see About SAML 2.0 federation in the *IAM User Guide*.

> **Note**
>
> Federated access to the Athena API is supported for a particular type of identity provider (IdP), the Active Directory Federation Service (ADFS 3.0), which is part of Windows Server. Federated access is not compatible with the IAM Identity Center trusted identity propagation feature. Access is established through the versions of JDBC or ODBC drivers that support SAML 2.0. For information, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

Topics

- Before you begin
- Understand the authentication process
- Procedure: Enable SAML-based federated access to the Athena API

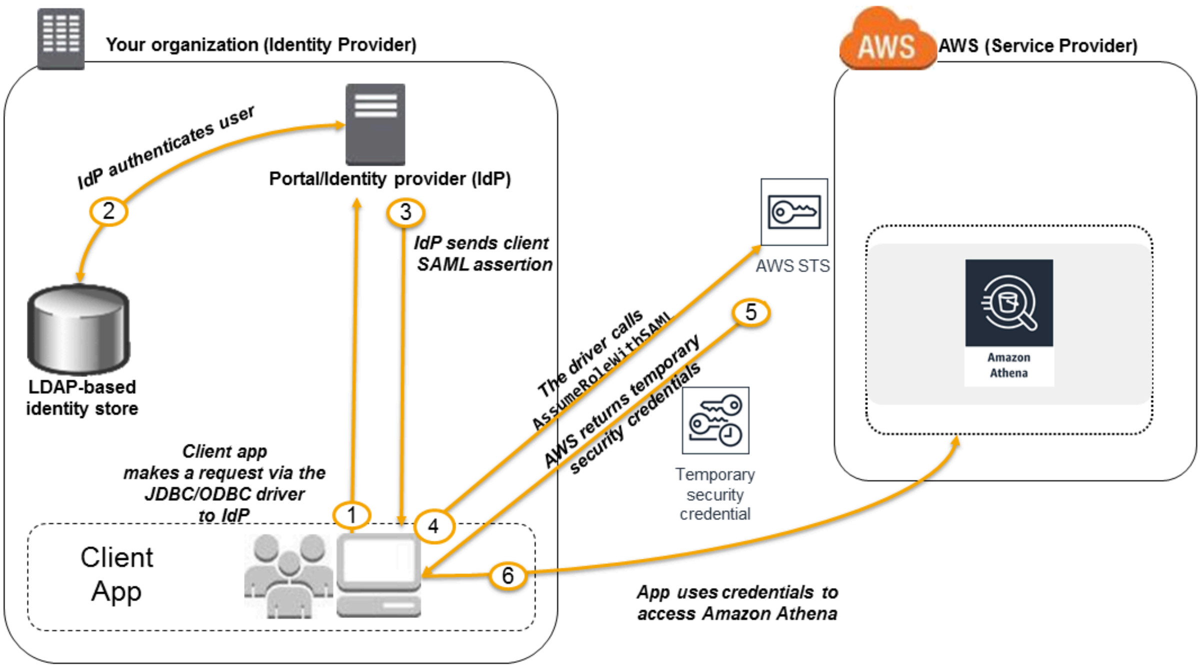
Before you begin

Before you begin, complete the following prerequisites:

- Inside your organization, install and configure the ADFS 3.0 as your IdP.
- Install and configure the latest available versions of JDBC or ODBC drivers on clients that are used to access Athena. The driver must include support for federated access compatible with SAML 2.0. For information, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

Understand the authentication process

The following diagram illustrates the authentication process of federated access to the Athena API.



1. A user in your organization uses a client application with the JDBC or ODBC driver to request authentication from your organization's IdP. The IdP is ADFS 3.0.
2. The IdP authenticates the user against Active Directory, which is your organization's Identity Store.
3. The IdP constructs a SAML assertion with information about the user and sends the assertion to the client application via the JDBC or ODBC driver.
4. The JDBC or ODBC driver calls the AWS Security Token Service AssumeRoleWithSAML API operation, passing it the following parameters:
   - The ARN of the SAML provider
   - The ARN of the role to assume
   - The SAML assertion from the IdP

   For more information, see AssumeRoleWithSAML, in the *AWS Security Token Service API Reference*.

5. The API response to the client application via the JDBC or ODBC driver includes temporary security credentials.
6. The client application uses the temporary security credentials to call Athena API operations, allowing your users to access Athena API operations.

Procedure: Enable SAML-based federated access to the Athena API

This procedure establishes trust between your organization's IdP and your AWS account to enable SAML-based federated access to the Amazon Athena API operation.

To enable federated access to the Athena API:

1. In your organization, register AWS as a service provider (SP) in your IdP. This process is known as *relying party trust*. For more information, see Configuring your SAML 2.0 IdP with relying party trust in the *IAM User Guide*. As part of this task, perform these steps:

   a. Obtain the sample SAML metadata document from this URL: https://signin.aws.amazon.com/static/saml-metadata.xml.

   b. In your organization's IdP (ADFS), generate an equivalent metadata XML file that describes your IdP as an identity provider to AWS. Your metadata file must include the issuer name, creation date, expiration date, and keys that AWS uses to validate authentication responses (assertions) from your organization.

2. In the IAM console, create a SAML identity provider entity. For more information, see Creating SAML identity providers in the *IAM User Guide*. As part of this step, do the following:

   a. Open the IAM console at https://console.aws.amazon.com/iam/.

   b. Upload the SAML metadata document produced by the IdP (ADFS) in Step 1 in this procedure.

3. In the IAM console, create one or more IAM roles for your IdP. For more information, see Creating a role for a third-party Identity Provider (federation) in the *IAM User Guide*. As part of this step, do the following:

   - In the role's permission policy, list actions that users from your organization are allowed to do in AWS.
   - In the role's trust policy, set the SAML provider entity that you created in Step 2 of this procedure as the principal.

This establishes a trust relationship between your organization and AWS.

4. In your organization's IdP (ADFS), define assertions that map users or groups in your organization to the IAM roles. The mapping of users and groups to the IAM roles is also known as a *claim rule*. Note that different users and groups in your organization might map to different IAM roles.

   For information about configuring the mapping in ADFS, see the blog post: [Enabling federation to AWS using Windows Active Directory, ADFS, and SAML 2.0](#).

5. Install and configure the JDBC or ODBC driver with SAML 2.0 support. For information, see [Connect to Amazon Athena with JDBC](#) and [Connect to Amazon Athena with ODBC](#).

6. Specify the connection string from your application to the JDBC or ODBC driver. For information about the connection string that your application should use, see the topic *"Using the Active Directory Federation Services (ADFS) Credentials Provider"* in the *JDBC Driver Installation and Configuration Guide*, or a similar topic in the *ODBC Driver Installation and Configuration Guide* available as PDF downloads from the [Connect to Amazon Athena with JDBC](#) and [Connect to Amazon Athena with ODBC](#) topics.

   Following is a high-level summary of configuring the connection string to the drivers:

   1. In the `AwsCredentialsProviderClass configuration`, set the `com.simba.athena.iamsupport.plugin.AdfsCredentialsProvider` to indicate that you want to use SAML 2.0 based authentication via ADFS IdP.
   2. For `idp_host`, provide the host name of the ADFS IdP server.
   3. For `idp_port`, provide the port number that the ADFS IdP listens on for the SAML assertion request.
   4. For `UID` and `PWD`, provide the AD domain user credentials. When using the driver on Windows, if `UID` and `PWD` are not provided, the driver attempts to obtain the user credentials of the user logged in to the Windows machine.
   5. Optionally, set `ssl_insecure` to `true`. In this case, the driver does not check the authenticity of the SSL certificate for the ADFS IdP server. Setting to `true` is needed if the ADFS IdP's SSL certificate has not been configured to be trusted by the driver.
   6. To enable mapping of an Active Directory domain user or group to one or more IAM roles (as mentioned in step 4 of this procedure), in the `preferred_role` for the JDBC or ODBC connection, specify the IAM role (ARN) to assume for the driver connection. Specifying the

`preferred_role` is optional, and is useful if the role is not the first role listed in the claim rule.

As a result of this procedure, the following actions occur:

1. The JDBC or ODBC driver calls the AWS STS AssumeRoleWithSAML API, and passes it the assertions, as shown in step 4 of the architecture diagram.
2. AWS makes sure that the request to assume the role comes from the IdP referenced in the SAML provider entity.
3. If the request is successful, the AWS STS AssumeRoleWithSAML API operation returns a set of temporary security credentials, which your client application uses to make signed requests to Athena.

   Your application now has information about the current user and can access Athena programmatically.

Log and monitor Athena

To detect incidents, receive alerts when incidents occur, and respond to them, use these options with Amazon Athena:

- **Monitor Athena with AWS CloudTrail** – AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Athena. It captures calls from the Athena console and code calls to the Athena API operations as events. This allow you to determine the request that was made to Athena, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Log Amazon Athena API calls with AWS CloudTrail.

  You can also use Athena to query the CloudTrail log files not only for Athena, but for other AWS services. For more information, see Query AWS CloudTrail logs.
- **Monitor Athena usage with CloudTrail and Amazon QuickSight** – Amazon QuickSight is a fully managed, cloud-powered business intelligence service that lets you create interactive dashboards your organization can access from any device. For an example of a solution that uses CloudTrail and Amazon QuickSight to monitor Athena usage, see the AWS Big Data blog post How Realtor.com monitors Amazon Athena usage with AWS CloudTrail and QuickSight.

- **Use EventBridge with Athena –** Amazon EventBridge delivers a near real-time stream of system events that describe changes in AWS resources. EventBridge becomes aware of operational changes as they occur, responds to them, and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information. Events are emitted on a best effort basis. For more information, see Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.
- **Use workgroups to separate users, teams, applications, or workloads, and to set query limits and control query costs –** You can view query-related metrics in Amazon CloudWatch, control query costs by configuring limits on the amount of data scanned, create thresholds, and trigger actions, such as Amazon SNS alarms, when these thresholds are breached. For more information, see Use workgroups to control query access and costs. Use resource-level IAM permissions to control access to a specific workgroup. For more information, see Use IAM policies to control workgroup access and Use CloudWatch and EventBridge to monitor queries and control costs.

Topics

Log Amazon Athena API calls with AWS CloudTrail

Athena is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Athena.

CloudTrail captures all API calls for Athena as events. The calls captured include calls from the Athena console and code calls to the Athena API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Athena. 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 Athena, 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.

You can use Athena to query CloudTrail log files from Athena itself and from other AWS services. For more information, see Query AWS CloudTrail logs, Hive JSON SerDe, and the AWS Big Data Blog post Use CTAS statements with Amazon Athena to reduce cost and improve performance, which uses CloudTrail to provide insight into Athena usage.

About Athena information in CloudTrail

CloudTrail is enabled on your Amazon Web Services account when you create the account. When activity occurs in Athena, 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 Amazon Web Services account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your Amazon Web Services account, including events for Athena, 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 Athena actions are logged by CloudTrail and are documented in the Amazon Athena API Reference. For example, calls to the StartQueryExecution and GetQueryResults 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 root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understand Athena 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.

> **Note**
>
> To prevent unintended disclosure of sensitive information, the `queryString` entry in both the `StartQueryExecution` and `CreateNamedQuery` logs has a value of `***OMITTED***`. This is by design. To access the actual query string, you can use the Athena [GetQueryExecution](#) API and pass in the value of `responseElements.queryExecutionId` from the CloudTrail log.

The following examples demonstrate CloudTrail log entries for:

- [StartQueryExecution (Successful)](#)
- [StartQueryExecution (Failed)](#)
- [CreateNamedQuery](#)

StartQueryExecution (successful)

```
{
 "eventVersion":"1.05",
 "userIdentity":{
    "type":"IAMUser",
    "principalId":"EXAMPLE_PRINCIPAL_ID",
    "arn":"arn:aws:iam::123456789012:user/johndoe",
    "accountId":"123456789012",
    "accessKeyId":"EXAMPLE_KEY_ID",
    "userName":"johndoe"
 },
 "eventTime":"2017-05-04T00:23:55Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "requestParameters":{
    "clientRequestToken":"16bc6e70-f972-4260-b18a-db1b623cb35c",
    "resultConfiguration":{
```

```
        "outputLocation":"s3://amzn-s3-demo-bucket/test/"
     },
     "queryString":"***OMITTED***"
  },
  "responseElements":{
     "queryExecutionId":"b621c254-74e0-48e3-9630-78ed857782f9"
  },
  "requestID":"f5039b01-305f-11e7-b146-c3fc56a7dc7a",
  "eventID":"c97cf8c8-6112-467a-8777-53bb38f83fd5",
  "eventType":"AwsApiCall",
  "recipientAccountId":"123456789012"
}
```


StartQueryExecution (failed)


```
{
 "eventVersion":"1.05",
 "userIdentity":{
  "type":"IAMUser",
  "principalId":"EXAMPLE_PRINCIPAL_ID",
  "arn":"arn:aws:iam::123456789012:user/johndoe",
  "accountId":"123456789012",
  "accessKeyId":"EXAMPLE_KEY_ID",
  "userName":"johndoe"
  },
 "eventTime":"2017-05-04T00:21:57Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "errorCode":"InvalidRequestException",
 "errorMessage":"Invalid result configuration. Should specify either output location or
 result configuration",
 "requestParameters":{
  "clientRequestToken":"ca0e965f-d6d8-4277-8257-814a57f57446",
  "queryString":"***OMITTED***"
  },
 "responseElements":null,
 "requestID":"aefbc057-305f-11e7-9f39-bbc56d5d161e",
 "eventID":"6e1fc69b-d076-477e-8dec-024ee51488c4",
 "eventType":"AwsApiCall",
 "recipientAccountId":"123456789012"
```

```
}
```

CreateNamedQuery

```
{
  "eventVersion":"1.05",
  "userIdentity":{
     "type":"IAMUser",
     "principalId":"EXAMPLE_PRINCIPAL_ID",
     "arn":"arn:aws:iam::123456789012:user/johndoe",
     "accountId":"123456789012",
     "accessKeyId":"EXAMPLE_KEY_ID",
     "userName":"johndoe"
  },
  "eventTime":"2017-05-16T22:00:58Z",
  "eventSource":"athena.amazonaws.com",
  "eventName":"CreateNamedQuery",
  "awsRegion":"us-west-2",
  "sourceIPAddress":"77.88.999.69",
  "userAgent":"aws-cli/1.11.85 Python/2.7.10 Darwin/16.6.0 botocore/1.5.48",
  "requestParameters":{
     "name":"johndoetest",
     "queryString":"***OMITTED***",
     "database":"default",
     "clientRequestToken":"fc1ad880-69ee-4df0-bb0f-1770d9a539b1"
     },
  "responseElements":{
     "namedQueryId":"cdd0fe29-4787-4263-9188-a9c8db29f2d6"
     },
  "requestID":"2487dd96-3a83-11e7-8f67-c9de5ac76512",
  "eventID":"15e3d3b5-6c3b-4c7c-bc0b-36a8dd95227b",
  "eventType":"AwsApiCall",
  "recipientAccountId":"123456789012"
},
```

Compliance validation for Athena

Third-party auditors assess the security and compliance of Amazon Athena as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, and others.

For a list of AWS services in scope of specific compliance programs, see AWS services in scope by compliance program. For general information, see AWS compliance programs.

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

Your compliance responsibility when using Athena 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 on Amazon Web Services – 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.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Athena

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 Availability 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, Athena offers several features to help support your data resiliency and backup needs.

Athena is serverless, so there is no infrastructure to set up or manage. Athena is highly available and runs queries using compute resources across multiple Availability Zones, automatically routing queries appropriately if a particular Availability Zone is unreachable. Athena uses Amazon S3 as its underlying data store, making your data highly available and durable. Amazon S3 provides durable infrastructure to store important data and is designed for durability of 99.999999999% of objects. Your data is redundantly stored across multiple facilities and multiple devices in each facility.

Infrastructure security in Athena

As a managed service, Amazon Athena is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access Athena through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

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

Use IAM policies to restrict access to Athena operations. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

Athena managed policies are easy to use, and are automatically updated with the required actions as the service evolves. Customer-managed and inline policies allow you to fine tune policies by specifying more granular Athena actions within the policy. Grant appropriate access to the Amazon S3 location of the data. For detailed information and scenarios about how to grant Amazon S3 access, see Example walkthroughs: Managing access in the *Amazon Simple Storage Service Developer Guide*. For more information and an example of which Amazon S3 actions to allow, see the example bucket policy in Cross-Account Access.

Topics

Connect to Amazon Athena using an interface VPC endpoint

You can improve the security posture of your VPC by using an interface VPC endpoint (AWS PrivateLink) and an AWS Glue VPC endpoint in your Virtual Private Cloud (VPC). An interface VPC endpoint improves security by giving you control over what destinations can be reached from inside your VPC. Each VPC endpoint is represented by one or more Elastic network interfaces (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to Athena without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Athena API.

To use Athena through your VPC, you must connect from an instance that is inside the VPC or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN connections in the *Amazon Virtual Private Cloud User Guide*. For information about AWS Direct Connect, see Creating a connection in the *AWS Direct Connect User Guide*.

Athena supports VPC endpoints in all AWS Regions where both Amazon VPC and Athena are available.

You can create an interface VPC endpoint to connect to Athena using the AWS Management Console or AWS Command Line Interface (AWS CLI) commands. For more information, see Creating an interface endpoint.

After you create an interface VPC endpoint, if you enable private DNS hostnames for the endpoint, the default Athena endpoint (https://athena.*Region*.amazonaws.com) resolves to your VPC endpoint.

If you do not enable private DNS hostnames, Amazon VPC provides a DNS endpoint name that you can use in the following format:

```
VPC_Endpoint_ID.athena.Region.vpce.amazonaws.com
```

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

Athena supports making calls to all of its API actions inside your VPC.

Create a VPC endpoint policy for Athena

You can create a policy for Amazon VPC endpoints for Athena to specify restrictions like the following:

- **Principal** – The principal that can perform actions.
- **Actions** – The actions that can be performed.
- **Resources** – The resources on which actions can be performed.
- **Only trusted identities** – Use the `aws:PrincipalOrgId` condition to restrict access to only credentials that are part of your AWS organization. This can help prevent access by unintended principals.
- **Only trusted resources** – Use the `aws:ResourceOrgId` condition to prevent access to unintended resources.
- **Only trusted identities and resources** – Create a combined policy for a VPC endpoint that helps prevent access to unintended principals and resources.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide* and [Appendix 2 – VPC endpoint policy examples](#) in the AWS Whitepaper *Building a data perimeter on AWS*.

Example – VPC endpoint policy

The following example allows requests by organization identities to organization resources and allows requests by AWS service principals.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "my-org-id",
```

```
                    "aws:ResourceOrgID": "my-org-id"
                }
            }
        },
        {
            "Sid": "AllowRequestsByAWSServicePrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "aws:PrincipalIsAWSService": "true"
                }
            }
        }
    ]
}
```

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

About VPC endpoints in shared subnets

You can't create, describe, modify, or delete VPC endpoints in subnets that are shared with you. However, you can use the VPC endpoints in subnets that are shared with you. For information about VPC sharing, see Share your VPC with other accounts in the *Amazon VPC User Guide*.

Configuration and vulnerability analysis in Athena

Athena is serverless, so there is no infrastructure to set up or manage. AWS handles basic security tasks, such as guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following AWSresources:

- Shared responsibility model
- Best practices for security, identity, & compliance

Use Athena to query data registered with AWS Lake Formation

AWS Lake Formation allows you to define and enforce database, table, and column-level access policies when using Athena queries to read data stored in Amazon S3 or accessed through federated data sources. Lake Formation provides an authorization and governance layer on data stored in Amazon S3 or federated data catalogs. You can use a hierarchy of permissions in Lake Formation to grant or revoke permissions to read data catalog objects such as databases, tables, and columns. Lake Formation simplifies the management of permissions and allows you to implement fine-grained access control (FGAC) for your data.

You can use Athena to query both data that is registered with Lake Formation and data that is not registered with Lake Formation.

Lake Formation permissions apply when using Athena to query source data from Amazon S3 locations or data catalogs that are registered with Lake Formation. Lake Formation permissions also apply when you create databases and tables that point to registered Amazon S3 data locations or data catalogs.

Lake Formation permissions do not apply when writing objects, nor do they apply when querying data or metadata that are not registered with Lake Formation. For source data and metadata that are not registered with Lake Formation, access is determined by IAM permissions policies and AWS Glue actions. Athena query results locations in Amazon S3 cannot be registered with Lake Formation, and IAM permissions policies for Amazon S3 control access. In addition, Lake Formation permissions do not apply to Athena query history. You can use Athena workgroups to control access to query history.

For more information about Lake Formation, see Lake Formation FAQs and the AWS Lake Formation Developer Guide.

Apply Lake Formation permissions to existing databases and tables

If you are new to Athena and you use Lake Formation to configure access to query data, you do not need to configure IAM policies so that users can read data and create metadata. You can use Lake Formation to administer permissions.

Registering data with Lake Formation and updating IAM permissions policies is not a requirement. If data is not registered with Lake Formation, Athena users who have appropriate permissions can continue to query data not registered with Lake Formation.

If you have existing Athena users who query Amazon S3 data not registered with Lake Formation, you can update IAM permissions for Amazon S3—and the AWS Glue Data Catalog, if applicable—so that you can use Lake Formation permissions to manage user access centrally. For permission to read Amazon S3 data locations, you can update resource-based and identity-based policies to modify Amazon S3 permissions. For access to metadata, if you configured resource-level policies for fine-grained access control with AWS Glue, you can use Lake Formation permissions to manage access instead.

For more information, see Configure access to databases and tables in the AWS Glue Data Catalog and Upgrading AWS Glue data permissions to the AWS Lake Formation model in the *AWS Lake Formation Developer Guide*.

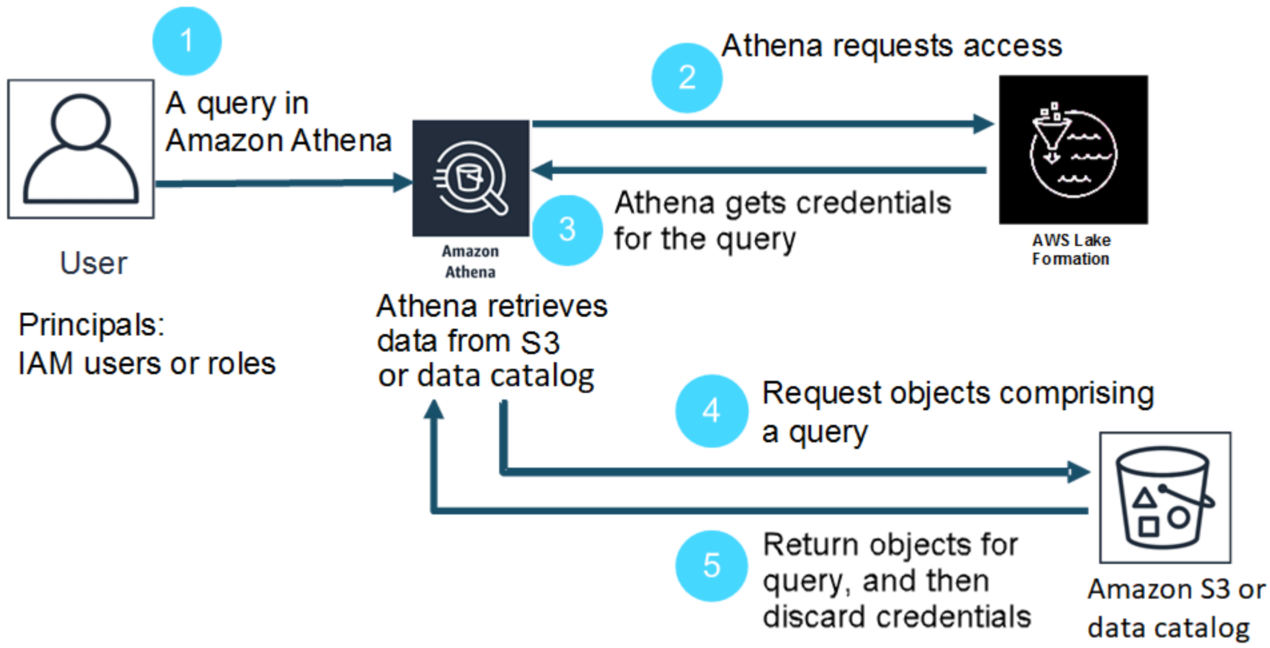
Topics

How Athena accesses data registered with Lake Formation

The access workflow described in this section applies when you run Athena queries on Amazon S3 locations, data catalogs, or metadata objects that are registered with Lake Formation. For more information, see Registering a data lake in the *AWS Lake Formation Developer Guide*. In addition to registering data, the Lake Formation administrator applies Lake Formation permissions that grant or revoke access to metadata in the data catalog, AWS Glue Data Catalog, or the data location in Amazon S3. For more information, see Security and access control to metadata and data in the *AWS Lake Formation Developer Guide*.

Each time an Athena principal (user, group, or role) runs a query on data registered using Lake Formation, Lake Formation verifies that the principal has the appropriate Lake Formation permissions to the database, table, and data source location as appropriate for the query. If the principal has access, Lake Formation *vends* temporary credentials to Athena, and the query runs.

The following diagram shows how credential vending works in Athena on a query-by-query basis for a hypothetical `SELECT` query on a table with an Amazon S3 location or data catalog registered in Lake Formation:



1. A principal runs a SELECT query in Athena.
2. Athena analyzes the query and checks Lake Formation permissions to see if the principal has been granted access to the table and table columns.
3. If the principal has access, Athena requests credentials from Lake Formation. If the principal *does not* have access, Athena issues an access denied error.
4. Lake Formation issues credentials to Athena to use when reading data from Amazon S3 or catalog, along with the list of allowed columns.
5. Athena uses the Lake Formation temporary credentials to query the data from Amazon S3 or catalog. After the query completes, Athena discards the credentials.

Considerations and limitations for querying data registered with Lake Formation

Consider the following when using Athena to query data registered in Lake Formation. For additional information, see Known issues for AWS Lake Formation in the *AWS Lake Formation Developer Guide*.

Considerations and Limitations

- Column metadata visible to unauthorized users in some circumstances with Avro and custom SerDe
- Understand Lake Formation and views

- Iceberg DDL support
- Lake Formation fine-grained access control and Athena workgroups
- Athena query results location in Amazon S3 not registered with Lake Formation
- Use Athena workgroups to limit access to query history
- Query CSE_KMS encrypted tables registered with Lake Formation
- Partitioned data locations registered with Lake Formation must be in table subdirectories
- Create table as select (CTAS) queries require Amazon S3 write permissions
- The DESCRIBE permission is required on the default database

Column metadata visible to unauthorized users in some circumstances with Avro and custom SerDe

Lake Formation column-level authorization prevents users from accessing data in columns for which the user does not have Lake Formation permissions. However, in certain situations, users are able to access metadata describing all columns in the table, including the columns for which they do not have permissions to the data.

This occurs when column metadata is stored in table properties for tables using either the Apache Avro storage format or using a custom Serializer/Deserializer (SerDe) in which table schema is defined in table properties along with the SerDe definition. When using Athena with Lake Formation, we recommend that you review the contents of table properties that you register with Lake Formation and, where possible, limit the information stored in table properties to prevent any sensitive metadata from being visible to users.

Understand Lake Formation and views

For data registered with Lake Formation, an Athena user can create a `VIEW` only if they have Lake Formation permissions to the tables, columns, and source Amazon S3 data locations on which the `VIEW` is based. After a `VIEW` is created in Athena, Lake Formation permissions can be applied to the `VIEW`. Column-level permissions are not available for a `VIEW`. Users who have Lake Formation permissions to a `VIEW` but do not have permissions to the table and columns on which the view was based are not able to use the `VIEW` to query data. However, users with this mix of permissions are able to use statements like `DESCRIBE VIEW`, `SHOW CREATE VIEW`, and `SHOW COLUMNS` to see `VIEW` metadata. For this reason, be sure to align Lake Formation permissions for each `VIEW` with underlying table permissions. Cell filters defined on a table do not apply to a `VIEW` for that table. Resource link names must have the same name as the resource in the originating account. There are additional limitations when working with views in a cross-account setup. For more information about setting up permissions for shared views across accounts, see Configure cross-account Data Catalog access.

Iceberg DDL support

Athena does not currently support DDL operations on Iceberg tables whose location is registered with Lake Formation. Attempting to run a DDL query on one of these Iceberg tables can return an Amazon S3 access denied error or fail with a query timeout. DDL operations on Iceberg tables require the user to have direct Amazon S3 access to the Iceberg table location.

Lake Formation fine-grained access control and Athena workgroups

Users in the same Athena workgroup can see the data that Lake Formation fine-grained access control has configured to be accessible to the workgroup. For more information about using fine-grained access control in Lake Formation, see Manage fine-grained access control using AWS Lake Formation in the *AWS Big Data Blog*.

Athena query results location in Amazon S3 not registered with Lake Formation

The query results locations in Amazon S3 for Athena cannot be registered with Lake Formation. Lake Formation permissions don't limit access to these locations. Unless you limit access, Athena users can access query result files and metadata when they don't have Lake Formation permissions for the data. To avoid this, we recommend that you use workgroups to specify the location for query results and align workgroup membership with Lake Formation permissions. You can then use IAM permissions policies to limit access to query results locations. For more information about query results, see Work with query results and recent queries.

Use Athena workgroups to limit access to query history

Athena query history exposes a list of saved queries and complete query strings. Unless you use workgroups to separate access to query histories, Athena users who are not authorized to query data in Lake Formation are able to view query strings run on that data, including column names, selection criteria, and so on. We recommend that you use workgroups to separate query histories, and align Athena workgroup membership with Lake Formation permissions to limit access. For more information, see Use workgroups to control query access and costs.

Query CSE_KMS encrypted tables registered with Lake Formation

Open Table Format (OTF) tables such as Apache Iceberg that have the following characteristics cannot be queried with Athena:

- The tables are based on Amazon S3 data locations that are registered with Lake Formation.
- The objects in Amazon S3 are encrypted using client-side encryption (CSE).
- The encryption uses AWS KMS customer-managed keys (`CSE_KMS`).

To query non-OTF tables that are encrypted with a `CSE_KMS` key), add the following block to the policy of the AWS KMS key that you use for CSE encryption. *<KMS_KEY_ARN>* is the ARN of the AWS KMS key that encrypts the data. *<IAM-ROLE-ARN>* is the ARN of the IAM role that registers the Amazon S3 location in Lake Formation.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "*"
    },
    "Action": "kms:Decrypt",
    "Resource": "<KMS-KEY-ARN>",
    "Condition": {
        "ArnLike": {
            "aws:PrincipalArn": "<IAM-ROLE-ARN>"
        }
    }
}
```

Partitioned data locations registered with Lake Formation must be in table subdirectories

Partitioned tables registered with Lake Formation must have partitioned data in directories that are subdirectories of the table in Amazon S3. For example, a table with the location `s3://amzn-s3-demo-bucket/mytable` and partitions `s3://amzn-s3-demo-bucket/mytable/dt=2019-07-11`, `s3://amzn-s3-demo-bucket/mytable/dt=2019-07-12`, and so on can be registered with Lake Formation and queried using Athena. On the other hand, a table with the location `s3://amzn-s3-demo-bucket/mytable` and partitions located in `s3://amzn-s3-demo-bucket/dt=2019-07-11`, `s3://amzn-s3-demo-bucket/dt=2019-07-12`, and so on, cannot be registered with Lake Formation. Because such partitions are not subdirectories of `s3://amzn-s3-demo-bucket/mytable`, they also cannot be read from Athena.

Create table as select (CTAS) queries require Amazon S3 write permissions

Create Table As Statements (CTAS) require write access to the Amazon S3 location of tables. To run CTAS queries on data registered with Lake Formation, Athena users must have IAM permissions to write to the table Amazon S3 locations in addition to the appropriate Lake Formation permissions to read the data locations. For more information, see [Create a table from query results (CTAS)](#).

The DESCRIBE permission is required on the default database

The Lake Formation `DESCRIBE` permission is required on the `default` database so that Lake Formation can view it. The following example AWS CLI command grants the `DESCRIBE` permission on the `default` database to the user `datalake_user1` in AWS account `111122223333`.

```
aws lakeformation grant-permissions --principal
 DataLakePrincipalIdentifier=arn:aws:iam::111122223333:user/datalake_user1 --
permissions "DESCRIBE" --resource '{ "Database": {"Name":"default"}}
```

For more information, see [DESCRIBE](#) in the *AWS Lake Formation Developer Guide*.

Configure cross-account Data Catalog access

To access a data catalog in another account, you can use Athena's cross-account AWS Glue feature or set up cross-account access in Lake Formation.

Option A: Configure cross-account Data Catalog access in Athena

You can use Athena's cross-account AWS Glue catalog feature to register the catalog in your account. This capability is available only in Athena engine version 2 and later versions and is limited to same-Region use between accounts. For more information, see [Register a Data Catalog from another account](#).

If the Data Catalog to be shared has a resource policy configured in AWS Glue, it must be updated to allow access to the AWS Resource Access Manager and grant permissions to Account B to use Account A's Data Catalog, as in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "ram.amazonaws.com"
```

```
        },
        "Action": "glue:ShareResource",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<ACCOUNT-B>:root"
        },
        "Action": "glue:*",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    }
    ]
}
```

For more information, see Configure cross-account access to AWS Glue data catalogs.

Option B: Configure cross-account access in Lake Formation

AWS Lake Formation lets you use a single account to manage a central Data Catalog. You can use this feature to implement cross-account access to Data Catalog metadata and underlying data. For example, an owner account can grant another (recipient) account SELECT permission on a table.

For a shared database or table to appear in the Athena Query Editor, you create a resource link in Lake Formation to the shared database or table. When the recipient account in Lake Formation queries the owner's table, CloudTrail adds the data access event to the logs for both the recipient account and the owner account.

For shared views, keep in mind the following points:

- Queries are run on target resource links, not on the source table or view, and then the output is shared to the target account.
- It is not sufficient to share only the view. All the tables that are involved in creating the view must be part of the cross-account share.

- The name of the resource link created on the shared resources must match the name of the resource in the owner account. If the name does not match, an error message like Failed analyzing stored view 'awsdatacatalog.*my-lf-resource-link*.*my-lf-view*': line 3:3: Schema *schema_name* does not exist occurs.

For more information about cross-account access in Lake Formation, see the following resources in the *AWS Lake Formation Developer Guide*:

Cross-account access

How resource links work in Lake Formation

Cross-account CloudTrail logging

Manage Lake Formation and Athena user permissions

Lake Formation vends credentials to query Amazon S3 data stores or federated catalogs that are registered with Lake Formation. If you previously used IAM policies to allow or deny permissions to read catalogs or data locations in Amazon S3, you can use Lake Formation permissions instead. However, other IAM permissions are still required.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

The following sections summarize the permissions required to use Athena to query data registered in Lake Formation. For more information, see Security in AWS Lake Formation in the *AWS Lake Formation Developer Guide*.

Permissions Summary

- Identity-based permissions for Lake Formation and Athena
- Amazon S3 permissions for Athena query results locations
- Athena workgroup memberships to query history
- Lake Formation permissions to data
- IAM permissions to write to Amazon S3 locations
- Permissions to encrypted data, metadata, and Athena query results
- Resource-based permissions for Amazon S3 buckets in external accounts (optional)

Identity-based permissions for Lake Formation and Athena

Anyone using Athena to query data registered with Lake Formation must have an IAM permissions policy that allows the `lakeformation:GetDataAccess` action. The AWS managed policy: AmazonAthenaFullAccess allows this action. If you use inline policies, be sure to update permissions policies to allow this action.

In Lake Formation, a *data lake administrator* has permissions to create metadata objects such as databases and tables, grant Lake Formation permissions to other users, and register new Amazon S3 locations or data catalogs. To register new locations, permissions to the service-linked role for Lake Formation are required. For more information, see Create a data lake administrator and Service-linked role permissions for Lake Formation in the *AWS Lake Formation Developer Guide*.

A Lake Formation user can use Athena to query databases, tables, table columns, and underlying Amazon S3 data stores or catalogs based on Lake Formation permissions granted to them by data lake administrators. Users cannot create databases or tables, or register new Amazon S3 locations with Lake Formation. For more information, see Create a data lake user in the *AWS Lake Formation Developer Guide*.

In Athena, identity-based permissions policies, including those for Athena workgroups, still control access to Athena actions for Amazon Web Services account users. In addition, federated access might be provided through the SAML-based authentication available with Athena drivers. For more information, see Use workgroups to control query access and costs, Use IAM policies to control workgroup access, and Enable federated access to the Athena API.

For more information, see Granting Lake Formation permissions in the *AWS Lake Formation Developer Guide*.

Amazon S3 permissions for Athena query results locations

The query results locations in Amazon S3 for Athena cannot be registered with Lake Formation. Lake Formation permissions don't limit access to these locations. Unless you limit access, Athena users can access query result files and metadata when they don't have Lake Formation permissions for the data. To avoid this, we recommend that you use workgroups to specify the location for query results and align workgroup membership with Lake Formation permissions. You can then use IAM permissions policies to limit access to query results locations. For more information about query results, see Work with query results and recent queries.

Athena workgroup memberships to query history

Athena query history exposes a list of saved queries and complete query strings. Unless you use workgroups to separate access to query histories, Athena users who are not authorized to query data in Lake Formation are able to view query strings run on that data, including column names, selection criteria, and so on. We recommend that you use workgroups to separate query histories, and align Athena workgroup membership with Lake Formation permissions to limit access. For more information, see Use workgroups to control query access and costs.

Lake Formation permissions to data

In addition to the baseline permission to use Lake Formation, Athena users must have Lake Formation permissions to access resources that they query. These permissions are granted and managed by a Lake Formation administrator. For more information, see Security and access control to metadata and data in the *AWS Lake Formation Developer Guide*.

IAM permissions to write to Amazon S3 locations

Lake Formation permissions to Amazon S3 do not include the ability to write to Amazon S3. Create Table As Statements (CTAS) require write access to the Amazon S3 location of tables. To run CTAS queries on data registered with Lake Formation, Athena users must have IAM permissions to write to the table Amazon S3 locations in addition to the appropriate Lake Formation permissions to read the data locations. For more information, see Create a table from query results (CTAS).

Permissions to encrypted data, metadata, and Athena query results

Underlying source data in Amazon S3 and metadata in the catalog that is registered with Lake Formation can be encrypted. There is no change to the way that Athena handles encryption of query results when using Athena to query data registered with Lake Formation. For more information, see Encrypt Athena query results stored in Amazon S3.

- **Encrypting source data –** Encryption of Amazon S3 data locations source data is supported. Athena users who query encrypted Amazon S3 locations that are registered with Lake Formation need permissions to encrypt and decrypt data. For more information about requirements, see Supported Amazon S3 encryption options and Permissions to encrypted data in Amazon S3.
- **Encrypting metadata –** Encrypting metadata in the AWS Glue Data Catalog is supported. For principals using Athena, identity-based policies must allow the `"kms:GenerateDataKey"`, `"kms:Decrypt"`, and `"kms:Encrypt"` actions for the key used to encrypt metadata. For more information, see Encrypting your Data Catalog in the *AWS Glue Developer Guide* and Configure access from Athena to encrypted metadata in the AWS Glue Data Catalog.

Resource-based permissions for Amazon S3 buckets in external accounts (optional)

To query an Amazon S3 data location in a different account, a resource-based IAM policy (bucket policy) must allow access to the location. For more information, see Configure cross-account access in Athena to Amazon S3 buckets.

For information about accessing catalogs in another account, see Option A: Configure cross-account Data Catalog access in Athena.

Use Lake Formation and JDBC or ODBC drivers for federated access to Athena

The Athena JDBC and ODBC drivers support SAML 2.0-based federation with Athena using Okta and Microsoft Active Directory Federation Services (AD FS) identity providers. By integrating Amazon Athena with AWS Lake Formation, you enable SAML-based authentication to Athena with corporate credentials. With Lake Formation and AWS Identity and Access Management (IAM), you can maintain fine-grained, column-level access control over the data available to the SAML user. With the Athena JDBC and ODBC drivers, federated access is available for tool or programmatic access.

To use Athena to access a data source controlled by Lake Formation, you need to enable SAML 2.0-based federation by configuring your identity provider (IdP) and AWS Identity and Access Management (IAM) roles. For detailed steps, see Tutorial: Configure federated access for Okta users to Athena using Lake Formation and JDBC.

Prerequisites

To use Amazon Athena and Lake Formation for federated access, you must meet the following requirements:

- You manage your corporate identities using an existing SAML-based identity provider, such as Okta or Microsoft Active Directory Federation Services (AD FS).
- You use the AWS Glue Data Catalog as a metadata store.
- You define and manage permissions in Lake Formation to access databases, tables, and columns in AWS Glue Data Catalog. For more information, see the AWS Lake Formation Developer Guide.
- You use version 2.0.14 or later of the Athena JDBC driver or version 1.1.3 or later of the Athena ODBC driver.

Considerations and limitations

When using the Athena JDBC or ODBC driver and Lake Formation to configure federated access to Athena, keep in mind the following points:

- Currently, the Athena JDBC driver and ODBC drivers support the Okta, Microsoft Active Directory Federation Services (AD FS), and Azure AD identity providers. Although the Athena JDBC driver has a generic SAML class that can be extended to use other identity providers, support for custom extensions that enable other identity providers (IdPs) for use with Athena may be limited.
- Federated access using the JDBC and ODBC drivers is not compatible with the IAM Identity Center trusted identity propagation feature.
- Currently, you cannot use the Athena console to configure support for IdP and SAML use with Athena. To configure this support, you use the third-party identity provider, the Lake Formation and IAM management consoles, and the JDBC or ODBC driver client.
- You should understand the SAML 2.0 specification and how it works with your identity provider before you configure your identity provider and SAML for use with Lake Formation and Athena.
- SAML providers and the Athena JDBC and ODBC drivers are provided by third parties, so support through AWS for issues related to their use may be limited.

Topics

Tutorial: Configure federated access for Okta users to Athena using Lake Formation and JDBC

This tutorial shows you how to configure Okta, AWS Lake Formation, AWS Identity and Access Management permissions, and the Athena JDBC driver to enable SAML-based federated use of Athena. Lake Formation provides fine-grained access control over the data that is available in Athena to the SAML-based user. To set up this configuration, the tutorial uses the Okta developer console, the AWS IAM and Lake Formation consoles, and the SQL Workbench/J tool.

Prerequisites

This tutorial assumes that you have done the following:

- Created an Amazon Web Services account. To create an account, visit the Amazon Web Services home page.

- Set up a query results location for Athena in Amazon S3.
- Registered an Amazon S3 data bucket location with Lake Formation.
- Defined a database and tables on the AWS Glue Data Catalog that point to your data in Amazon S3.
  - If you have not yet defined a table, either run a AWS Glue crawler or use Athena to define a database and one or more tables for the data that you want to access.
  - This tutorial uses a table based on the NYC taxi trips dataset available in the Registry of open data on AWS. The tutorial uses the database name `tripdb` and the table name `nyctaxi`.

Tutorial Steps

Step 1: Create an Okta account

This tutorial uses Okta as a SAML-based identity provider. If you do not already have an Okta account, you can create a free one. An Okta account is required so that you can create an Okta application for SAML authentication.

To create an Okta account

1. To use Okta, navigate to the Okta developer sign up page and create a free Okta trial account. The Developer Edition Service is free of charge up to the limits specified by Okta at developer.okta.com/pricing.
2. When you receive the activation email, activate your account.

An Okta domain name will be assigned to you. Save the domain name for reference. Later, you use the domain name (*<okta-idp-domain>*) in the JDBC string that connects to Athena.

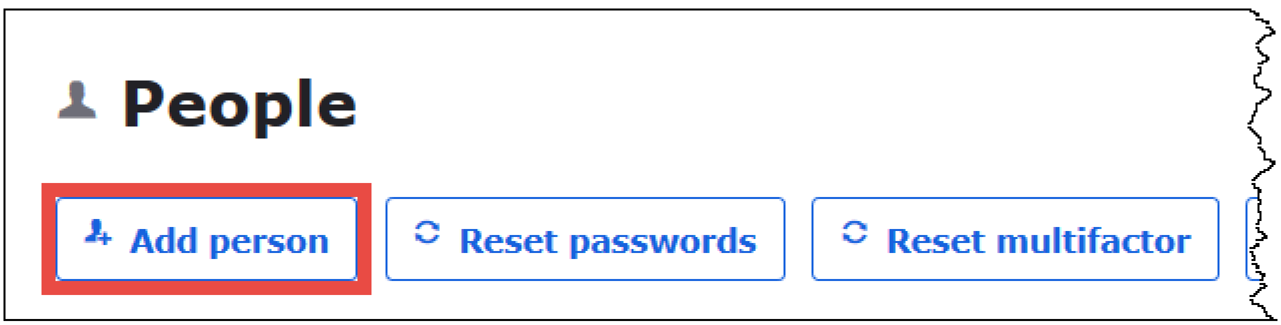
Step 2: Add users and groups to Okta

In this step, you use the Okta console to perform the following tasks:

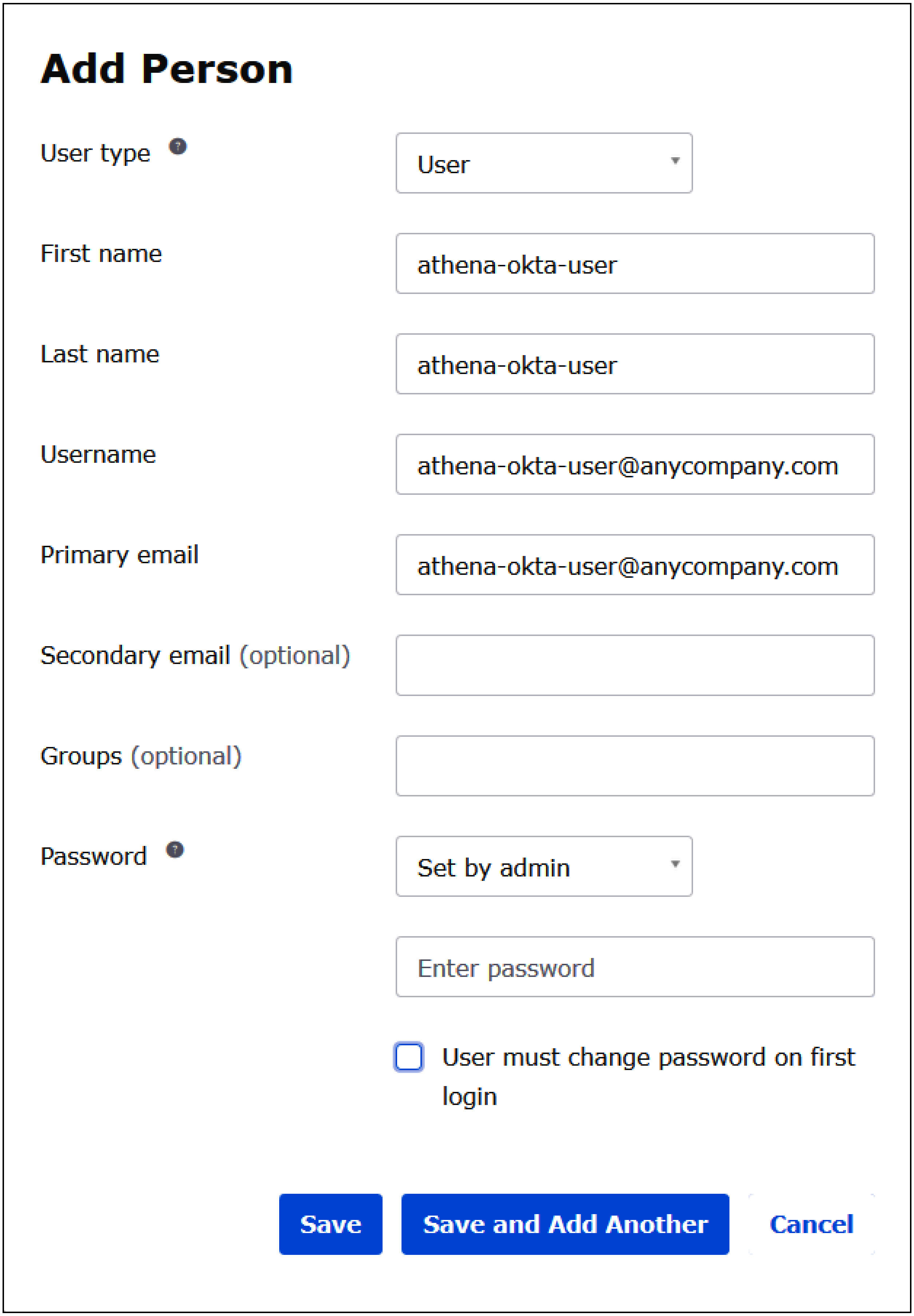
- Create two Okta users.
- Create two Okta groups.
- Add one Okta user to each Okta group.

To add users to Okta

1. After you activate your Okta account, log in as administrative user to the assigned Okta domain.
2. In the left navigation pane, choose **Directory**, and then choose **People**.
3. Choose **Add Person** to add a new user who will access Athena through the JDBC driver.

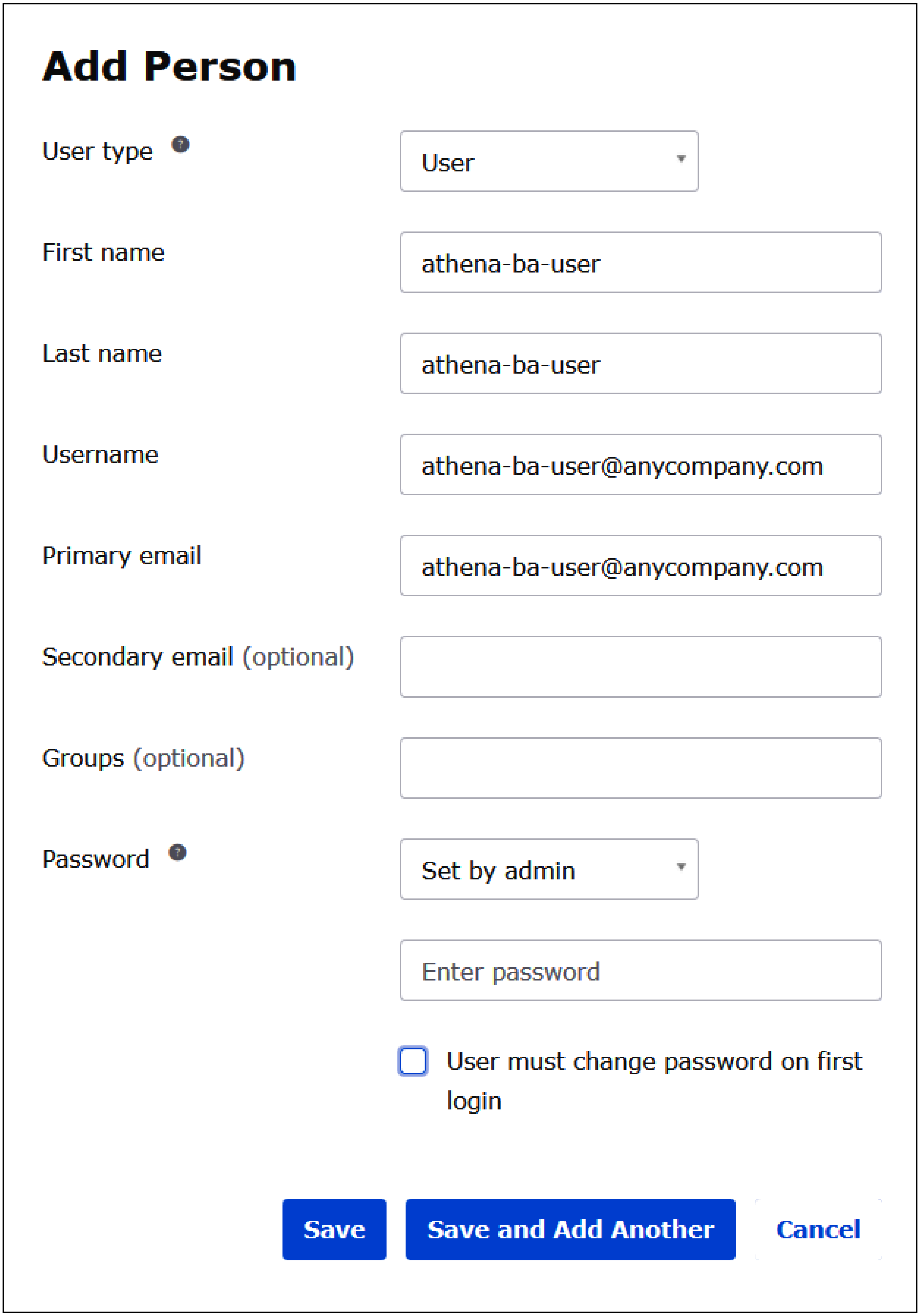


4. In the **Add Person** dialog box, enter the required information.
   - Enter values for **First name** and **Last name**. This tutorial uses *athena-okta-user*.
   - Enter a **Username** and **Primary email**. This tutorial uses *athena-okta-user@anycompany.com*.
   - For **Password**, choose **Set by admin**, and then provide a password. This tutorial clears the option for **User must change password on first login**; your security requirements may vary.


Add Person
User type
User
First name
athena-okta-user
Last name
athena-okta-user
Username
athena-okta-user@anycompany.com
Primary email
athena-okta-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

5. Choose **Save and Add Another**.

6. Enter the information for another user. This example adds the business analyst user *athena-ba-user@anycompany.com*.

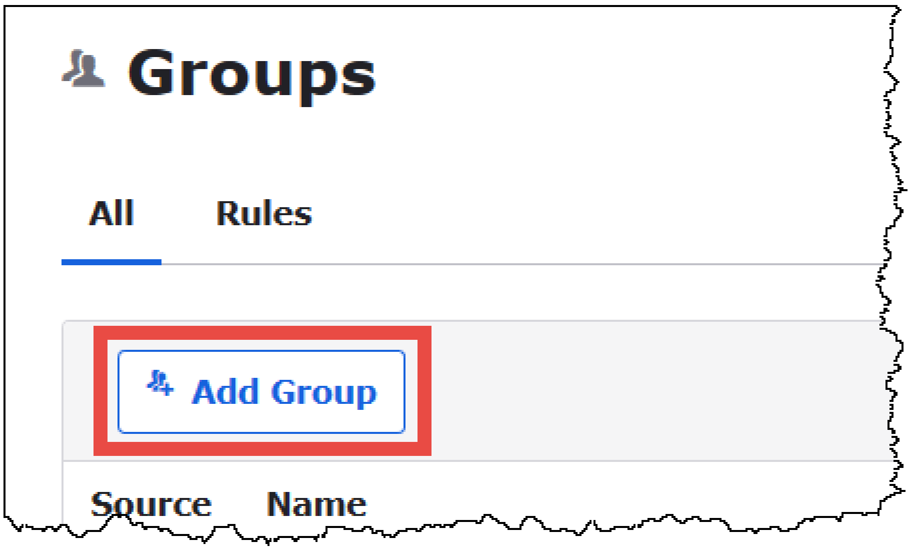

Add Person
User type
User
First name
athena-ba-user
Last name
athena-ba-user
Username
athena-ba-user@anycompany.com
Primary email
athena-ba-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

7. Choose **Save**.

In the following procedure, you provide access for two Okta groups through the Athena JDBC driver by adding a "Business Analysts" group and a "Developer" group.

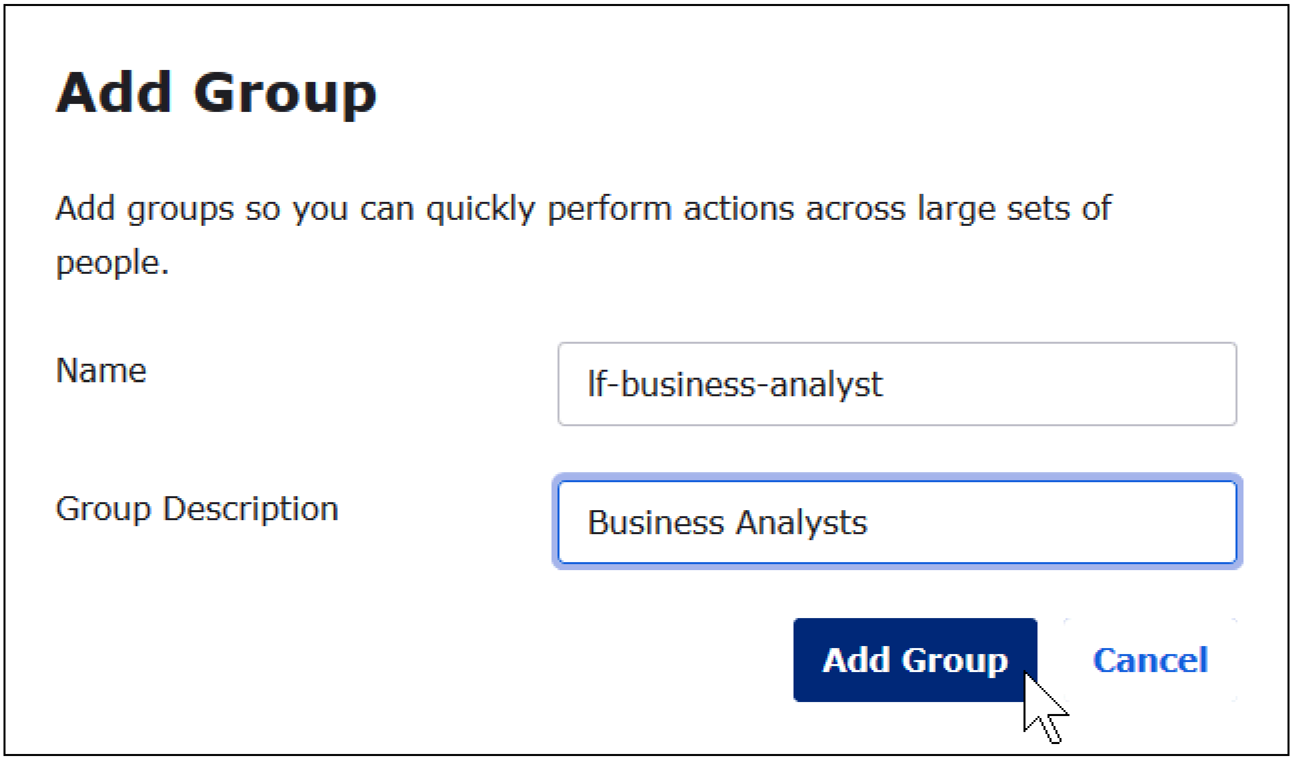
To add Okta groups

1. In the Okta navigation pane, choose **Directory**, and then choose **Groups**.
2. On the **Groups** page, choose **Add Group**.



3. In the **Add Group** dialog box, enter the required information.

   - For **Name**, enter *`lf-business-analyst`*.
   - For **Group Description**, enter *`Business Analysts`*.

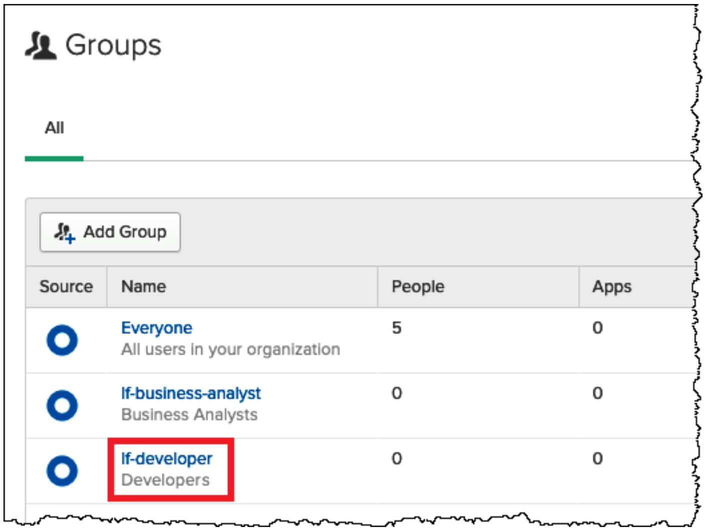


4. Choose **Add Group**.
5. On the **Groups** page, choose **Add Group** again. This time you will enter information for the Developer group.
6. Enter the required information.
   - For **Name**, enter *`lf-developer`*.
   - For **Group Description**, enter *`Developers`*.
7. Choose **Add Group**.

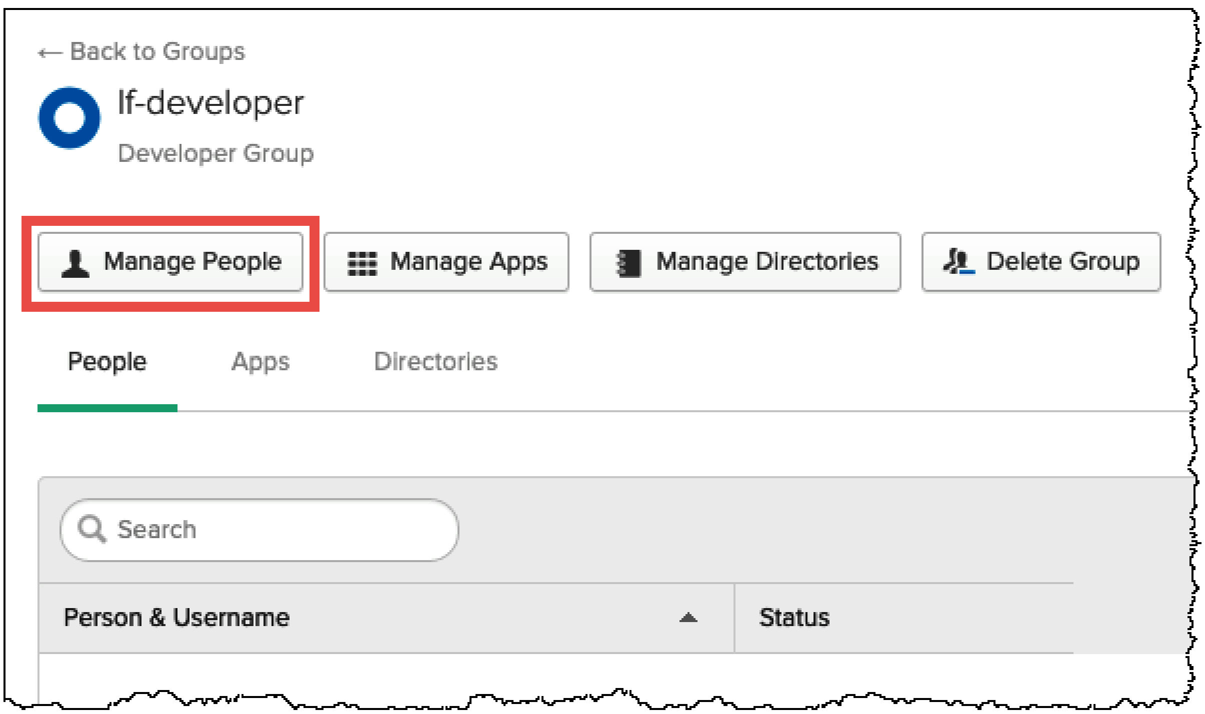
Now that you have two users and two groups, you are ready to add a user to each group.

To add users to groups

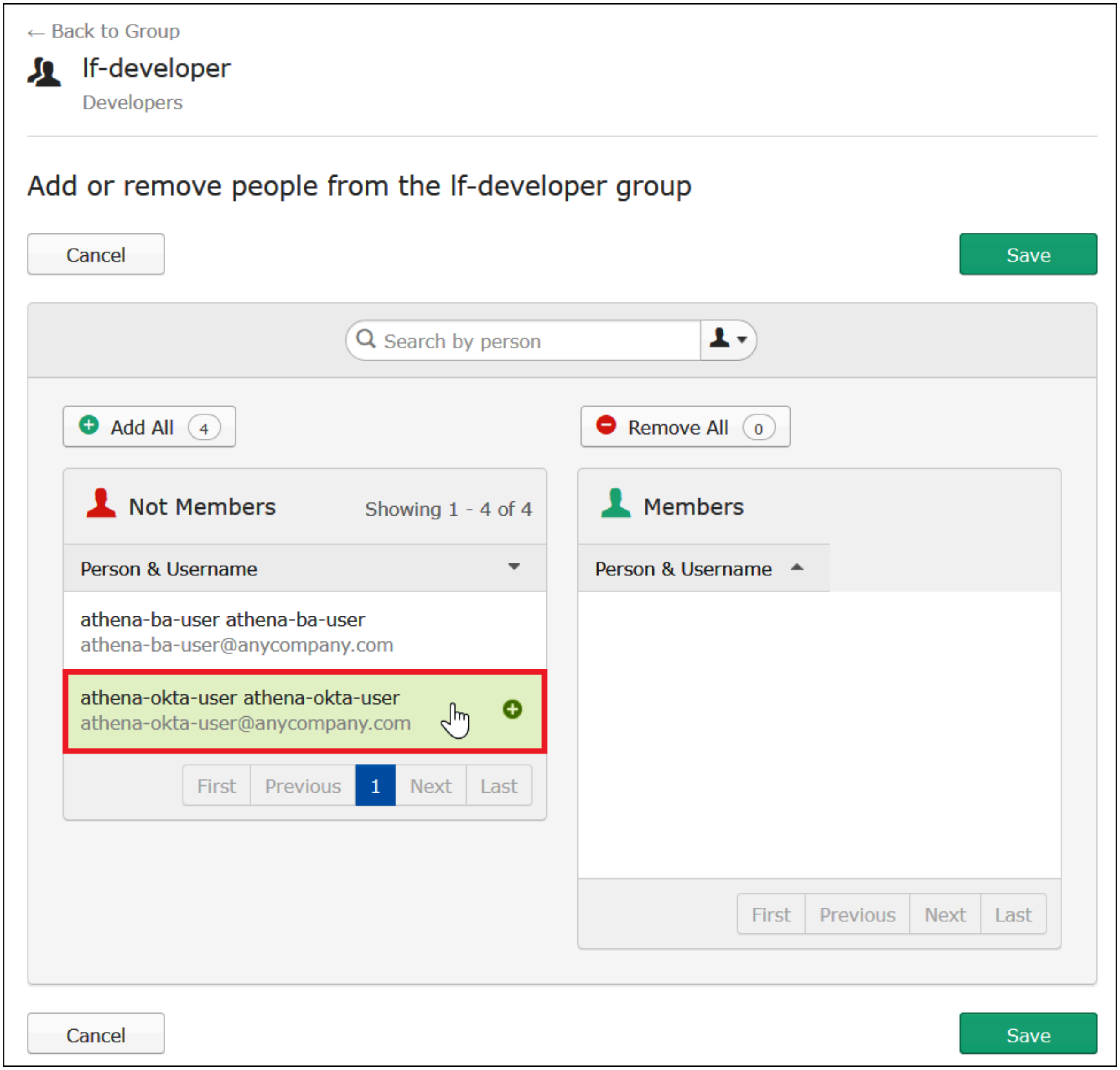
1. On the **Groups** page, choose the **lf-developer** group that you just created. You will add one of the Okta users that you created as a developer to this group.



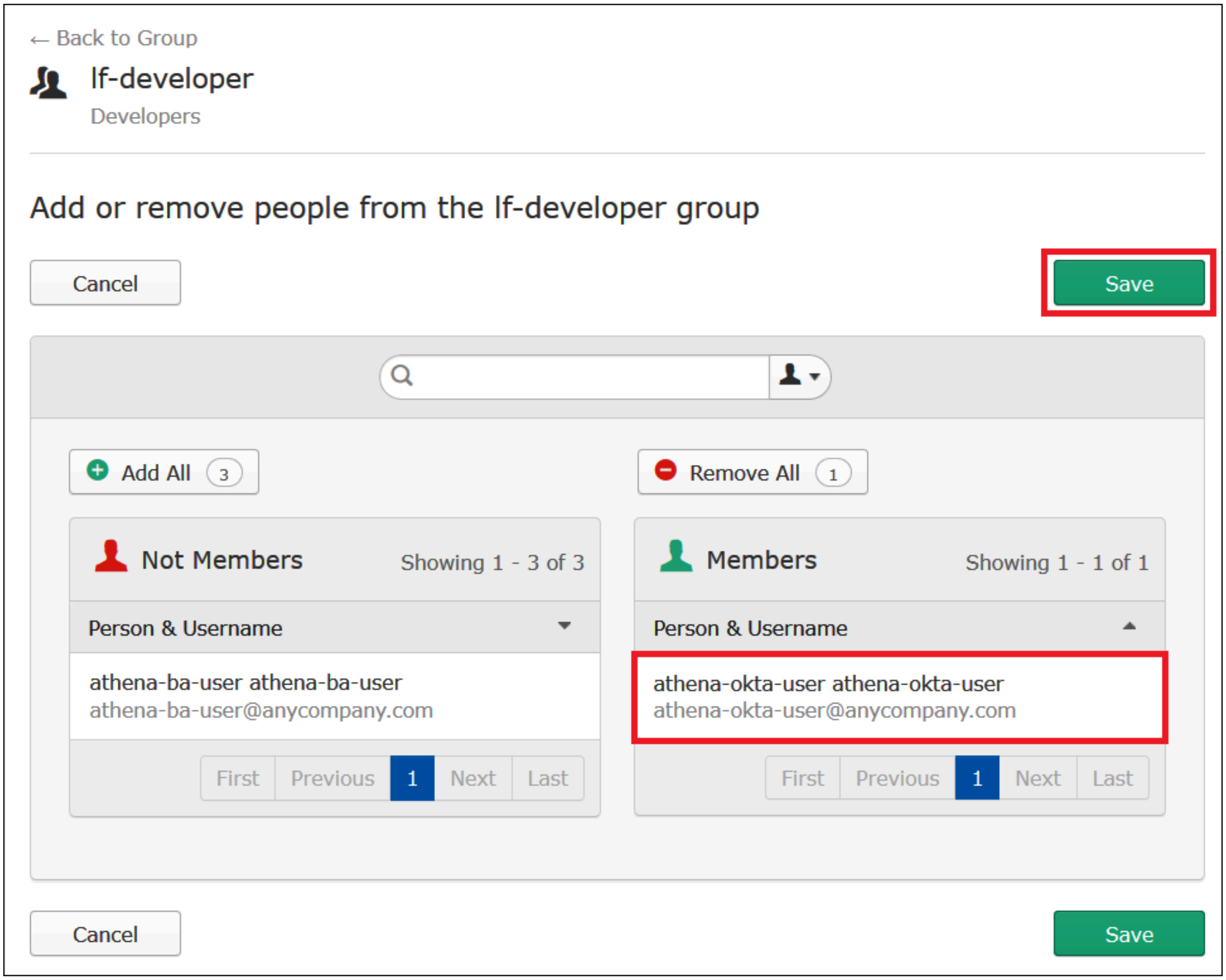
2. Choose **Manage People**.



3. From the **Not Members** list, choose **athena-okta-user**.

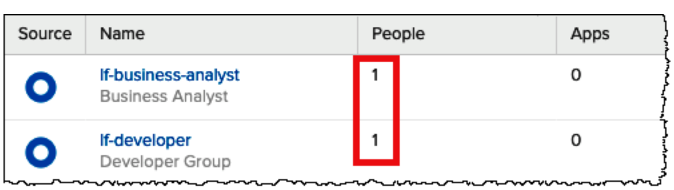


The entry for the user moves from the **Not Members** list on the left to the **Members** list on the right.



4. Choose **Save**.
5. Choose **Back to Group**, or choose **Directory**, and then choose **Groups**.
6. Choose the **lf-business-analyst** group.
7. Choose **Manage People**.
8. Add the **athena-ba-user** to the **Members** list of the **lf-business-analyst** group, and then choose **Save**.
9. Choose **Back to Group**, or choose **Directory**, **Groups**.

   The **Groups** page now shows that each group has one Okta user.



Source	Name	People	Apps
	lf-business-analyst Business Analyst	1	0
	lf-developer Developer Group	1	0

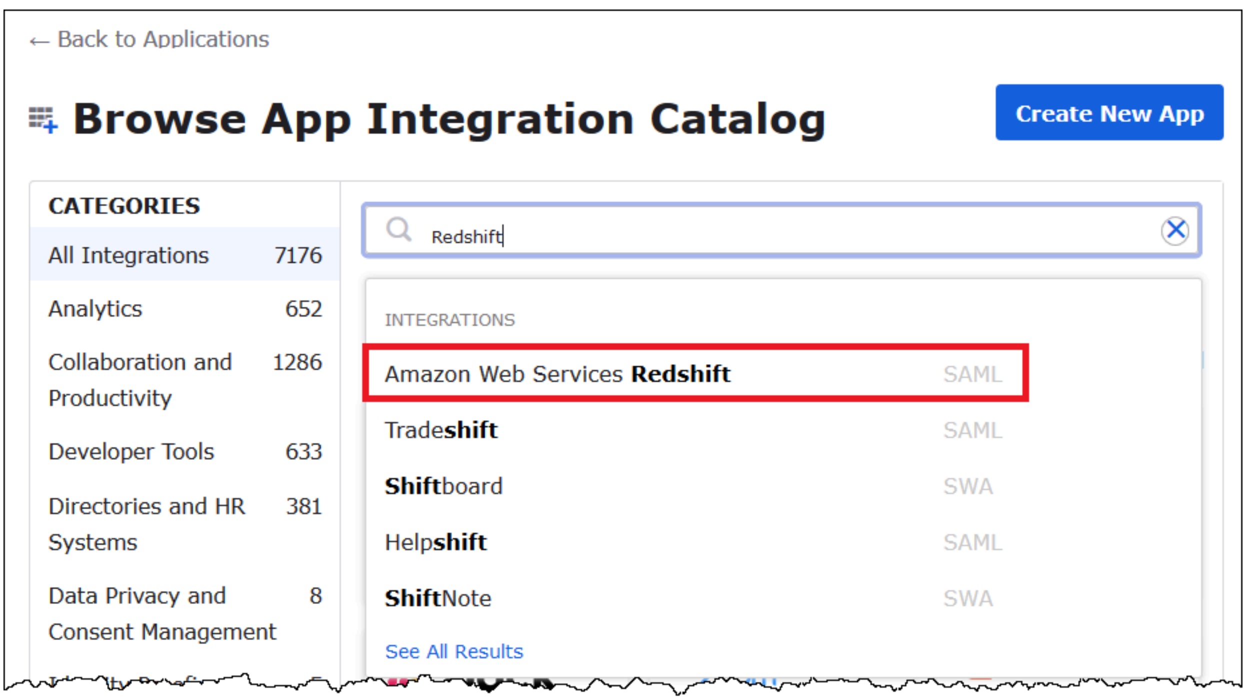
Step 3: Set up an Okta application for SAML authentication

In this step, you use the Okta developer console to perform the following tasks:

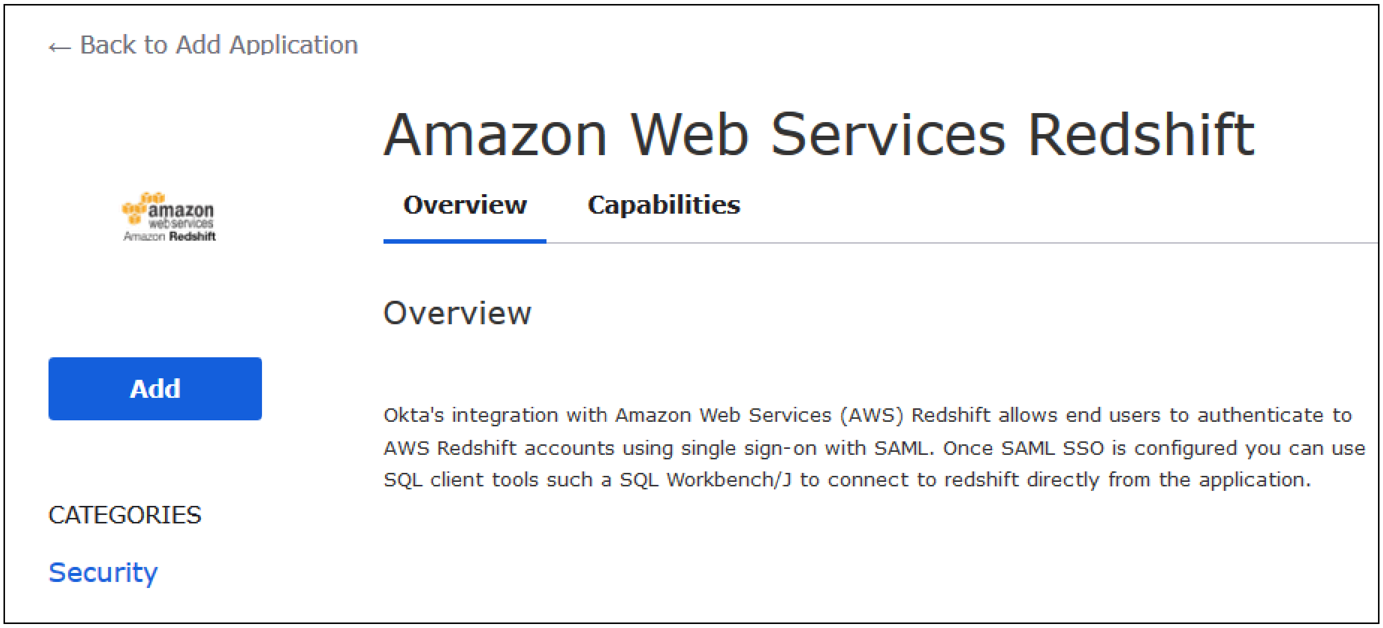
- Add a SAML application for use with AWS.
- Assign the application to the Okta user.
- Assign the application to an Okta group.
- Download the resulting identity provider metadata for later use with AWS.

To add an application for SAML authentication

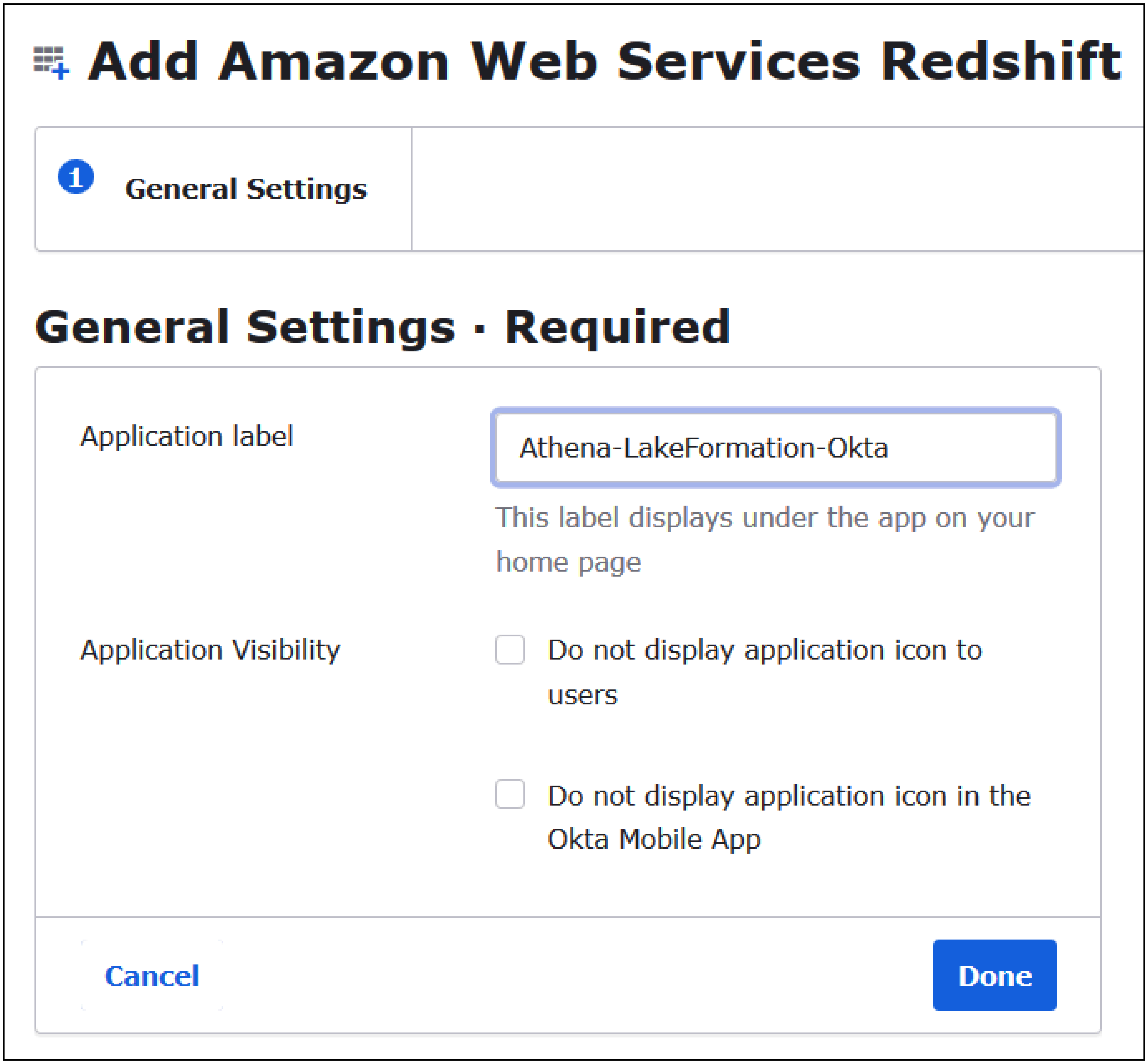
1. In the Okta navigation pane, choose **Applications**, **Applications** so that you can configure an Okta application for SAML authentication to Athena.
2. Click **Browse App Catalog**.
3. In the search box, enter **`Redshift`**.
4. Choose **Amazon Web Services Redshift**. The Okta application in this tutorial uses the existing SAML integration for Amazon Redshift.



5. On the **Amazon Web Services Redshift** page, choose **Add** to create a SAML-based application for Amazon Redshift.



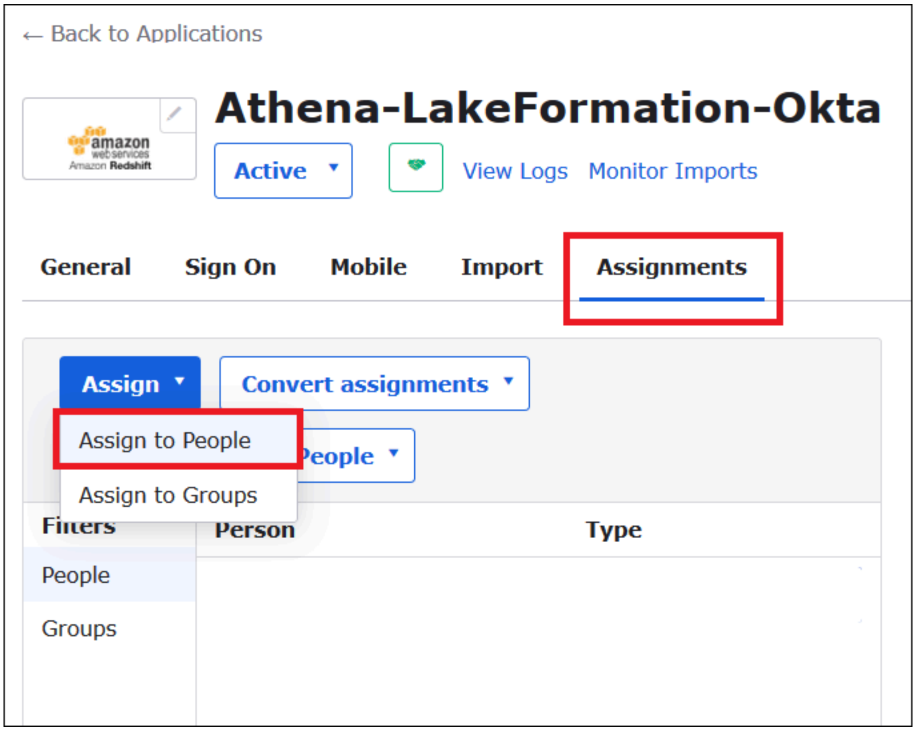
6. For **Application label**, enter `Athena-LakeFormation-Okta`, and then choose **Done**.



Now that you have created an Okta application, you can assign it to the users and groups that you created.

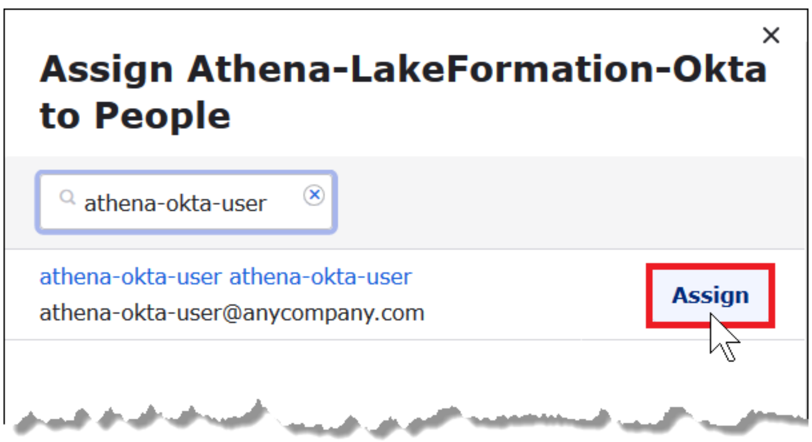
To assign the application to users and groups

1. On the **Applications** page, choose the **Athena-LakeFormation-Okta** application.
2. On the **Assignments** tab, choose **Assign**, **Assign to People**.



3. In the **Assign Athena-LakeFormation-Okta to People** dialog box, find the **athena-okta-user** user that you created previously.

4. Choose **Assign** to assign the user to the application.

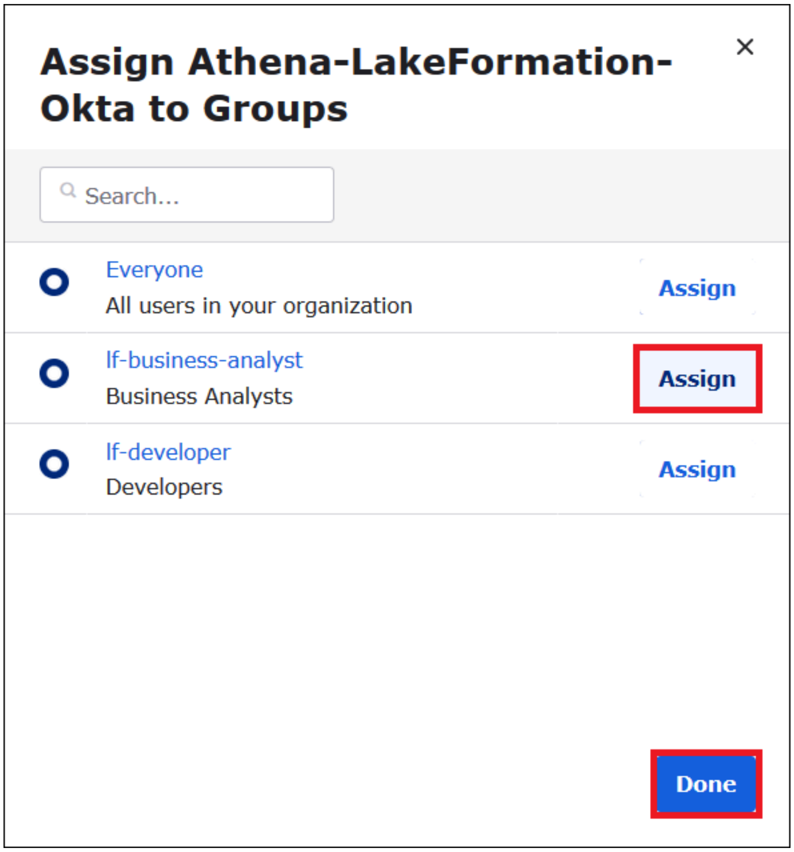


5. Choose **Save and Go Back**.

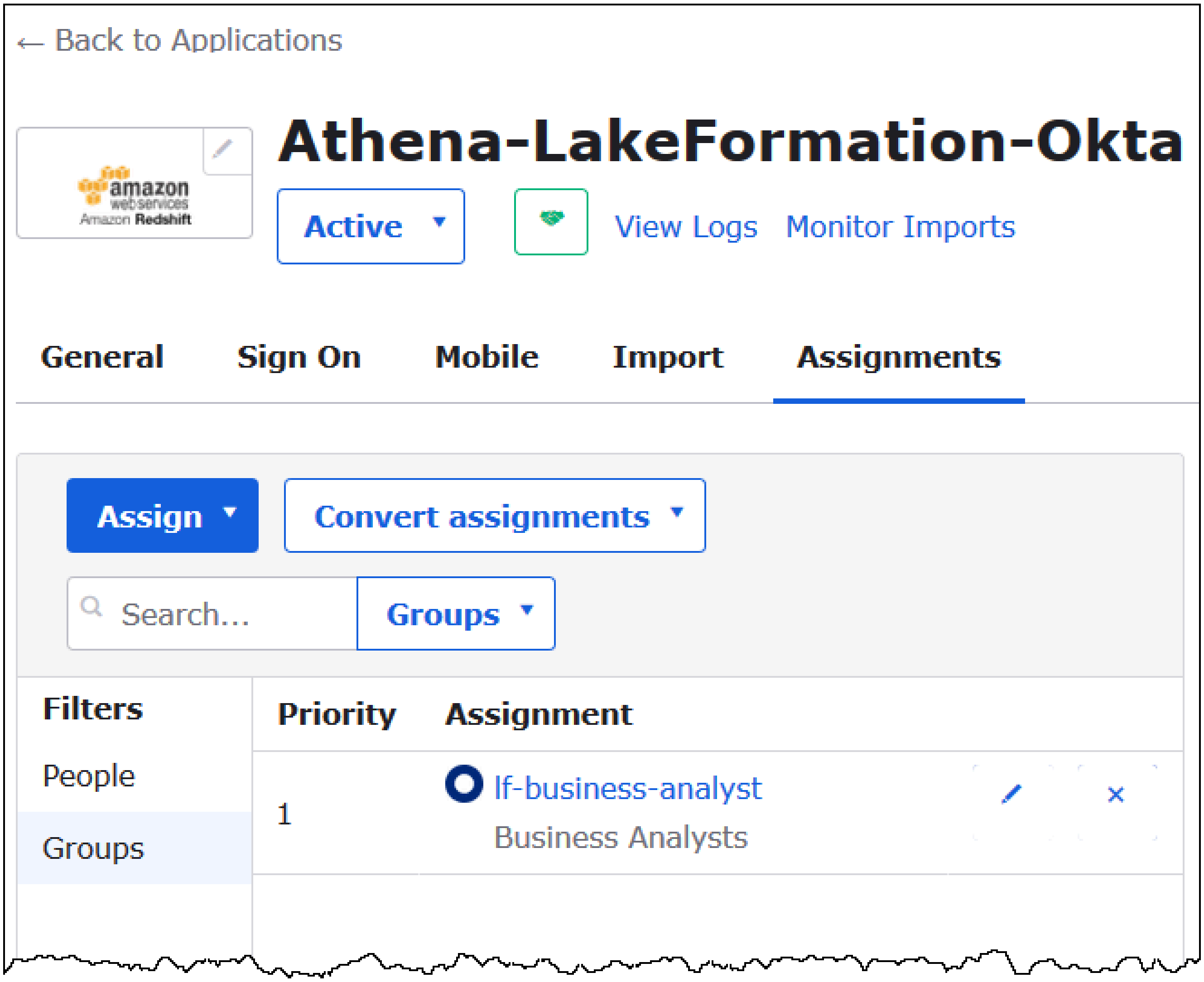
6. Choose **Done**.

7. On the **Assignments** tab for the **Athena-LakeFormation-Okta** application, choose **Assign**, **Assign to Groups**.

8. For **lf-business-analyst**, choose **Assign** to assign the **Athena-LakeFormation-Okta** application to the **lf-business-analyst** group, and then choose **Done**.



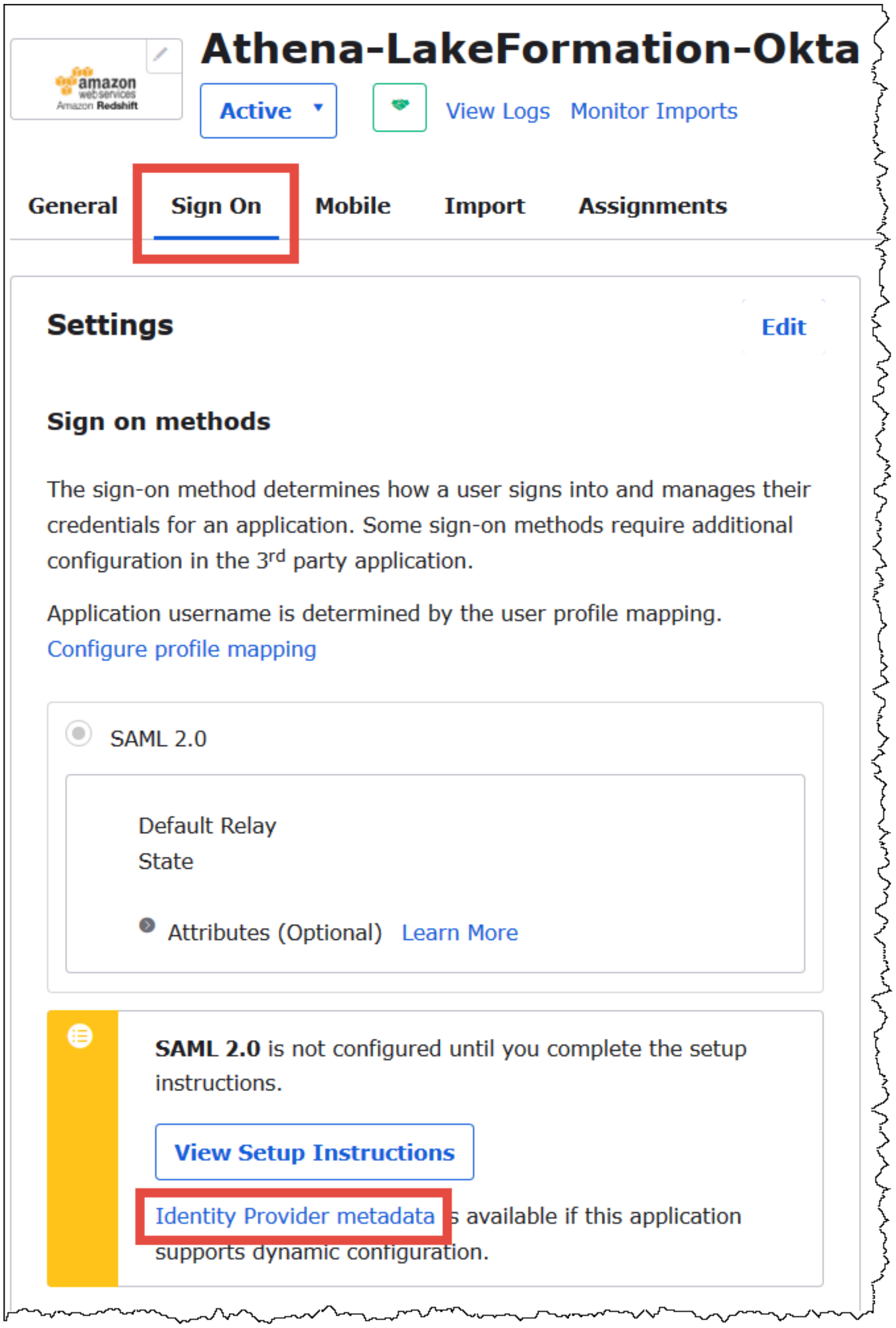
The group appears in the list of groups for the application.



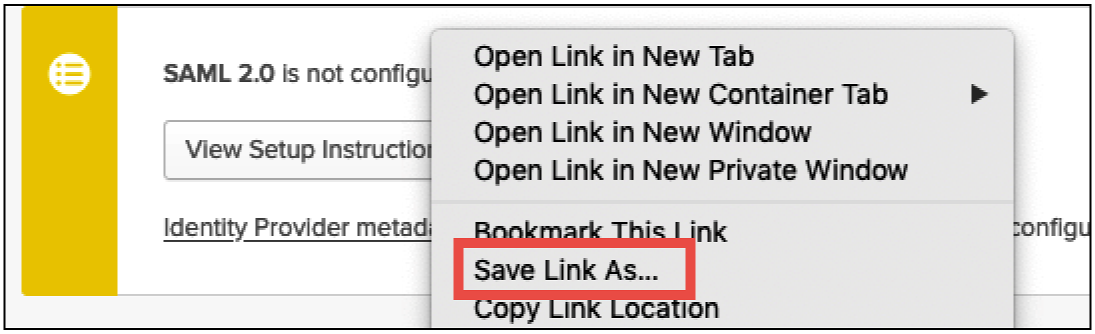
Now you are ready to download the identity provider application metadata for use with AWS.

To download the application metadata

1. Choose the Okta application **Sign On** tab, and then right-click **Identity Provider metadata**.



2. Choose **Save Link As** to save the identity provider metadata, which is in XML format, to a file. Give it a name that you recognize (for example, `Athena-LakeFormation-idp-metadata.xml`).



Step 4: Create an AWS SAML Identity Provider and Lake Formation access IAM role

In this step, you use the AWS Identity and Access Management (IAM) console to perform the following tasks:

- Create an identity provider for AWS.
- Create an IAM role for Lake Formation access.
- Add the AmazonAthenaFullAccess managed policy to the role.
- Add a policy for Lake Formation and AWS Glue to the role.
- Add a policy for Athena query results to the role.

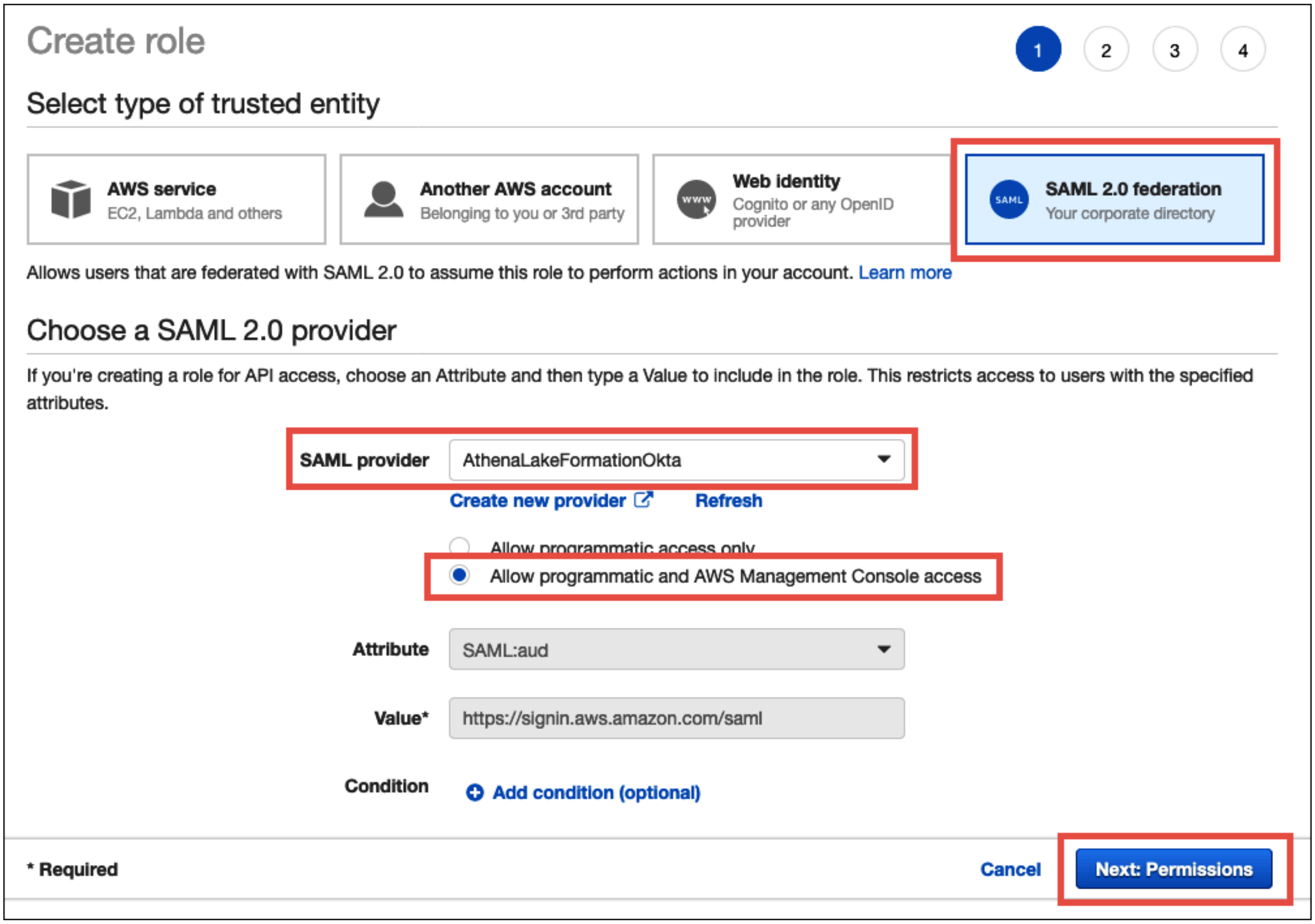
To create an AWS SAML identity provider

1. Sign in to the **Amazon Web Services account console** as **Amazon Web Services account administrator** and navigate to the **IAM** console (https://console.aws.amazon.com/iam/).
2. In the navigation pane, choose **Identity providers**, and then click **Add provider**.
3. On the **Configure provider** screen, enter the following information:
   - For **Provider type**, choose **SAML**.
   - For **Provider name**, enter `AthenaLakeFormationOkta`.
   - For **Metadata document**, use the **Choose file** option to upload the identity provider (IdP) metadata XML file that you downloaded.
4. Choose **Add provider**.

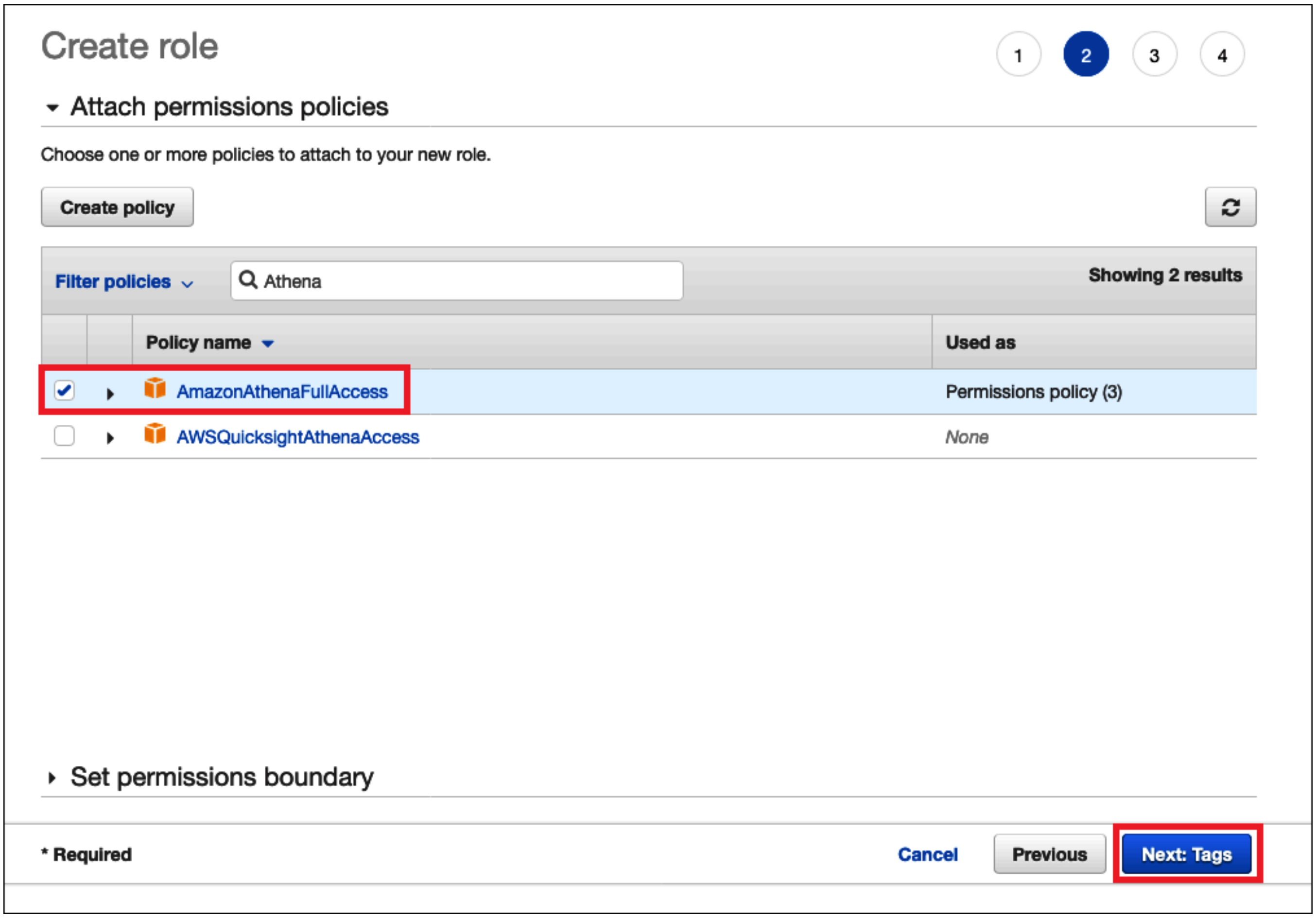
Next, you create an IAM role for AWS Lake Formation access. You add two inline policies to the role. One policy provides permissions to access Lake Formation and the AWS Glue APIs. The other policy provides access to Athena and the Athena query results location in Amazon S3.

To create an IAM role for AWS Lake Formation access

1. In the IAM console navigation pane, choose **Roles**, and then choose **Create role**.
2. On the **Create role** page, perform the following steps:

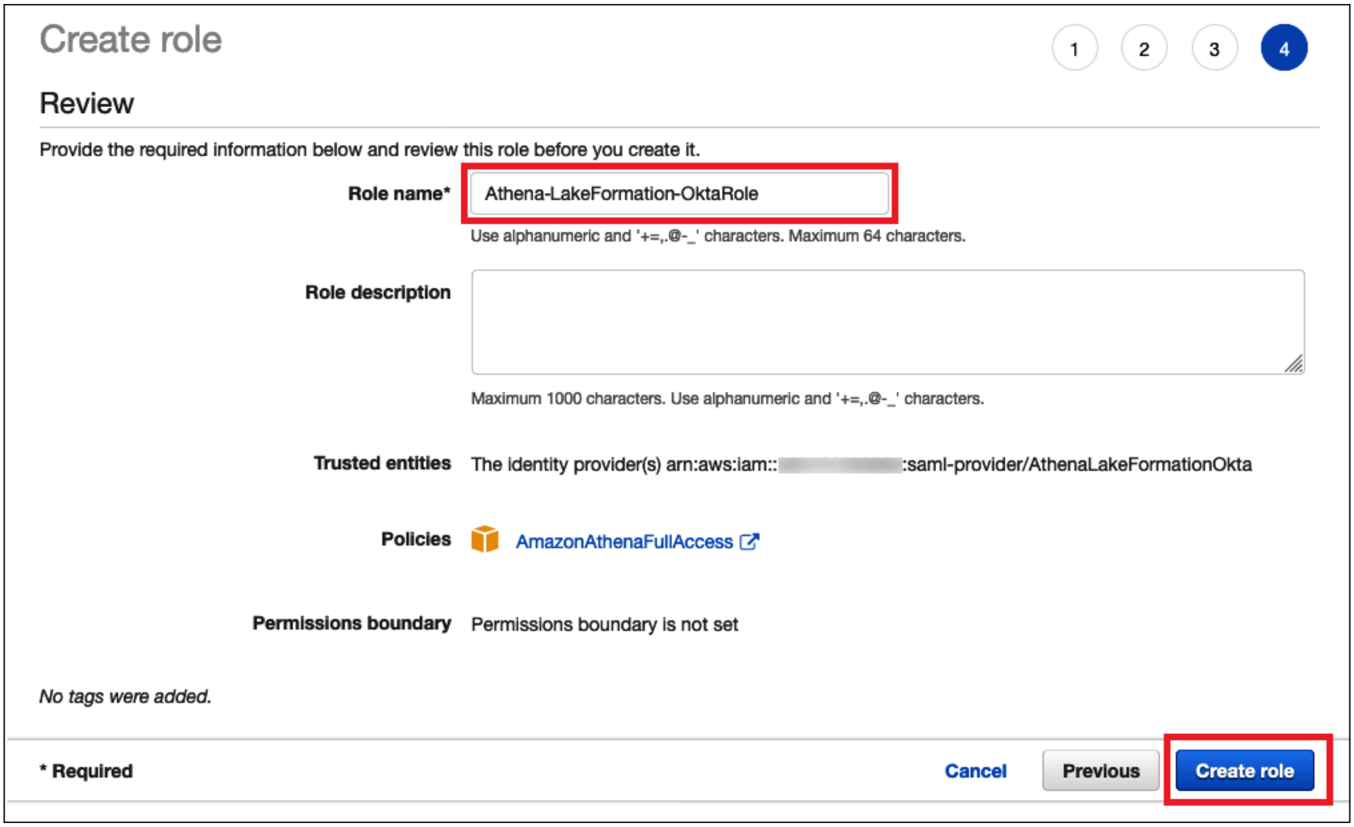


   a. For **Select type of trusted entity**, choose **SAML 2.0 Federation.**
   b. For **SAML provider**, select **AthenaLakeFormationOkta**.
   c. For **SAML provider**, select the option **Allow programmatic and AWS Management Console access**.
   d. Choose **Next: Permissions**.
3. On the **Attach Permissions policies** page, for **Filter policies**, enter **Athena**.
4. Select the **AmazonAthenaFullAccess** managed policy, and then choose **Next: Tags**.



5. On the **Add tags** page, choose **Next: Review**.

6. On the **Review** page, for **Role name**, enter a name for the role (for example, *`Athena-LakeFormation-OktaRole`*), and then choose **Create role**.



Next, you add inline policies that allow access to Lake Formation, AWS Glue APIs, and Athena query results in Amazon S3.

Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

To add an inline policy to the role for Lake Formation and AWS Glue

1. From the list of roles in the IAM console, choose the newly created `Athena-LakeFormation-OktaRole`.
2. On the **Summary** page for the role, on the **Permissions** tab, choose **Add inline policy**.
3. On the **Create policy** page, choose **JSON**.
4. Add an inline policy like the following that provides access to Lake Formation and the AWS Glue APIs.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
```

```
            "Action": [
                "lakeformation:GetDataAccess",
                "glue:GetTable",
                "glue:GetTables",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:CreateDatabase",
                "glue:GetUserDefinedFunction",
                "glue:GetUserDefinedFunctions"
            ],
            "Resource": "*"
        }
}
```

5. Choose **Review policy**.
6. For **Name**, enter a name for the policy (for example, **`LakeFormationGlueInlinePolicy`**).
7. Choose **Create policy**.

To add an inline policy to the role for the Athena query results location

1. On the **Summary** page for the `Athena-LakeFormation-OktaRole` role, on the **Permissions** tab, choose **Add inline policy**.
2. On the **Create policy** page, choose **JSON**.
3. Add an inline policy like the following that allows the role access to the Athena query results location. Replace the *`<athena-query-results-bucket>`* placeholders in the example with the name of your Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AthenaQueryResultsPermissionsForS3",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<athena-query-results-bucket>",
```

```
                    "arn:aws:s3:::<athena-query-results-bucket>/*"
                ]
            }
        ]
}
```

4. Choose **Review policy**.
5. For **Name**, enter a name for the policy (for example, **AthenaQueryResultsInlinePolicy**).
6. Choose **Create policy**.

Next, you copy the ARN of the Lake Formation access role and the ARN of the SAML provider that you created. These are required when you configure the Okta SAML application in the next section of the tutorial.

To copy the role ARN and SAML identity provider ARN

1. In the IAM console, on the **Summary** page for the `Athena-LakeFormation-OktaRole` role, choose the **Copy to clipboard** icon next to **Role ARN**. The ARN has the following format:

```
arn:aws:iam::<account-id>:role/Athena-LakeFormation-OktaRole
```

2. Save the full ARN securely for later reference.
3. In the IAM console navigation pane, choose **Identity providers**.
4. Choose the **AthenaLakeFormationOkta** provider.
5. On the **Summary** page, choose the **Copy to clipboard** icon next to **Provider ARN**. The ARN should look like the following:

```
arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta
```

6. Save the full ARN securely for later reference.

Step 5: Add the IAM role and SAML Identity Provider to the Okta application

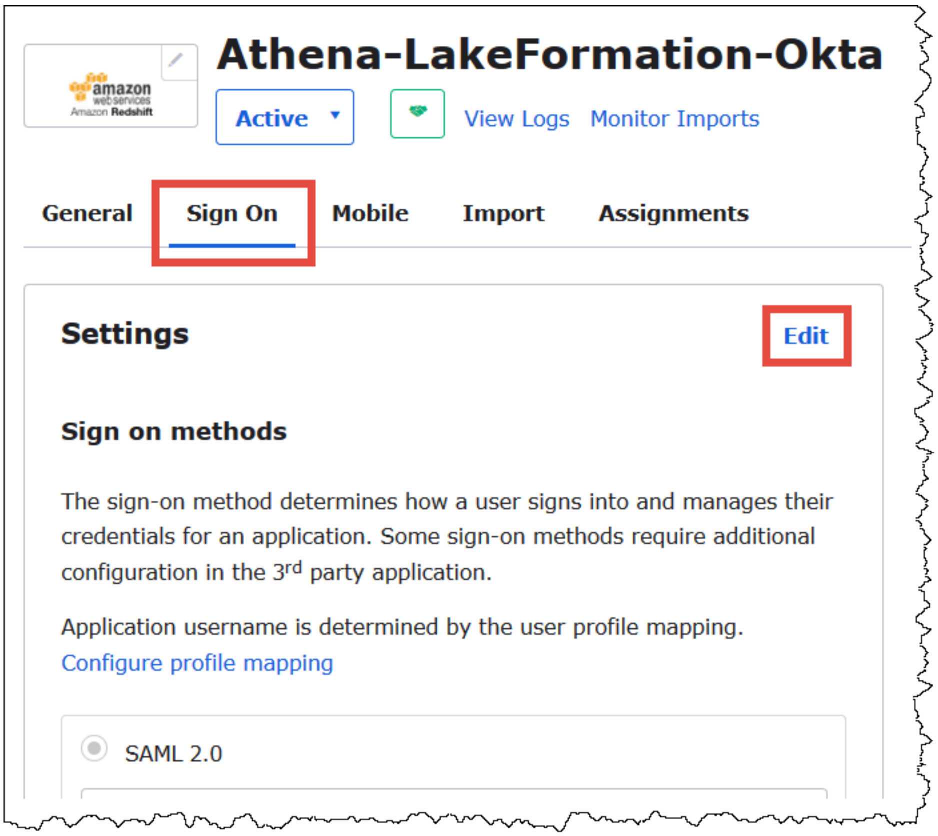
In this step, you return to the Okta developer console and perform the following tasks:

- Add user and group Lake Formation URL attributes to the Okta application.
- Add the ARN for the identity provider and the ARN for the IAM role to the Okta application.

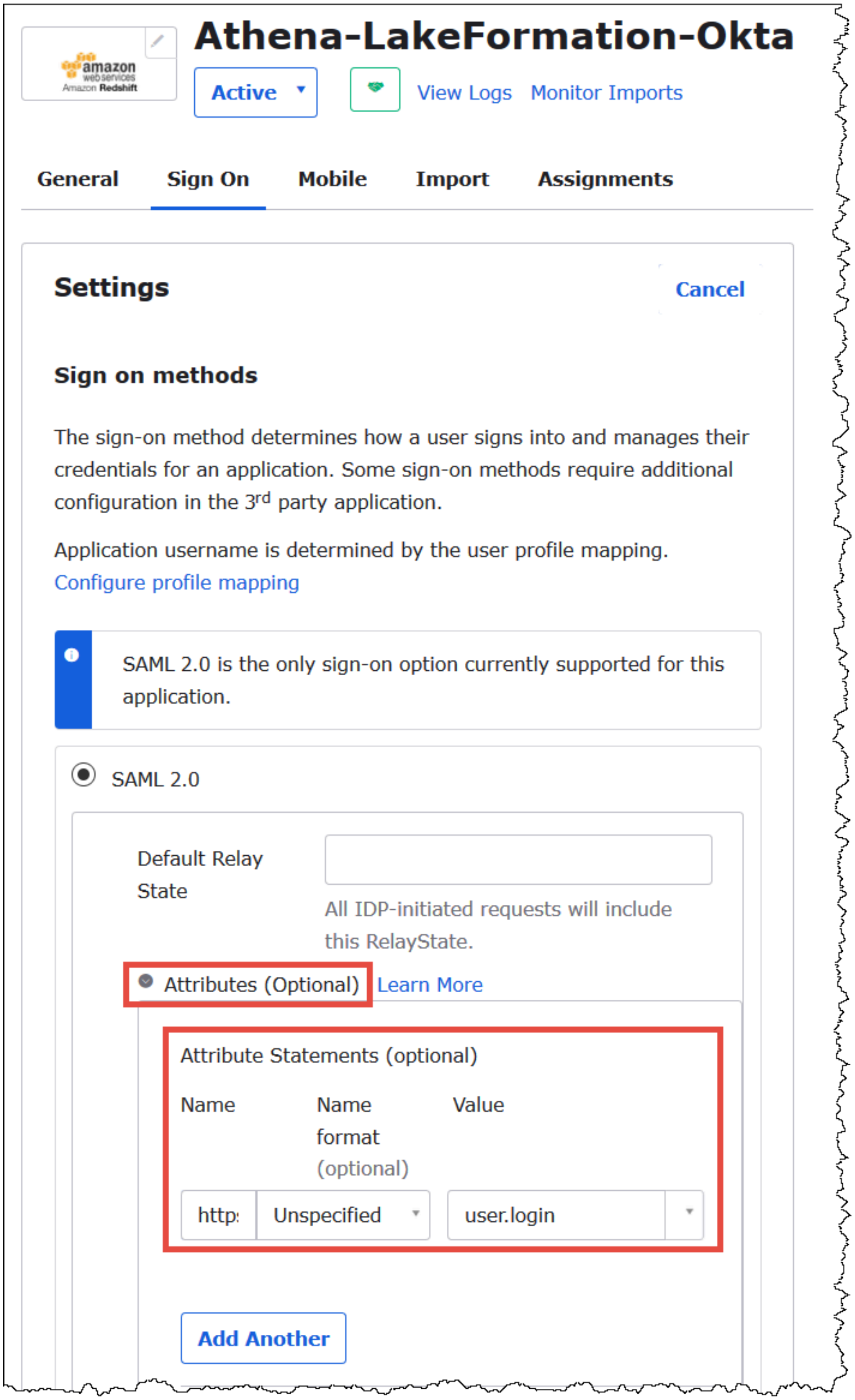
- Copy the Okta application ID. The Okta application ID is required in the JDBC profile that connects to Athena.

To add user and group Lake Formation URL attributes to the Okta application

1. Sign into the Okta developer console.
2. Choose the **Applications** tab, and then choose the `Athena-LakeFormation-Okta` application.
3. Choose on the **Sign On** tab for the application, and then choose **Edit**.



4. Choose **Attributes (optional)** to expand it.



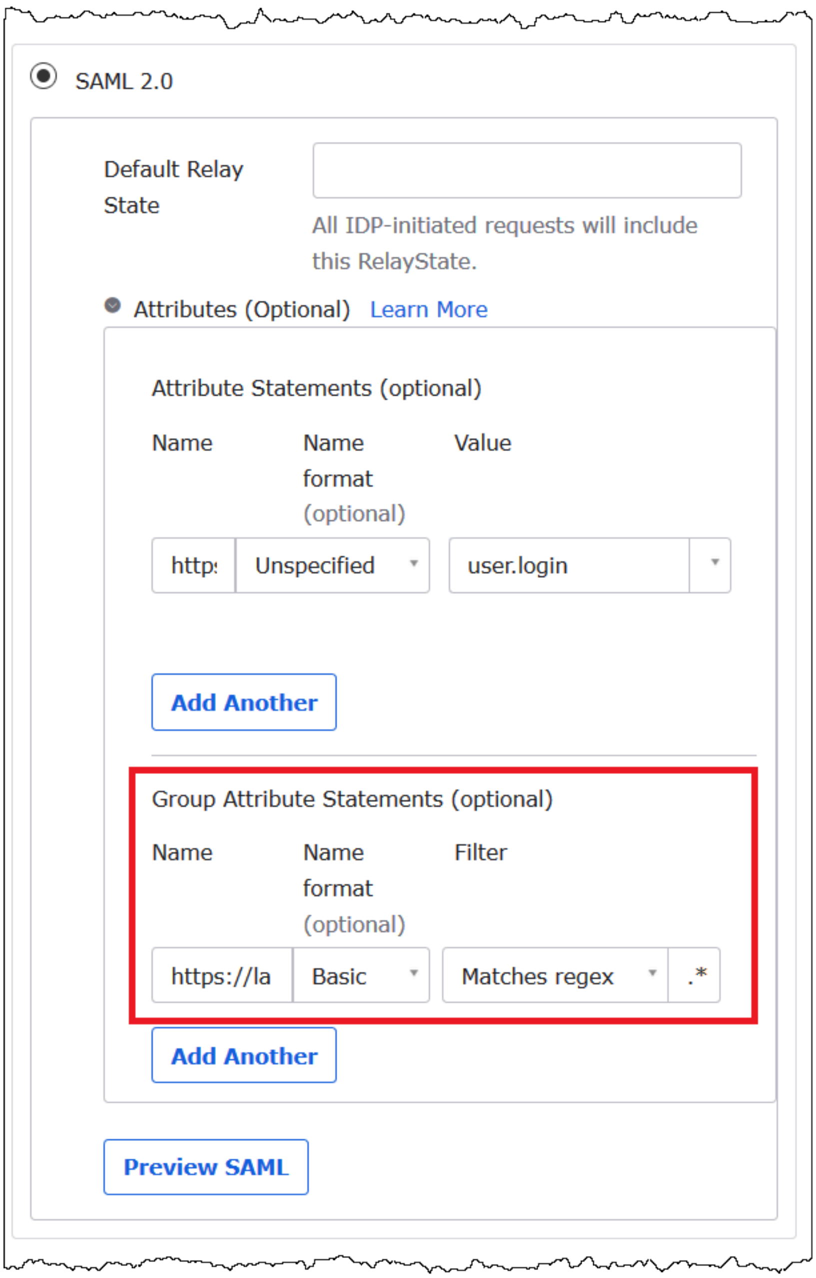
5. For **Attribute Statements (optional)**, add the following attribute:

   - For **Name**, enter **`https://lakeformation.amazon.com/SAML/Attributes/Username`**.

- For **Value**, enter **`user.login`**

6. Under **Group Attribute Statements (optional)**, add the following attribute:

   - For **Name**, enter **`https://lakeformation.amazon.com/SAML/Attributes/Groups`**.
   - For **Name format**, enter **`Basic`**
   - For **Filter**, choose **Matches regex**, and then enter **`.*`** in the filter box.

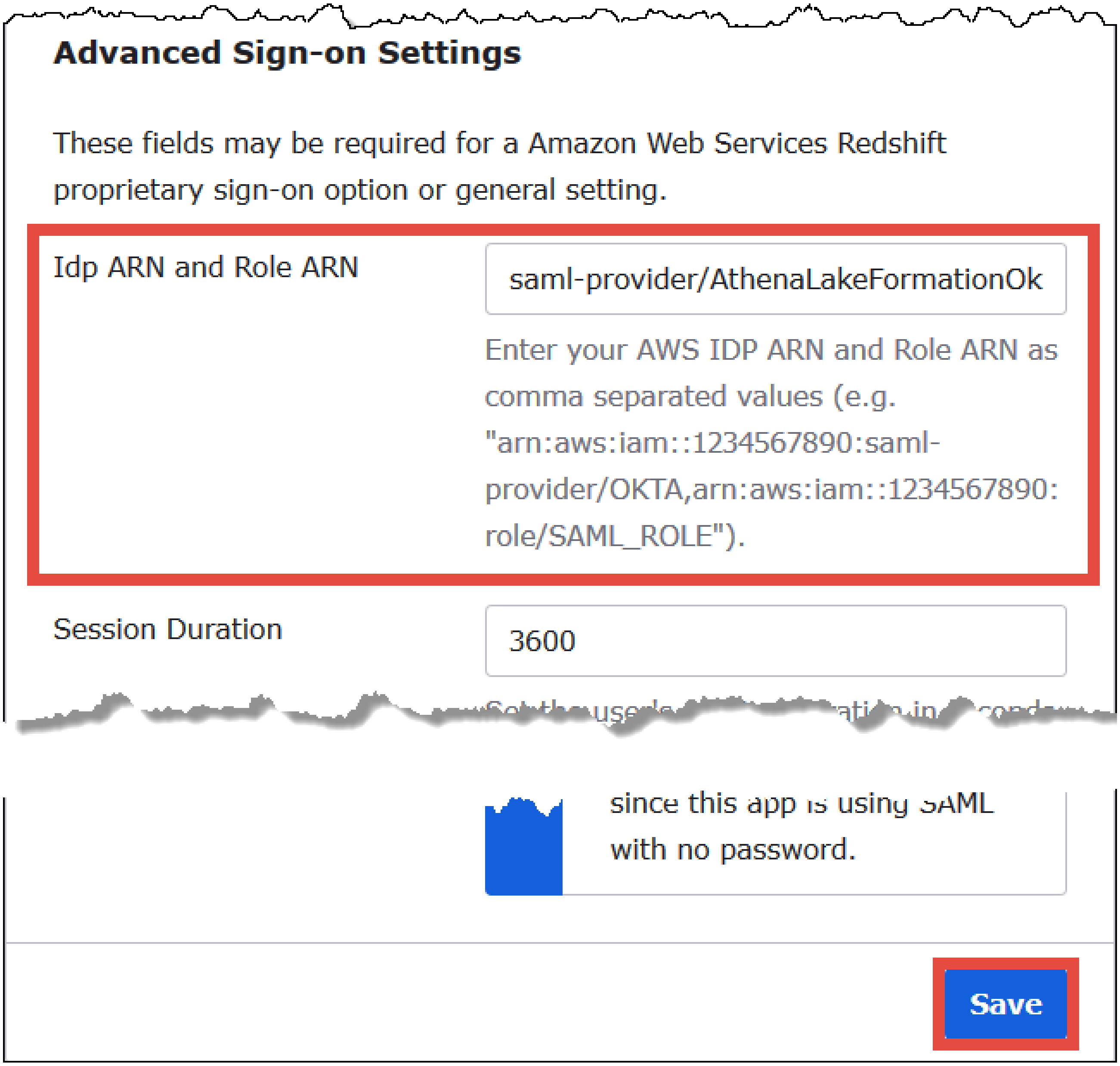


7. Scroll down to the **Advanced Sign-On Settings** section, where you will add the identity provider and IAM Role ARNs to the Okta application.

To add the ARNs for the identity provider and IAM role to the Okta application

1. For **Idp ARN and Role ARN**, enter the AWS identity provider ARN and role ARN as comma separated values in the format *<saml-arn>*,*<role-arn>*. The combined string should look like the following:

```
arn:aws:iam::<account-id>:saml-provider/
AthenaLakeFormationOkta,arn:aws:iam::<account-id>:role/Athena-LakeFormation-
OktaRole
```

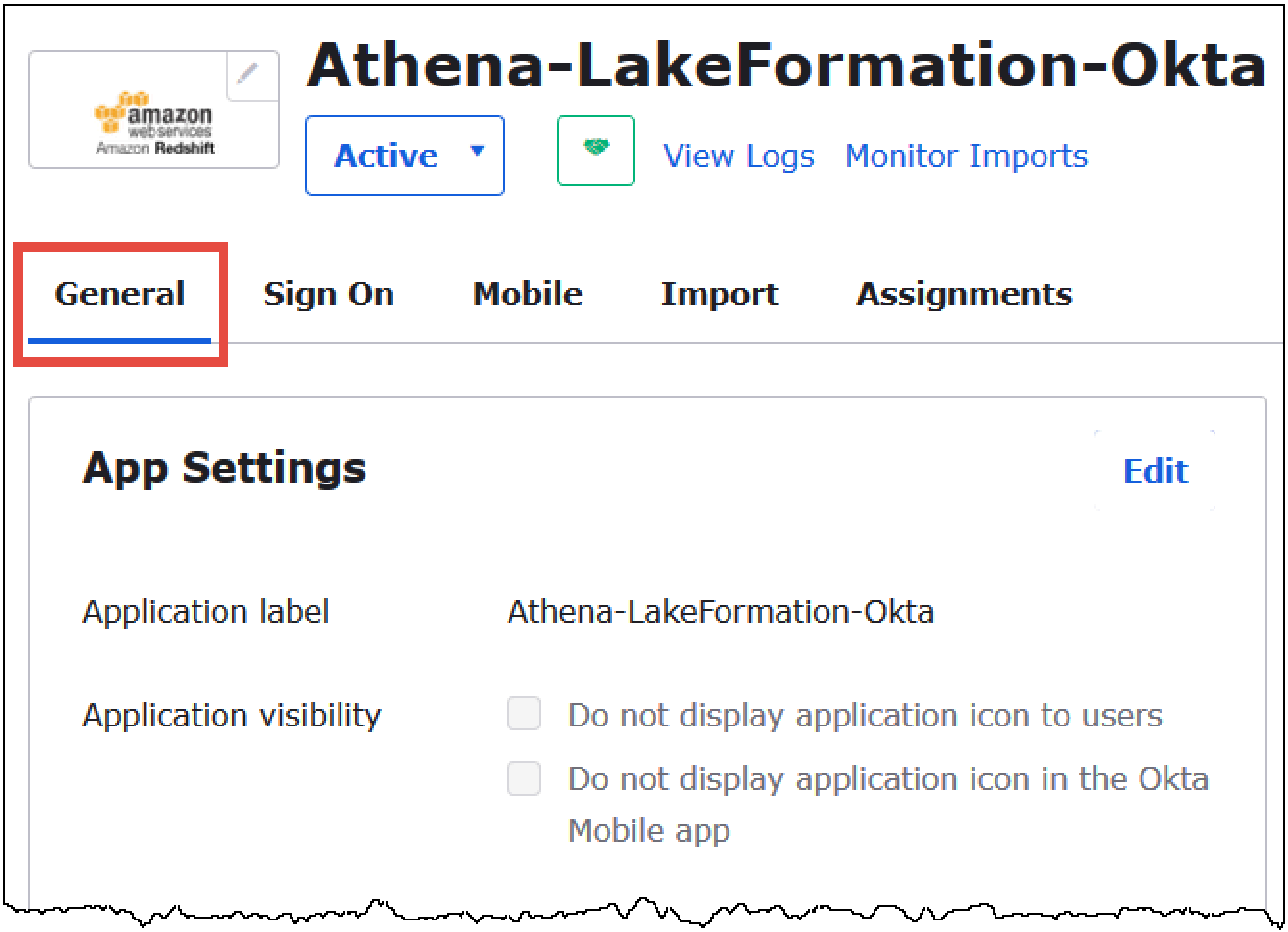


2. Choose **Save**.

Next, you copy the Okta application ID. You will require this later for the JDBC string that connects to Athena.

To find and copy the Okta application ID

1. Choose the **General** tab of the Okta application.



2. Scroll down to the **App Embed Link** section.
3. From **Embed Link**, copy and securely save the Okta application ID portion of the URL. The Okta application ID is the part of the URL after `amazon_aws_redshift/` but before the next forward slash. For example, if the URL contains `amazon_aws_redshift/aaa/bbb`, the application ID is `aaa`.

> **Note**
>
> The embed link cannot be used to log directly into the Athena console to view databases. The Lake Formation permissions for SAML users and groups are recognized only when you use the JDBC or ODBC driver to submit queries to Athena. To view the databases, you can use the SQL Workbench/J tool, which uses the JDBC driver to connect to Athena. The SQL Workbench/J tool is covered in Step 7: Verify access through the Athena JDBC client.

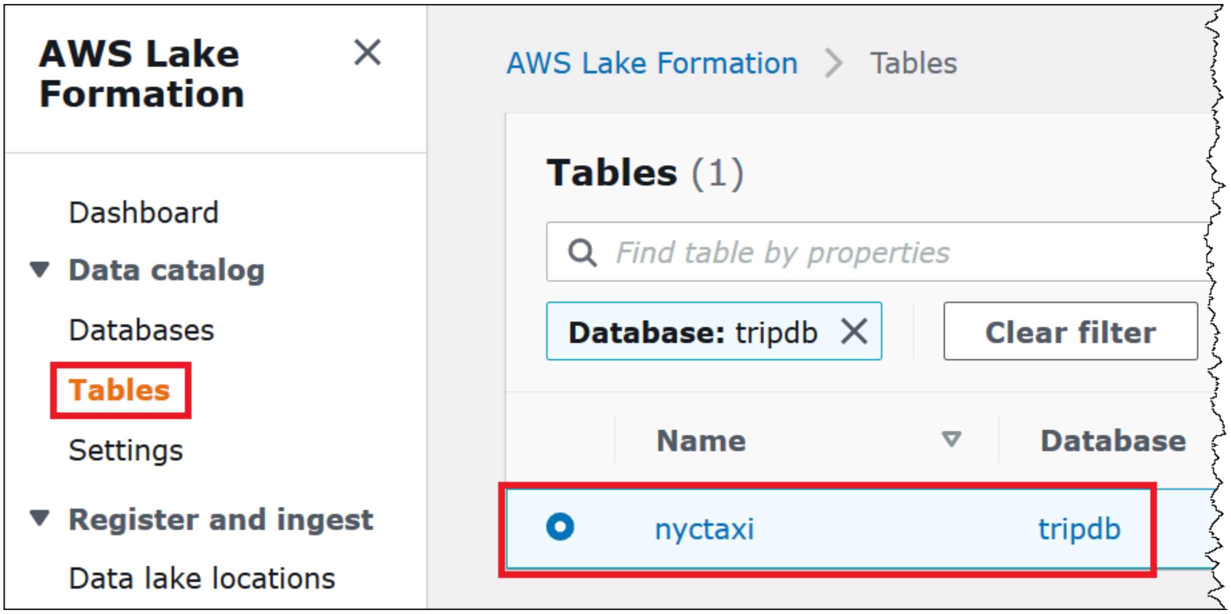
Step 6: Grant user and group permissions through AWS Lake Formation

In this step, you use the Lake Formation console to grant permissions on a table to the SAML user and group. You perform the following tasks:

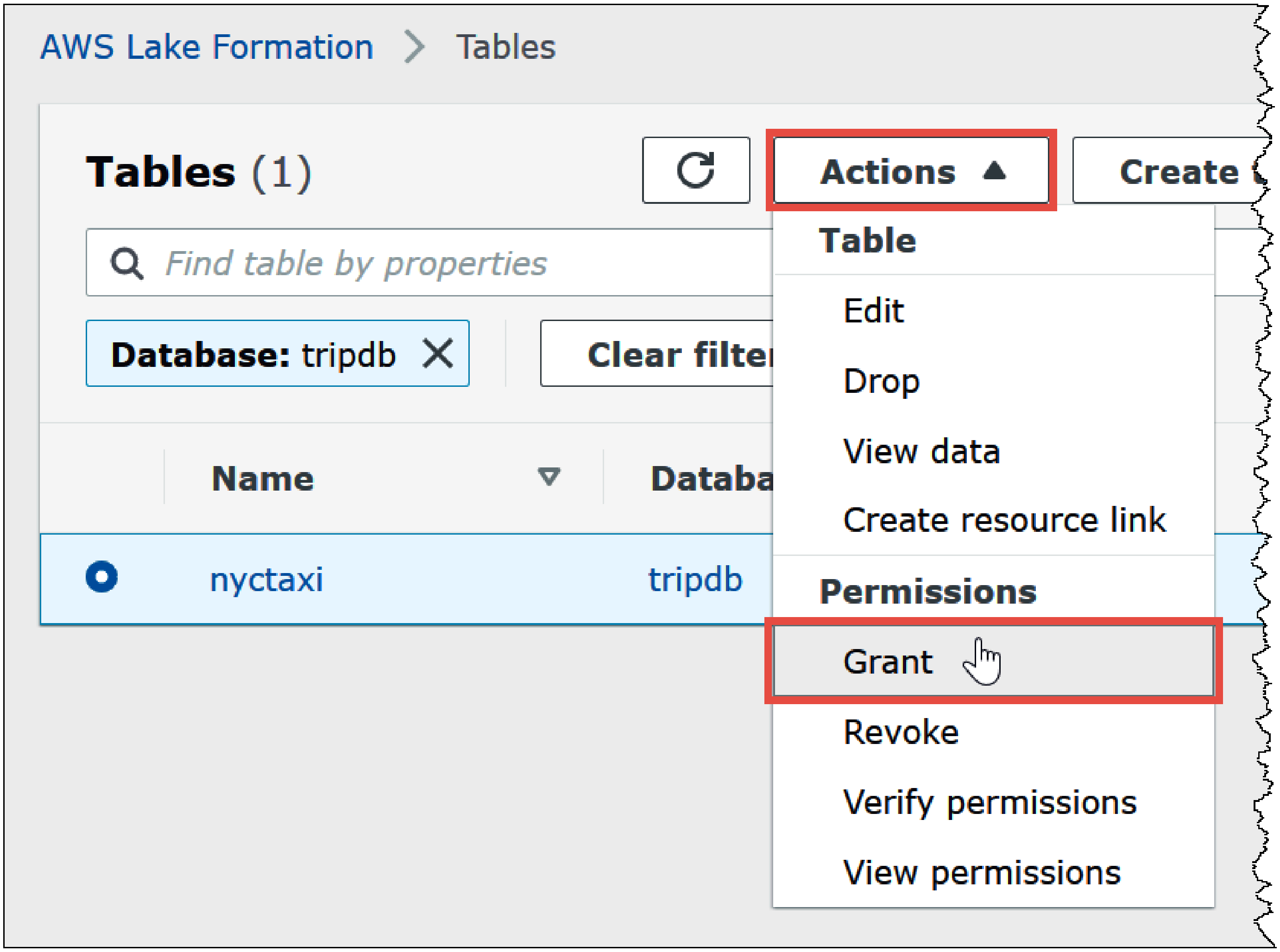
- Specify the ARN of the Okta SAML user and associated user permissions on the table.
- Specify the ARN of the Okta SAML group and associated group permissions on the table.
- Verify the permissions that you granted.

To grant permissions in Lake Formation for the Okta user

1. Sign in as data lake administrator to the AWS Management Console.
2. Open the Lake Formation console at https://console.aws.amazon.com/lakeformation/.
3. From the navigation pane, choose **Tables**, and then select the table that you want to grant permissions for. This tutorial uses the `nyctaxi` table from the `tripdb` database.



4. From **Actions**, choose **Grant**.



5. In the **Grant permissions** dialog, enter the following information:

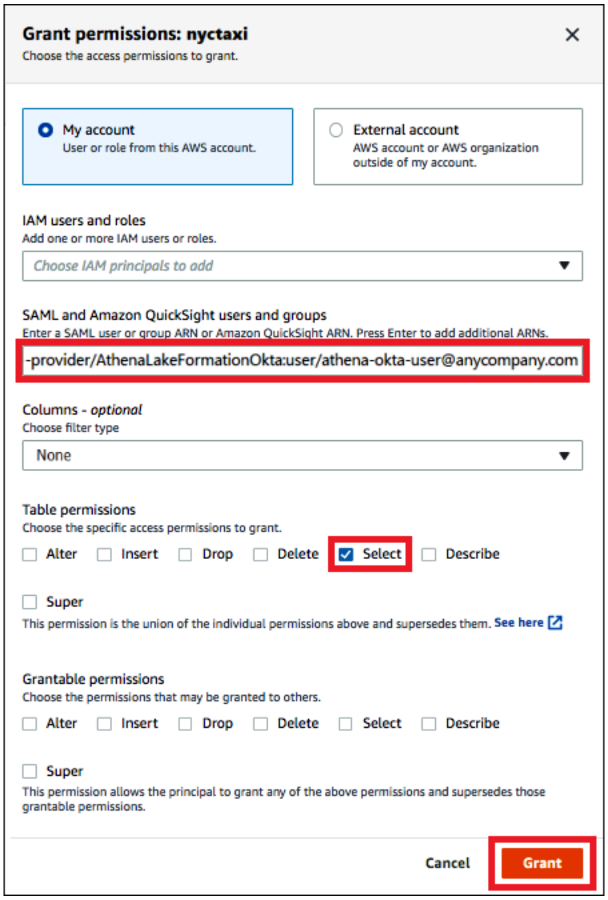
   a. Under **SAML and Amazon QuickSight users and groups**, enter the Okta SAML user ARN in the following format:

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:user/<athena-okta-user>@<anycompany.com>
   ```

   b. For **Columns**, for **Choose filter type**, and optionally choose **Include columns** or **Exclude columns**.

   c. Use the **Choose one or more columns** dropdown under the filter to specify the columns that you want to include or exclude for or from the user.

   d. For **Table permissions**, choose **Select**. This tutorial grants only the SELECT permission; your requirements may vary.



6. Choose **Grant**.

Now you perform similar steps for the Okta group.

To grant permissions in Lake Formation for the Okta group

1. On the **Tables** page of the Lake Formation console, make sure that the **nyctaxi** table is still selected.
2. From **Actions**, choose **Grant**.
3. In the **Grant permissions** dialog, enter the following information:

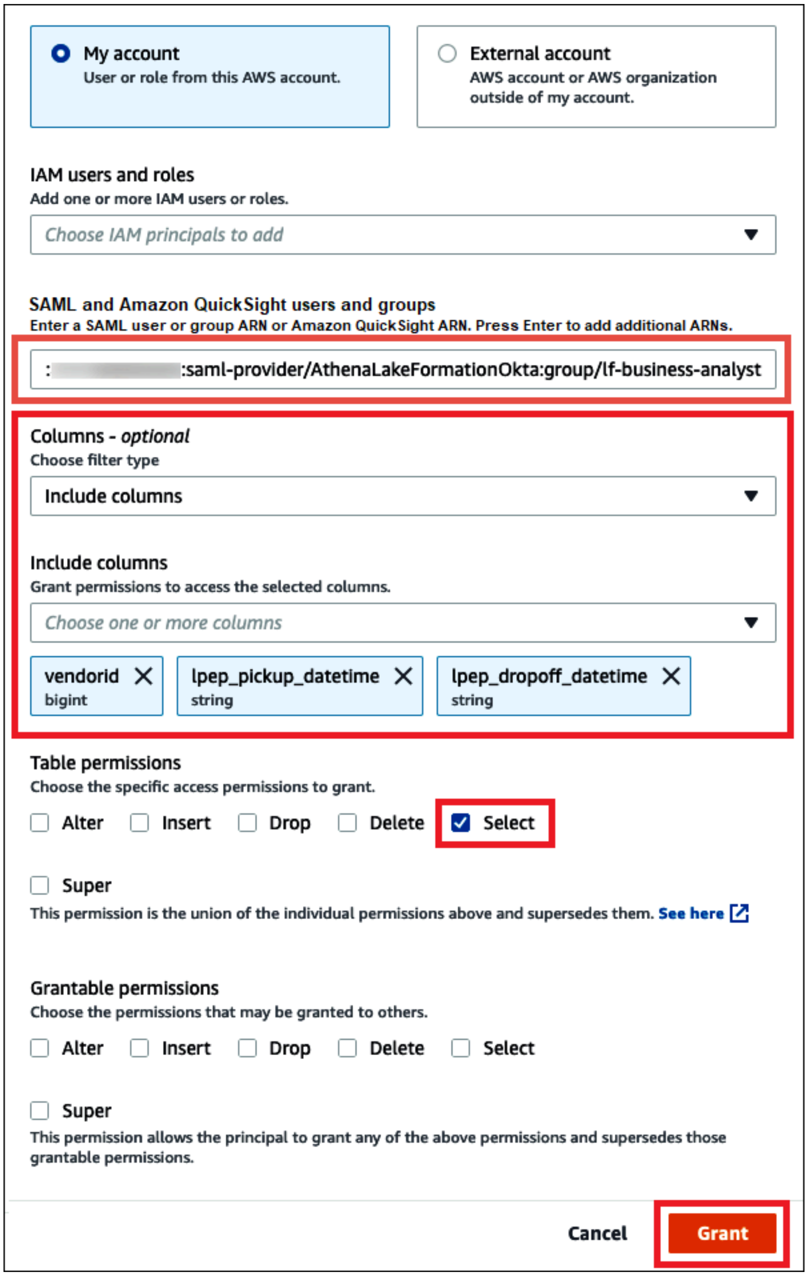
   a. Under **SAML and Amazon QuickSight users and groups**, enter the Okta SAML group ARN in the following format:

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:group/lf-
   business-analyst
   ```

   b. For **Columns**, **Choose filter type**, choose **Include columns**.

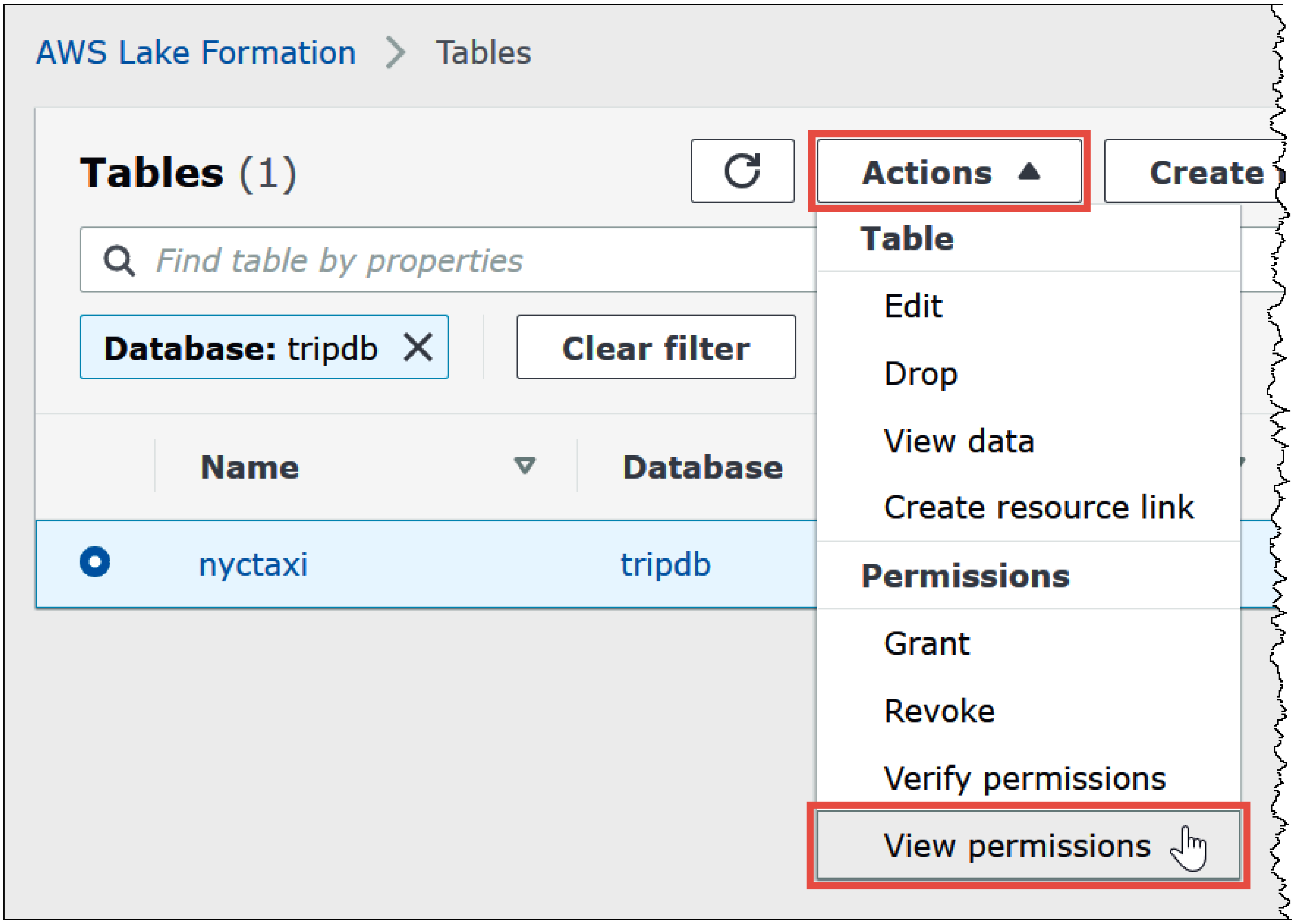
c. For **Choose one or more columns**, choose the first three columns of the table.

d. For **Table permissions**, choose the specific access permissions to grant. This tutorial grants only the SELECT permission; your requirements may vary.

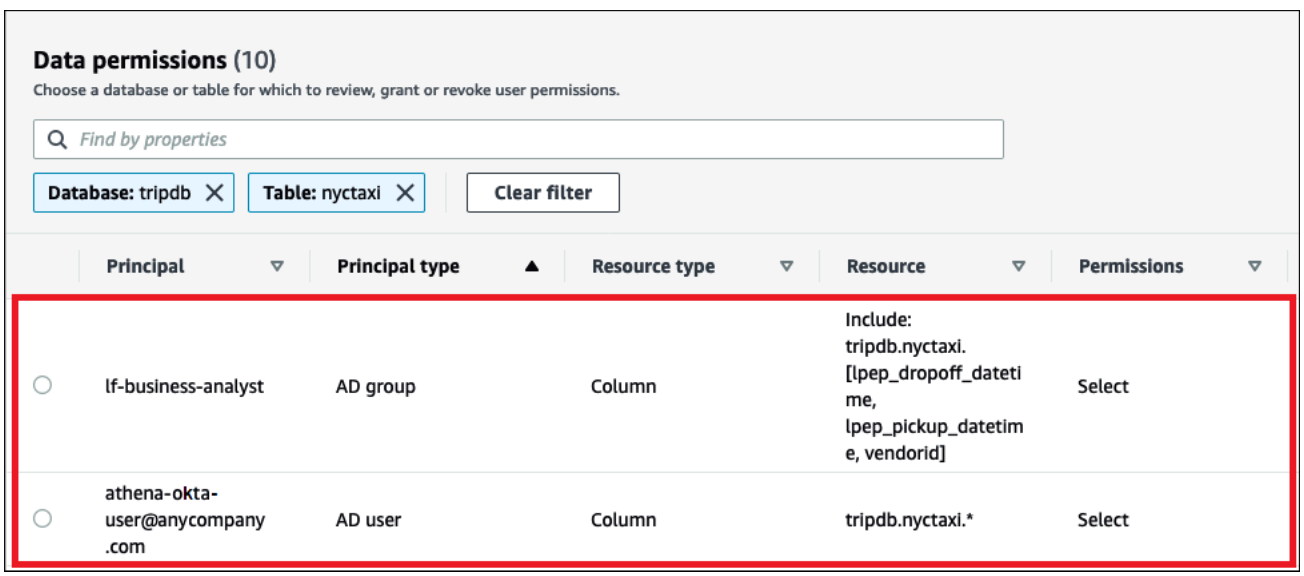


4. Choose **Grant**.

5. To verify the permissions that you granted, choose **Actions**, **View permissions**.



The **Data permissions** page for the `nyctaxi` table shows the permissions for **athena-okta-user** and the **lf-business-analyst** group.



Step 7: Verify access through the Athena JDBC client

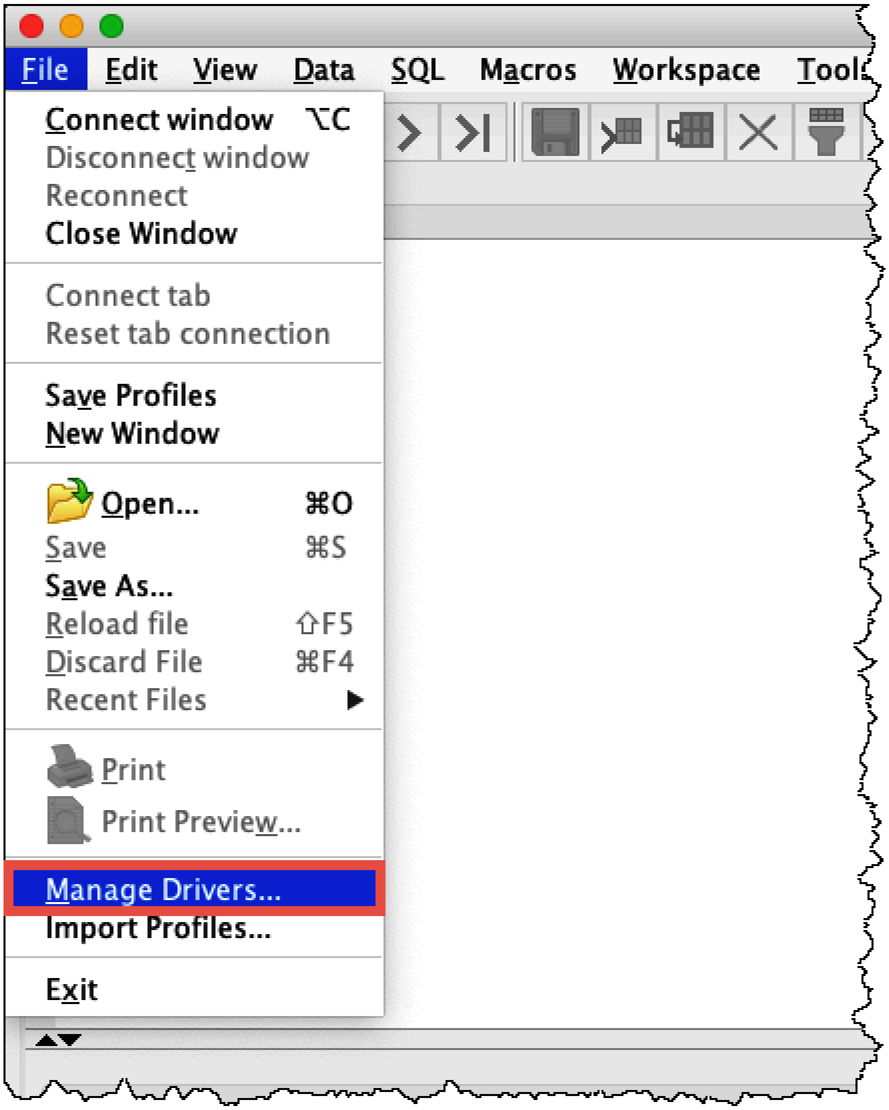
Now you are ready to use a JDBC client to perform a test connection to Athena as the Okta SAML user.

In this section, you perform the following tasks:

- Prepare the test client – Download the Athena JDBC driver, install SQL Workbench, and add the driver to Workbench. This tutorial uses SQL Workbench to access Athena through Okta authentication and to verify Lake Formation permissions.
- In SQL Workbench:
  - Create a connection for the Athena Okta user.
  - Run test queries as the Athena Okta user.
  - Create and test a connection for the business analyst user.
- In the Okta console, add the business analyst user to the developer group.
- In the Lake Formation console, configure table permissions for the developer group.
- In SQL Workbench, run test queries as the business analyst user and verify how the change in permissions affects the results.

To prepare the test client

1. Download and extract the Lake Formation compatible Athena JDBC driver (2.0.14 or later version) from [Connect to Amazon Athena with JDBC](#).
2. Download and install the free [SQL Workbench/J](#) SQL query tool, available under a modified Apache 2.0 license.
3. In SQL Workbench, choose **File**, and then choose **Manage Drivers**.



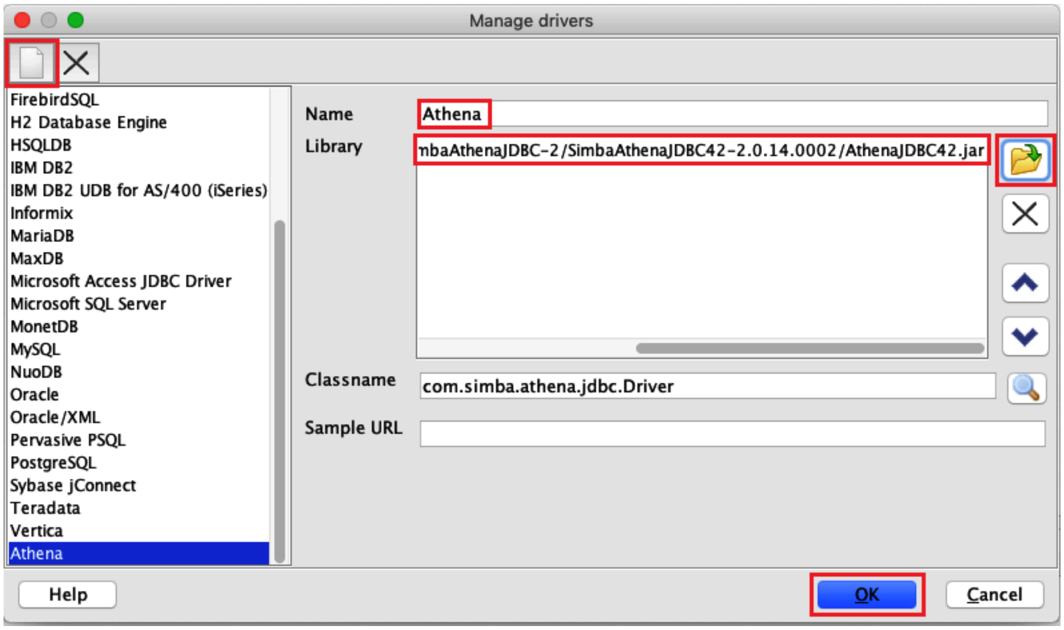
4. In the **Manage Drivers** dialog box, perform the following steps:

   a. Choose the new driver icon.

   b. For **Name**, enter **`Athena`**.

   c. For **Library**, browse to and choose the Simba Athena JDBC `.jar` file that you just downloaded.

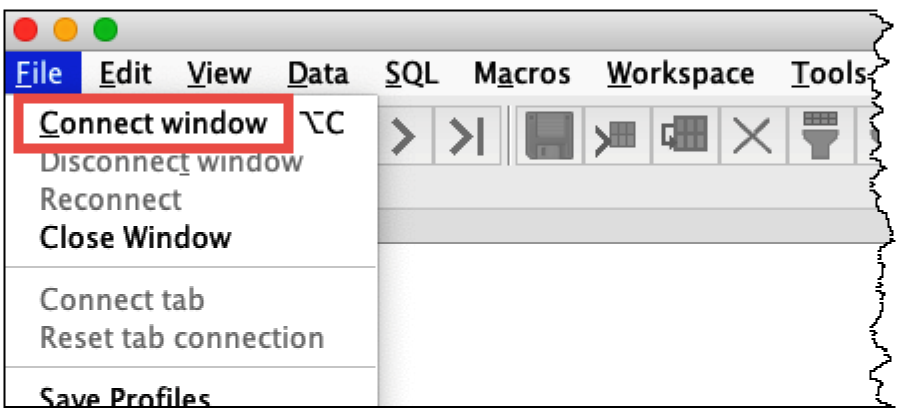
   d. Choose **OK**.



You are now ready to create and test a connection for the Athena Okta user.

To create a connection for the Okta user

1. Choose **File**, **Connect window**.



2. In the **Connection profile** dialog box, create a connection by entering the following information:

   - In the name box, enter **`Athena_Okta_User_Connection`**.
   - For **Driver**, choose the Simba Athena JDBC Driver.
   - For **URL**, do one of the following:
     - To use a connection URL, enter a single-line connection string. The following example adds line breaks for readability.

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://amzn-s3-demo-bucket/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-okta-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-app-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

- To use an AWS profile-based URL, perform the following steps:

  1. Configure an AWS profile that has an AWS credentials file like the following example.

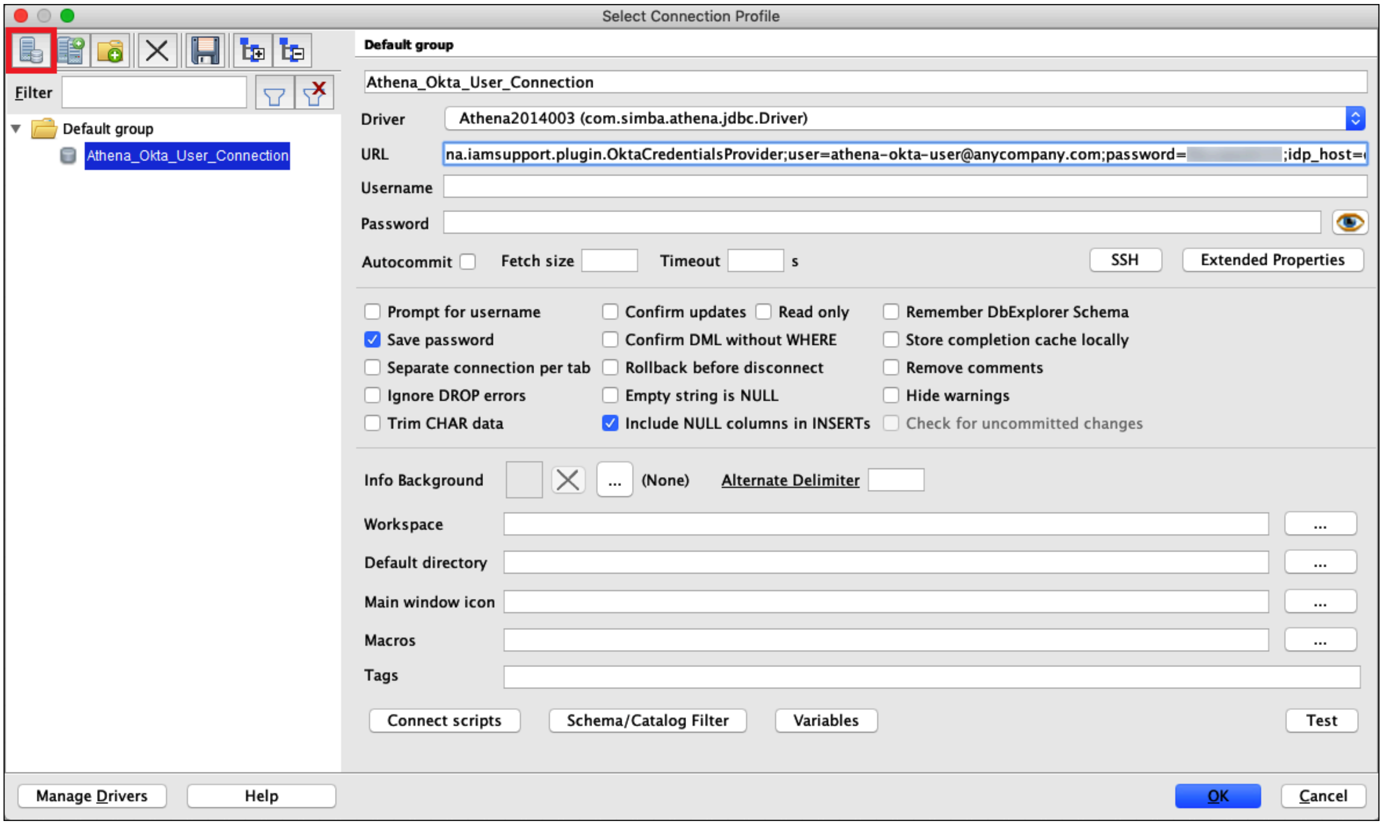
     ```
     [athena_lf_dev]
     plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
     idp_host=okta-idp-domain
     app_id=okta-app-id
     uid=athena-okta-user@anycompany.com
     pwd=password
     ```

  2. For **URL**, enter a single-line connection string like the following example. The example adds line breaks for readability.

     ```
     jdbc:awsathena://AwsRegion=region-id;
     S3OutputLocation=s3://amzn-s3-demo-bucket/athena_results;
     profile=athena_lf_dev;
     SSL_Insecure=true;
     LakeFormationEnabled=true;
     ```

Note that these examples are basic representations of the URL needed to connect to Athena. For the full list of parameters supported in the URL, refer to the JDBC documentation.

The following image shows a SQL Workbench connection profile that uses a connection URL.

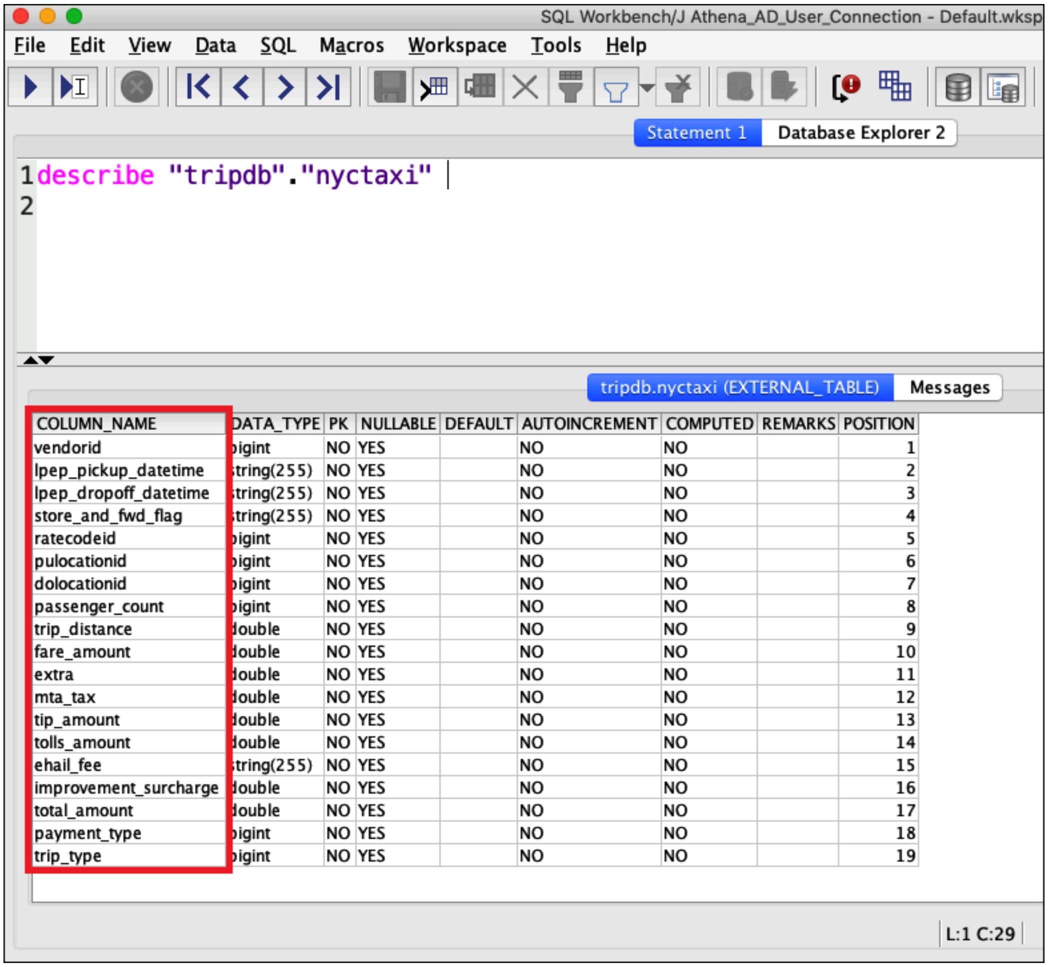


Now that you have established a connection for the Okta user, you can test it by retrieving some data.

To test the connection for the Okta user

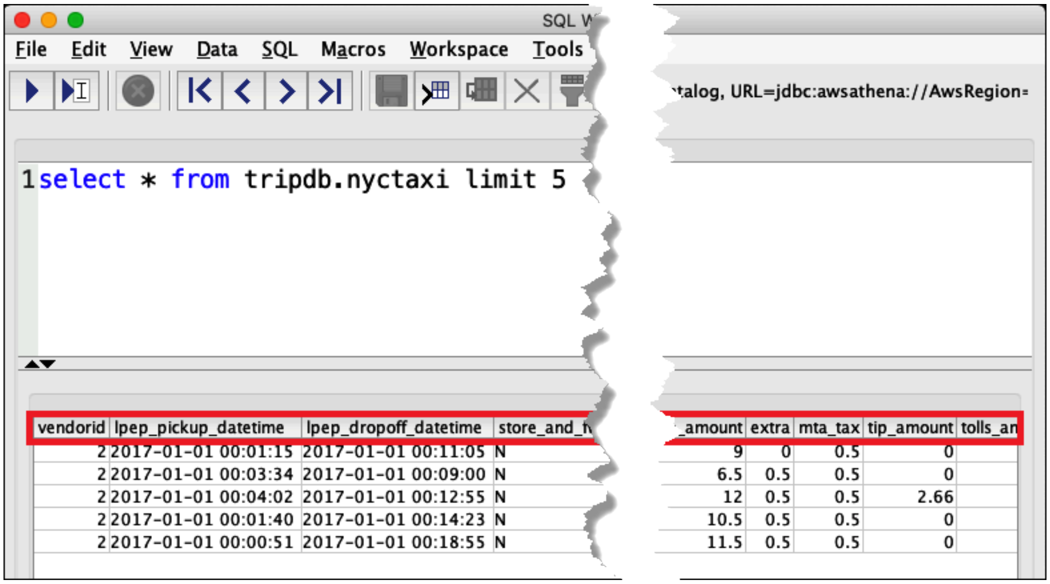
1. Choose **Test**, and then verify that the connection succeeds.
2. From the SQL Workbench **Statement** window, run the following SQL `DESCRIBE` command. Verify that all columns are displayed.

```
DESCRIBE "tripdb"."nyctaxi"
```



3. From the SQL Workbench **Statement** window, run the following SQL SELECT command. Verify that all columns are displayed.

```
SELECT * FROM tripdb.nyctaxi LIMIT 5
```



Next, you verify that the **athena-ba-user**, as a member of the **lf-business-analyst** group, has access to only the first three columns of the table that you specified earlier in Lake Formation.

To verify access for the athena-ba-user

1. In SQL Workbench, in the **Connection profile** dialog box, create another connection profile.

   - For the connection profile name, enter **`Athena_Okta_Group_Connection`**.
   - For **Driver**, choose the Simba Athena JDBC driver.
   - For **URL**, do one of the following:
     - To use a connection URL, enter a single-line connection string. The following example adds line breaks for readability.

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://amzn-s3-demo-bucket/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-ba-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-application-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

     - To use an AWS profile-based URL, perform the following steps:

1. Configure an AWS profile that has a credentials file like the following example.

```
[athena_lf_ba]
plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
idp_host=okta-idp-domain
app_id=okta-application-id
uid=athena-ba-user@anycompany.com
pwd=password
```

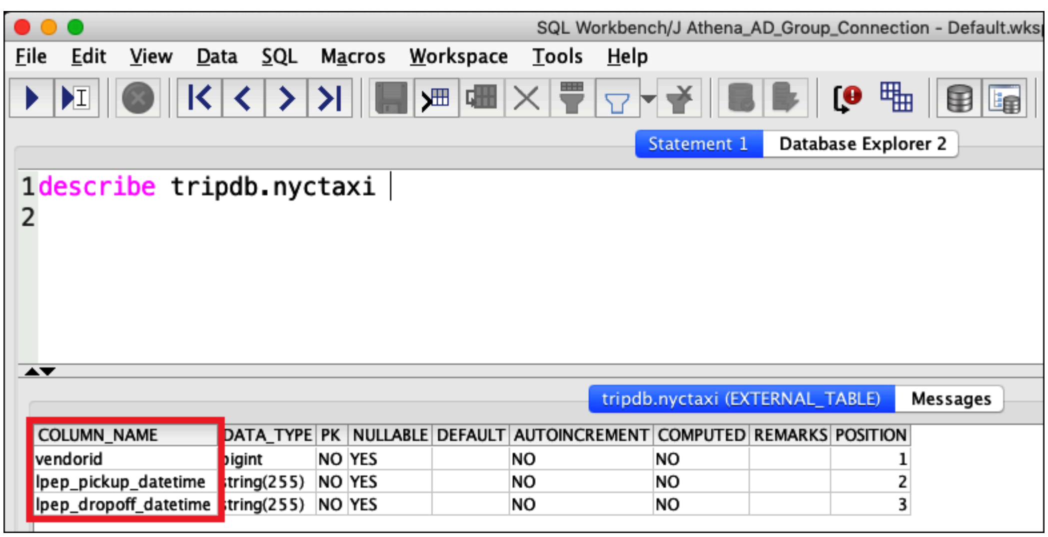
2. For **URL**, enter a single-line connection string like the following. The example adds line breaks for readability.

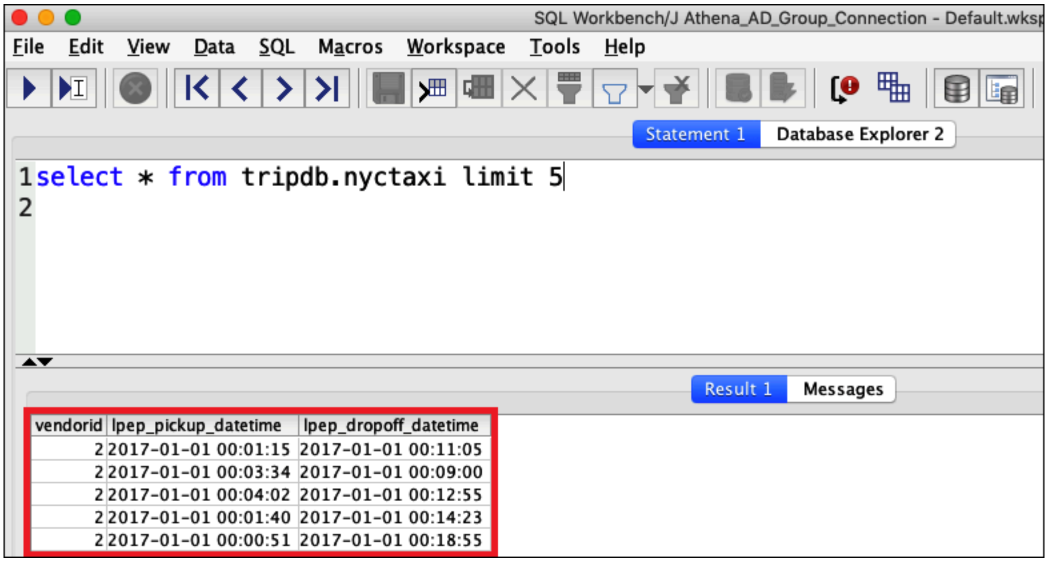
```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://amzn-s3-demo-bucket/athena_results;
profile=athena_lf_ba;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

2. Choose **Test** to confirm that the connection is successful.

3. From the **SQL Statement** window, run the same DESCRIBE and SELECT SQL commands that you did before and examine the results.

   Because **athena-ba-user** is a member of the **lf-business-analyst** group, only the first three columns that you specified in the Lake Formation console are returned.

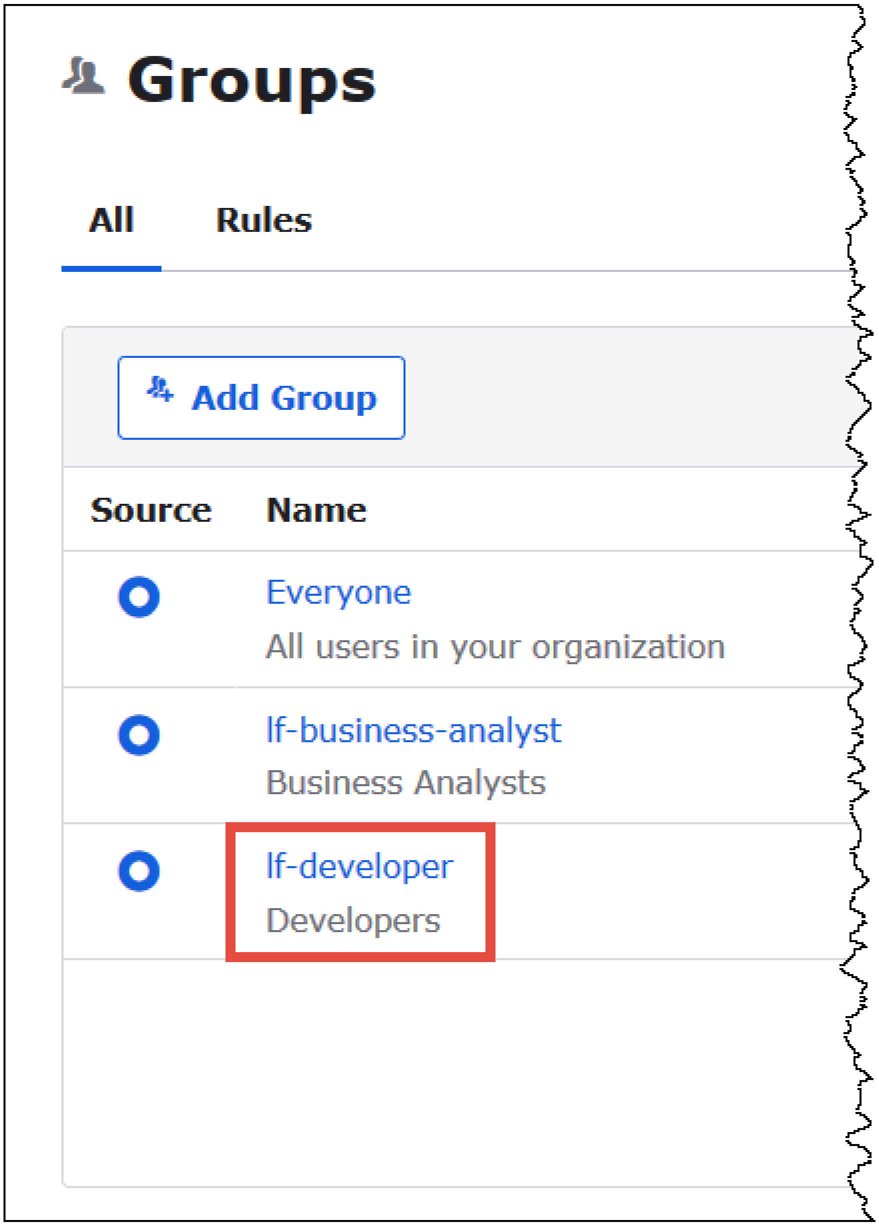




Next, you return to the Okta console to add the `athena-ba-user` to the `lf-developer` Okta group.

To add the athena-ba-user to the lf-developer group

1. Sign in to the Okta console as an administrative user of the assigned Okta domain.
2. Choose **Directory**, and then choose **Groups**.
3. On the Groups page, choose the **lf-developer** group.



4. Choose **Manage People**.
5. From the **Not Members** list, choose the **athena-ba-user** to add it to the **lf-developer group**.
6. Choose **Save**.

Now you return to the Lake Formation console to configure table permissions for the **lf-developer** group.

To configure table permissions for the lf-developer-group

1. Log into the Lake Formation console as Data Lake administrator.

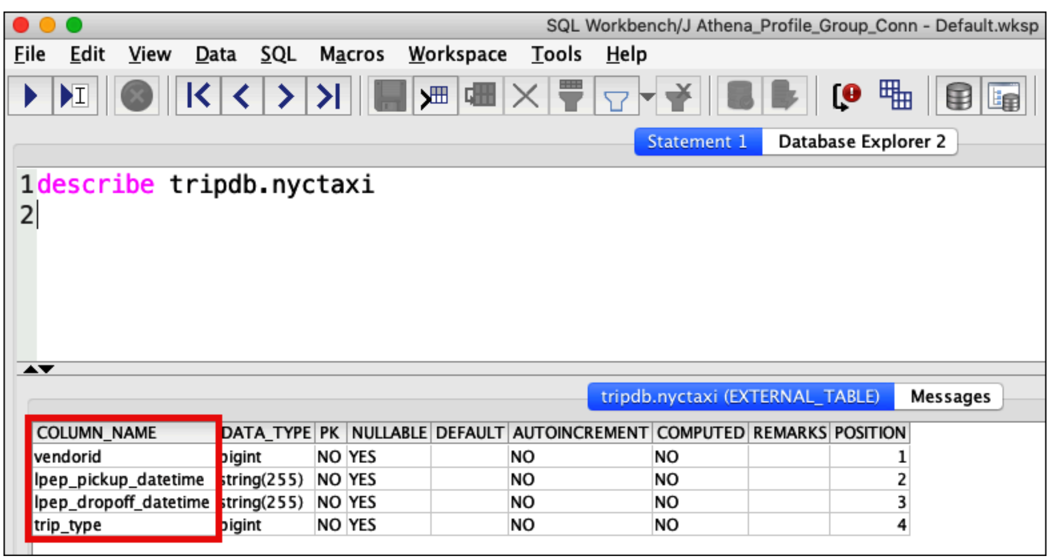
2. In the navigation pane, choose **Tables**.
3. Select the **nyctaxi** table.
4. Choose **Actions**, **Grant**.
5. In the **Grant Permissions** dialog, enter the following information:
   - For **SAML and Amazon QuickSight users and groups**, enter the Okta SAML lf-developer group ARN in the following format:
   - For **Columns**, **Choose filter type**, choose **Include columns**.
   - Choose the **trip_type** column.
   - For **Table permissions**, choose **SELECT**.
6. Choose **Grant**.

Now you can use SQL Workbench to verify the change in permissions for the **lf-developer** group. The change should be reflected in the data available to **athena-ba-user**, who is now a member of the **lf-developer** group.

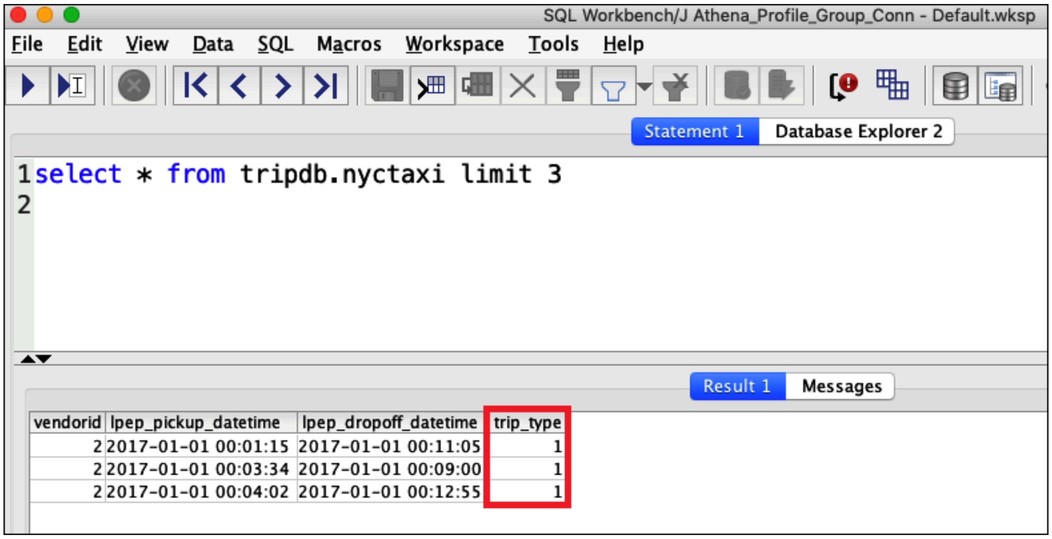
To verify the change in permissions for athena-ba-user

1. Close the SQL Workbench program, and then re-open it.
2. Connect to the profile for **athena-ba-user**.
3. From the **Statement** window, issue the same SQL statements that you ran previously:

   This time, the **trip_type** column is displayed.



Because **athena-ba-user** is now a member of both the **lf-developer** and **lf-business-analyst** groups, the combination of Lake Formation permissions for those groups determines the columns that are returned.



Conclusion

In this tutorial you configured Athena integration with AWS Lake Formation using Okta as the SAML provider. You used Lake Formation and IAM to control the resources that are available to the SAML user in your data lake AWS Glue Data Catalog.

Related resources

For related information, see the following resources.

- Connect to Amazon Athena with JDBC
- Enable federated access to the Athena API
- AWS Lake Formation Developer Guide
- Granting and revoking Data Catalog permissions in the *AWS Lake Formation Developer Guide*.
- Identity providers and federation in the *IAM User Guide*.
- Creating IAM SAML identity providers in the *IAM User Guide*.
- Enabling federation to AWS using Windows Active Directory, ADFS, and SAML 2.0 on the *AWS Security Blog*.

Workload management

You can use Athena's workgroup, capacity management, performance tuning, compression support, tags, and service quotas features to manage your workload.

Topics

- Use workgroups to control query access and costs
- Manage query processing capacity
- Optimize Athena performance
- Use compression in Athena
- Tag Athena resources
- Service Quotas

Use workgroups to control query access and costs

You can use Athena workgroups to separate workloads, control team access, enforce configuration, and track query metrics and control costs.

Separate your workloads

You can use workgroups to separate workloads. For example, you can create two independent workgroups, one for automated scheduled applications, such as report generation, and another for ad-hoc usage by analysts.

Control access by teams

Because workgroups act as IAM resources, you can use resource-level identity-based policies to control who can access a workgroup and run queries in it. To isolate queries for two different teams in your organization, you can create a separate workgroup for each team. Each workgroup has its own query history and a list of saved queries for the queries in that workgroup, and not for all queries in the account. For more information, see Use IAM policies to control workgroup access.

Enforce configuration

You can optionally enforce the same workgroup-wide settings for all queries that run in the workgroup. These settings include query results location in Amazon S3, expected bucket owner, encryption, and control of objects written to the query results bucket. For more information, see Override client-side settings.

Track query metrics, query events, and control costs

To track query metrics, query events, and control costs for each Athena workgroup, you can use the following features:

- **Publish query metrics** – Publish the query metrics for your workgroup to CloudWatch. In the Athena console, you can view query metrics for each workgroup. In CloudWatch, you can create custom dashboards, and set thresholds and alarms on these metrics. For more information, see Enable CloudWatch query metrics in Athena and Monitor Athena query metrics with CloudWatch.
- **Monitor Athena usage metrics** – See how your account uses resources by displaying your current service usage through CloudWatch graphs and dashboards. For more information, see Monitor Athena usage metrics with CloudWatch
- **Monitor query events** – Use Amazon EventBridge to receive real-time notifications regarding the state of your queries. For more information, see Monitor Athena query events with EventBridge.
- **Create data usage controls** – In Athena, you can configure per-query and per-workgroup data usage controls. Athena cancels queries when they exceed the specified threshold or activates an Amazon SNS alarm when a workgroup threshold is breached. For more information, see Configure per-query and per-workgroup data usage controls.
- **Use cost allocation tags** – Use the Billing and Cost Management console to tag workgroups with cost allocation tags. The costs associated with running queries in the workgroup appear in your Cost and Usage Reports with the corresponding cost allocation tag. For more information, see Using user-defined cost allocation tags in the *AWS Billing User Guide*.
- **Use capacity reservations** – You can create capacity reservations with the number of data processing units that you specify and add one or more workgroups to the reservation. For more information, see Manage query processing capacity.

For additional information about using Athena workgroups to separate workloads, control user access, and manage query usage and costs, see the AWS Big Data Blog post Separate queries and managing costs using Amazon Athena workgroups.

> **Note**
>
> Amazon Athena resources can now be accessed within Amazon SageMaker Unified Studio (Preview), which helps you access your organization's data and act on it with the best tools. You can migrate saved queries from an Athena workgroup to a SageMaker Unified

Studio project, configure projects with existing Athena workgroups, and maintain necessary permissions through IAM role updates. For more information, see Migrating Amazon Athena resources to Amazon SageMaker Unified Studio (Preview).

Considerations and limitations

When you use workgroups in Athena, keep in mind the following points:

- Each account has a primary workgroup. By default, if you have not created any workgroups, all queries in your account run in the primary workgroup. The primary workgroup cannot be deleted. The default permissions allow all authenticated users access to this workgroup.
- When you have access to a workgroup, you can view the workgroup's settings, metrics, and data usage control limits. With additional permissions, you can edit the settings and data usage control limits.
- When you run queries, they run in the current workgroup. You can run queries in the context of a workgroup in the console, through API operations, through the command line interface, or through a client application by using a JDBC or ODBC driver.
- In the Athena console query editor, you can open up to ten query tabs within each workgroup. When you switch between workgroups, your query tabs remain open for a maximum of three workgroups.
- You can create up to 1000 workgroups per AWS Region in your account.
- Workgroups can be disabled. Disabling a workgroup prevents queries from running in the workgroup until you re-enable the workgroup.
- Athena warns you if you attempt to delete a workgroup that contains saved queries. Before you delete a workgroup to which other users have access, make sure they have access to another workgroup that they can use to run queries.

Topics

Create a workgroup

Creating a workgroup requires permissions to `CreateWorkgroup` API actions. See Configure access to workgroups and tags and Use IAM policies to control workgroup access. If you are adding tags, you also need to add permissions to `TagResource`. See Tag policy examples for workgroups.

The following procedure shows how to use the Athena console to create a workgroup. To create a workgroup using the Athena API, see CreateWorkGroup.

To create a workgroup in the Athena console

1. Decide which workgroups to create. A few factors to consider include:
   - Who can run queries in each workgroup, and who owns workgroup configuration. Use IAM policies to enforce workgroup permissions. For more information, see Use IAM policies to control workgroup access.
   - The location in Amazon S3 to use for the query results for the workgroup. All users of the workgroup must have access to this location.
   - Whether the workgroup query results must be encrypted. Because encryption is per-workgroup (not per query), you should create separate workgroups for encrypted and non-encrypted query results. For more information, see Encrypt Athena query results stored in Amazon S3.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the Athena console navigation pane, choose **Workgroups**.
4. On the **Workgroups** page, choose **Create workgroup**.

5. On the **Create workgroup** page, fill in the fields as follows:

Field	Description
Workgroup name	Required. Enter a unique name for your workgroup. The name can contain from 1 to 128 characters, including alphanumeric characters, dashes, and underscores. After you create a workgroup , you cannot change its name.
Description	Optional. Enter a description for your workgroup. It can contain up to 1024 characters.
Choose the type of engine	Choose **Athena SQL** if you want to run ad-hoc SQL queries on data in Amazon S3 or use a prebuilt data source connector to run federated queries on a variety of data sources external to Amazon S3. You can run queries using the Athena query editor, AWS CLI, or Athena APIs. Choose **Apache Spark** if you want to create, edit, and run Jupyter notebook applications using Python and Apache Spark. Jupyter notebooks contain a list of cells that can include code, text, Markdown, mathematics, plots and rich media. Cells are run in order as calculations in an interactive notebook session in Athena. For information about creating and configuring a Spark-enabled workgroup, see Step 1: Create a Spark enabled workgroup in Athena. After you create a workgroup, its analytics engine can be upgraded (for example, from Athena engine version 2 to Athena engine version 3), but its engine type cannot be changed. For example, an Athena engine version 3 workgroup cannot be changed to a PySpark engine version 3 workgroup.
Update query engine	Choose how you want to update your workgroup when a new Athena engine version is released. You can let Athena decide when to update your workgroup or manually choose an engine version. For more information, see Athena engine versioning.

Field	Description
Authentication	Choose **AWS Identity and Access Management (IAM)** to use IAM authentication or federation for the workgroup. Choose **IAM Identity Center** if you want to support workforce identities such as users and groups from SAML 2.0 identity providers such as Microsoft Active Directory. For more information, see Use IAM Identity Center enabled Athena workgroups and Trusted identity propagation across applications in the *AWS IAM Identity Center User Guide*. You cannot change the type of authentication for the workgroup after the workgroup is created.
Service role for IAM Identity Center access	Athena requires IAM permissions to access IAM Identity Center on your behalf. For more information about IAM service roles, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
Location of query result	Enter a path to an Amazon S3 bucket or prefix. This bucket and prefix must exist before you can specify them. For information about creating an Amazon S3 bucket, see Create a bucket. **Note** If you don't specify a query results location for the workgroup or in **Settings**, the Athena query will fail. If you run queries with the API or the drivers, you must specify query results location in at least one of two places: for individual queries with OutputLocation, or for the workgroup, with WorkGroupConfiguration.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Expected bucket owner</td><td>Optional. Enter the ID of the AWS account that you expect to be the owner of the output location bucket. This is an added security measure. If the account ID of the bucket owner does not match the ID that you specify here, attempts to output to the bucket will fail. For in-depth information, see Verifying bucket ownership with bucket owner condition in the Amazon S3 User Guide.

Note
The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. It does not apply to other Amazon S3 locations like data source locations in external Amazon S3 buckets, <code>CTAS</code> and <code>INSERT INTO</code> destination table locations, <code>UNLOAD</code> statement output locations, operations to spill buckets for federated queries, or <code>SELECT</code> queries run against a table in another account.</td></tr>
</table>

Field	Description
Assign bucket owner full control over query results	This field is unselected by default. If you select it and ACLs are enabled for the query result location bucket, you grant full control access over query results to the bucket owner. For example, if your query result location is owned by another account, you can use this option to grant ownership and full control over your query results to the other account. If the bucket's S3 Object Ownership setting is **Bucket owner preferred**, the bucket owner also owns all query result objects written from this workgroup. For example, if an external account's workgroup enables this option and sets its query result location to your account's Amazon S3 bucket which has an S3 Object Ownership setting of **Bucket owner preferred**, you own and have full control access over the external workgroup's query results. Selecting this option when the query result bucket's S3 Object Ownership setting is **Bucket owner enforced** has no effect. For more information, see Object ownership settings in the *Amazon S3 User Guide*.
Encrypt query results	Optional. For all workgroup queries, encrypt the query results in Amazon S3. Because you must encrypt all queries in a workgroup or none, we recommend that you create separate workgroups for encrypted and non-encrypted queries. If selected, you can select the **Encryption type**, the **Encryption key** and enter the **KMS Key ARN**. If you don't have the key, open the AWS KMS console to create it. For more information, see Creating keys in the *AWS Key Management Service Developer Guide*.

Field	Description
Set *`encryption_type`* as minimum encryption	Optional. Select this option to enforce a minimum type of encryption for query results for all users of the workgroup. Selecting this option shows you a table with the hierarchy of encryption types. The table also shows you which encryption types workgroup users will be allowed to use when you specify a particular encryption type as the minimum. To use this option, the **Override client-side settings** must not be selected. For more information, see Configure minimum encryption for a workgroup.
Enable S3 Access Grants	This field is selected by default when you choose **IAM Identity Center** as the authentication mode. When selected, this option applies IAM Identity Center user or group based permissions to Amazon S3 locations.
Create user identity based S3 prefix	When this option is selected, Athena creates an Amazon S3 prefix when it stores query results. The prefix is based on the user's IAM Identity Center user identity.
Override client-side settings	This field is unselected by default. If you select it, workgroup settings apply to all queries in the workgroup and override client-side settings. For more information, see Override client-side settings.
Publish query metrics to CloudWatch	This field is selected by default. Publish query metrics to CloudWatch. See Monitor Athena query metrics with CloudWatch.
Requester Pays S3 buckets	Optional. Choose **Turn on queries on requester pays buckets in Amazon S3** if workgroup users will run queries on data stored in Amazon S3 buckets that are configured as Requester Pays. The account of the user running the query is charged for applicable data access and data transfer fees associated with the query. For more information, see Requester Pays buckets in the *Amazon Simple Storage Service User Guide*.

Field	Description
Per query data usage control	Optional. Sets the limit for the maximum amount of data a query is allowed to scan. You can set only one per query limit for a workgroup. The limit applies to all queries in the workgroup and if query exceeds the limit, it will be cancelled. For more information, see Configure per-query and per-workgroup data usage controls.
Workgroup data usage alerts	Optional. Set multiple alert thresholds when queries running in this workgroup scan a specified amount of data within a specific period. Alerts are implemented using Amazon CloudWatch alarms and applies to all queries in the workgroup. For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.
Tags	Optional. Add one or more tags to a workgroup. A tag is a label that you assign to an Athena workgroup resource. It consists of a key and a value. Use AWS tagging best practices to create a consistent set of tags and categorize workgroups by purpose, owner, or environment. You can also use tags in IAM policies, and to control billing costs. Do not use duplicate tag keys the same workgroup. For more information, see the section called “Tag resources”.

6. Choose **Create workgroup**. The workgroup appears in the list on the **Workgroups** page.

   In the query editor, Athena displays the current workgroup in the **Workgroup** option on the upper right of the console. You can use this option to switch workgroups. When you run queries, they run in the current workgroup.

7. Create IAM policies for your users, groups, or roles to enable their access to workgroups. The policies establish the workgroup membership and access to actions on a `workgroup` resource. For more information, see Use IAM policies to control workgroup access. For example JSON policies, see Configure access to workgroups and tags.

8. (Optional) Configure a minimal level of encryption in Amazon S3 for all query results from the workgroup when workgroup-wide encryption is not enforced by the override client-side settings option. You can use this feature to ensure that query results are never stored in an

Amazon S3 bucket in an unencrypted state. For more information, see Configure minimum encryption for a workgroup.

9. (Optional) Use Amazon CloudWatch and Amazon EventBridge to monitor your workgroup's queries and control costs. For more information, see Use CloudWatch and EventBridge to monitor queries and control costs.
10. (Optional) Use the Billing and Cost Management console to tag the workgroup with cost allocation tags. For more information, see Using user-defined cost allocation tags in the *AWS Billing User Guide*.
11. (Optional) To get dedicated processing capacity for the queries in the workgroup, add the workgroup to a capacity reservation. You can assign one or more workgroups to a reservation. For more information, see Manage query processing capacity.

Override client-side settings

When you create or edit a workgroup, you can choose the option **Override client-side settings**. This option is not selected by default. Depending on whether you select it, Athena does the following:

- If **Override client-side settings** is not selected, workgroup settings are not enforced at the client level. When the override client-side settings option is not selected for the workgroup, Athena uses the client's settings for all queries that run in the workgroup, including the settings for query results location, expected bucket owner, encryption, and control of objects written to the query results bucket. Each user can specify their own settings in the **Settings** menu on the console. If the client-side settings are not set, the workgroup-wide settings apply. If you use the AWS CLI, API actions, or JDBC and ODBC drivers to run queries in a workgroup that does not override client-side settings, your queries use the settings that you specify in your queries.
- If **Override client-side settings** is selected, workgroup settings are enforced at the workgroup level for all clients in the workgroup. When the override client-side settings option is selected for the workgroup, Athena uses the workgroup's settings for all queries that run in the workgroup, including the settings for query results location, expected bucket owner, encryption, and control of objects written to the query results bucket. Workgroup settings override any client-side settings that you specify for a query when you use the console, API actions, or JDBC or ODBC drivers. After workgroup settings are set to override client-side settings, client-side settings in the drivers or the API can be omitted.

If you override client-side settings, then the next time that you or any workgroup user opens the Athena console, Athena notifies you that queries in the workgroup use the workgroup's settings, and prompts you to acknowledge this change.

> **Note**
>
> Because overriding client-side settings can break custom automation that is based on the availability of results in an arbitrary Amazon S3 bucket, we recommend that you inform your users before overriding.

> **Important**
>
> If you use API actions, the AWS CLI, or the JDBC and ODBC drivers to run queries in a workgroup that overrides client-side settings, make sure that you either omit the client-side settings in your queries or update them to match the settings of the workgroup. If you specify client-side settings in your queries but run them in a workgroup that overrides the settings, the queries will run, but the workgroup settings will be used. For information about viewing the settings for a workgroup, see View the workgroup's details.

Manage workgroups

In the Athena console at https://console.aws.amazon.com/athena/, you can perform the following tasks:

Statement	Description
Create a workgroup	Create a new workgroup.
View the workgroup's details	View the workgroup's details, such as its name, description, data usage limits, location of query results, expected query results bucket owner, encryption, and control of objects written to the query results bucket. You can also verify whether this workgroup enforces its settings, if **Override client-side settings** is checked.

Statement	Description
Specify a workgroup for queries	Before you can run queries, you must specify to Athena which workgroup to use. You must have permissions to the workgroup.
Switch workgroups	Switch between workgroups to which you have access.
Edit a workgroup	Edit a workgroup and change its settings. You cannot change a workgroup's name, but you can create a new workgroup with the same settings and a different name.
Enable or disable a workgroup	Enable or disable a workgroup. When a workgroup is disabled, its users cannot run queries, or create new named queries. If you have access to it, you can still view metrics, data usage limit controls, workgroup's settings, query history, and saved queries.
Copy a saved query between workgroups	Copy a saved query between workgroups. You might want to do this if, for example, you created a query in a preview workgroup and you want to make it available in a nonpreview workgroup.
Delete a workgroup	Delete a workgroup. If you delete a workgroup, query history, saved queries, the workgroup's settings and per-query data limit controls are deleted. The workgroup-wide data limit controls remain in CloudWatch, and you can delete them individually. The primary workgroup cannot be deleted.
Use IAM policies to control workgroup access	Use IAM policies to control workgroup access. For example workgroup policies, see Example workgroup policies.
Create an Athena workgroup that uses IAM Identity Center authentication	To use IAM Identity Center identities with Athena, you must create an IAM Identity Center enabled workgroup. After you create the workgroup, you can use the IAM Identity Center console or API to assign IAM Identity Center users or groups to the workgroup.

Statement	Description
Configure minimum encryption for a workgroup	Enforce a minimal level of encryption in Amazon S3 for all query results from the workgroup. Use this feature to ensure that query results are never stored in an Amazon S3 bucket in an unencrypted state.

View the workgroup's details

For each workgroup, you can view its details. The details include the workgroup's name, description, whether it is enabled or disabled, and the settings used for queries that run in the workgroup, which include the location of the query results, expected bucket owner, encryption, and control of objects written to the query results bucket. If a workgroup has data usage limits, they are also displayed.

To view the workgroup's details

1. In the Athena console navigation pane, choose **Workgroups**.
2. On the **Workgroups** page, choose the link of the workgroup that you want to view. The **Overview Details** page for the workgroup displays.

Specify a workgroup for queries

To specify a workgroup to use, you must have permissions to the workgroup.

To specify the workgroup to use

1. Make sure your permissions allow you to run queries in a workgroup that you intend to use. For more information, see the section called "Use IAM policies to control workgroup access".
2. To specify the workgroup, use one of these options:

   - If you are using the Athena console, set the workgroup by switching workgroups.
   - If you are using the Athena API operations, specify the workgroup name in the API action. For example, you can set the workgroup name in StartQueryExecution, as follows:

```
StartQueryExecutionRequest startQueryExecutionRequest = new
 StartQueryExecutionRequest()
              .withQueryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
```

```
                .withQueryExecutionContext(queryExecutionContext)
                .withWorkGroup(WorkgroupName)
```

- If you are using the JDBC or ODBC driver, set the workgroup name in the connection string using the `Workgroup` configuration parameter. The driver passes the workgroup name to Athena. Specify the workgroup parameter in the connection string as in the following example:

```
jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
PWD=<SECRETKEY>;S3OutputLocation=s3://amzn-s3-demo-bucket/<athena-
output>-<AWSREGION>/;
Workgroup=<WORKGROUPNAME>;
```

Switch workgroups

You can switch from one workgroup to another if you have permissions to both of them.

You can open up to ten query tabs within each workgroup. When you switch between workgroups, your query tabs remain open for up to three workgroups.

To switch workgroups

1. In the Athena console, use the **Workgroup** option on the upper right to choose a workgroup.
2. If the **Workgroup** ***`workgroup-name`*** **settings** dialog box appears, choose **Acknowledge**.

The **Workgroup** option shows the name of the workgroup that you switched to. You can now run queries in this workgroup.

Edit a workgroup

Editing a workgroup requires permissions to `UpdateWorkgroup` API operations. See Configure access to workgroups and tags and Use IAM policies to control workgroup access. If you are adding or editing tags, you also need to have permissions to `TagResource`. See Tag policy examples for workgroups.

To edit a workgroup in the console

1. In the Athena console navigation pane, choose **Workgroups**.
2. On the **Workgroups** page, select the button for the workgroup that you want to edit.

3. Choose **Actions**, **Edit**.
4. Change the fields as needed. For the list of fields, see Create workgroup. You can change all fields except for the workgroup's name. If you need to change the name, create another workgroup with the new name and the same settings.
5. Choose **Save changes**. The updated workgroup appears in the list on the **Workgroups** page.

Enable or disable a workgroup

If you have permissions to do so, you can enable or disable workgroups in the console, by using the API operations, or with the JDBC and ODBC drivers.

To enable or disable a workgroup

1. In the Athena console navigation pane, choose **Workgroups**.
2. On the **Workgroups** page, choose the link for the workgroup.
3. On the upper right, choose **Enable workgroup** or **Disable workgroup**.
4. At the confirmation prompt, choose **Enable** or **Disable**. If you disable a workgroup, its users cannot run queries in it, or create new named queries. If you enable a workgroup, users can use it to run queries.

Copy a saved query between workgroups

Currently, the Athena console does not have an option to to copy a saved query from one workgroup to another directly, but you can perform the same task manually by using the following procedure.

To copy a saved query between workgroups

1. In the Athena console, from the workgroup that you want to copy the query from, choose the **Saved queries** tab.
2. Choose the link of the saved query that you want to copy. Athena opens the query in the query editor.
3. In the query editor, select the query text, and then press **`Ctrl+C`** to copy it.
4. Switch to the destination workgroup, or create a workgroup, and then switch to it.
5. Open a new tab in the query editor, and then press **`Ctrl+V`** to paste the text into the new tab.
6. In the query editor, choose **Save as** to save the query in the destination workgroup.

7. In the **Choose a name** dialog box, enter a name for the query and an optional description.
8. Choose **Save**.

Delete a workgroup

You can delete a workgroup if you have permissions to do so. The primary workgroup cannot be deleted.

If you have permissions, you can delete an empty workgroup at any time. You can also delete a workgroup that contains saved queries. In this case, before proceeding to delete a workgroup, Athena warns you that saved queries are deleted.

If you delete a workgroup while you are in it, the console switches focus to the primary workgroup. If you have access to it, you can run queries and view its settings.

If you delete a workgroup, its settings and per-query data limit controls are deleted. The workgroup-wide data limit controls remain in CloudWatch, and you can delete them there if needed.

> **Important**
>
> Before deleting a workgroup, ensure that its users also belong to other workgroups where they can continue to run queries. If the users' IAM policies allowed them to run queries *only* in this workgroup, and you delete it, they no longer have permissions to run queries. For more information, see Example policy for running queries in the primary workgroup.

To delete a workgroup in the console

1. In the Athena console navigation pane, choose **Workgroups**.
2. On the **Workgroups** page, select the button for the workgroup that you want to delete.
3. Choose **Actions**, **Delete**.
4. At the **Delete workgroup** confirmation prompt, enter the name of the workgroup, and then choose **Delete**.

To delete a workgroup with the API operation, use the `DeleteWorkGroup` action.

Use CloudWatch and EventBridge to monitor queries and control costs

Workgroups allow you to set data usage control limits per query or per workgroup, set up alarms when those limits are exceeded, and publish query metrics to CloudWatch.

In each workgroup, you can:

- Configure **Data usage controls** per query and per workgroup, and establish actions that will be taken if queries breach the thresholds.
- View and analyze query metrics, and publish them to CloudWatch. If you create a workgroup in the console, the setting for publishing the metrics to CloudWatch is selected for you. If you use the API operations, you must enable publishing the metrics. When metrics are published, they are displayed under the **Metrics** tab in the **Workgroups** panel. Metrics are disabled by default for the primary workgroup.

Video

The following video shows how to create custom dashboards and set alarms and triggers on metrics in CloudWatch. You can use pre-populated dashboards directly from the Athena console to consume these query metrics.

Monitoring Amazon Athena queries using Amazon CloudWatch

Topics

Enable CloudWatch query metrics in Athena

When you create a workgroup in the console, the setting for publishing query metrics to CloudWatch is selected by default.

To enable or disable query metrics in the Athena console for a workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Workgroups**.
4. Choose the link of the workgroup that you want to modify.
5. On the details page for the workgroup, choose **Edit**.
6. In the **Settings** section, select or clear **Publish query metrics to AWS CloudWatch**.

If you use API operations, the command line interface, or the client application with the JDBC driver to create workgroups, to enable publishing of query metrics, set `PublishCloudWatchMetricsEnabled` to `true` in WorkGroupConfiguration. The following example shows only the metrics configuration and omits other configuration:

```
"WorkGroupConfiguration": {
      "PublishCloudWatchMetricsEnabled": "true"
     ....
     }
```

Monitor Athena query metrics with CloudWatch

Athena publishes query-related metrics to Amazon CloudWatch, when the publish query metrics to CloudWatch option is selected. You can create custom dashboards, set alarms and triggers on metrics in CloudWatch, or use pre-populated dashboards directly from the Athena console.

When you enable query metrics for queries in workgroups, the metrics are displayed within the **Metrics** tab in the **Workgroups** panel, for each workgroup in the Athena console.

Athena publishes the following metrics to the CloudWatch console:

- `DPUAllocated` – The total number of DPUs (data processing units) provisioned in a capacity reservation to run queries.
- `DPUConsumed` – The number of DPUs actively consumed by queries in a `RUNNING` state at a given time in a reservation. Metric emitted only when workgroup is associated with a capacity reservation and includes all workgroups associated with a reservation.
- `DPUCount` – The maximum number of DPUs consumed by your query, published exactly once as the query completes.
- `EngineExecutionTime` – The number of milliseconds that the query took to run.
- `ProcessedBytes` – The number of bytes that Athena scanned per DML query.
- `QueryPlanningTime` – The number of milliseconds that Athena took to plan the query processing flow.
- `QueryQueueTime` – The number of milliseconds that the query was in the query queue waiting for resources.
- `ServicePreProcessingTime` – The number of milliseconds that Athena took to preprocess the query before submitting the query to the query engine.
- `ServiceProcessingTime` – The number of milliseconds that Athena took to process the query results after the query engine finished running the query.
- `TotalExecutionTime` – The number of milliseconds that Athena took to run a DDL or DML query.

For more complete descriptions, see the List of CloudWatch metrics and dimensions for Athena later in this document.

These metrics have the following dimensions:

- `CapacityReservation` – The name of the capacity reservation used to execute the query, if applicable.
- `QueryState` – `SUCCEEDED`, `FAILED`, or `CANCELED`
- `QueryType` – `DML`, `DDL`, or `UTILITY`
- `WorkGroup` – name of the workgroup

Athena publishes the following metric to the CloudWatch console under the `AmazonAthenaForApacheSpark` namespace:

- `DPUCount` – number of DPUs consumed during the session to execute the calculations.

This metric has the following dimensions:

- `SessionId` – The ID of the session in which the calculations are submitted.
- `WorkGroup` – Name of the workgroup.

For more information, see the List of CloudWatch metrics and dimensions for Athena later in this topic. For information about Athena usage metrics, see Monitor Athena usage metrics with CloudWatch.

You can view query metrics in the Athena console or in the CloudWatch console.

View query metrics in the Athena console

To view query metrics for a workgroup in the Athena console

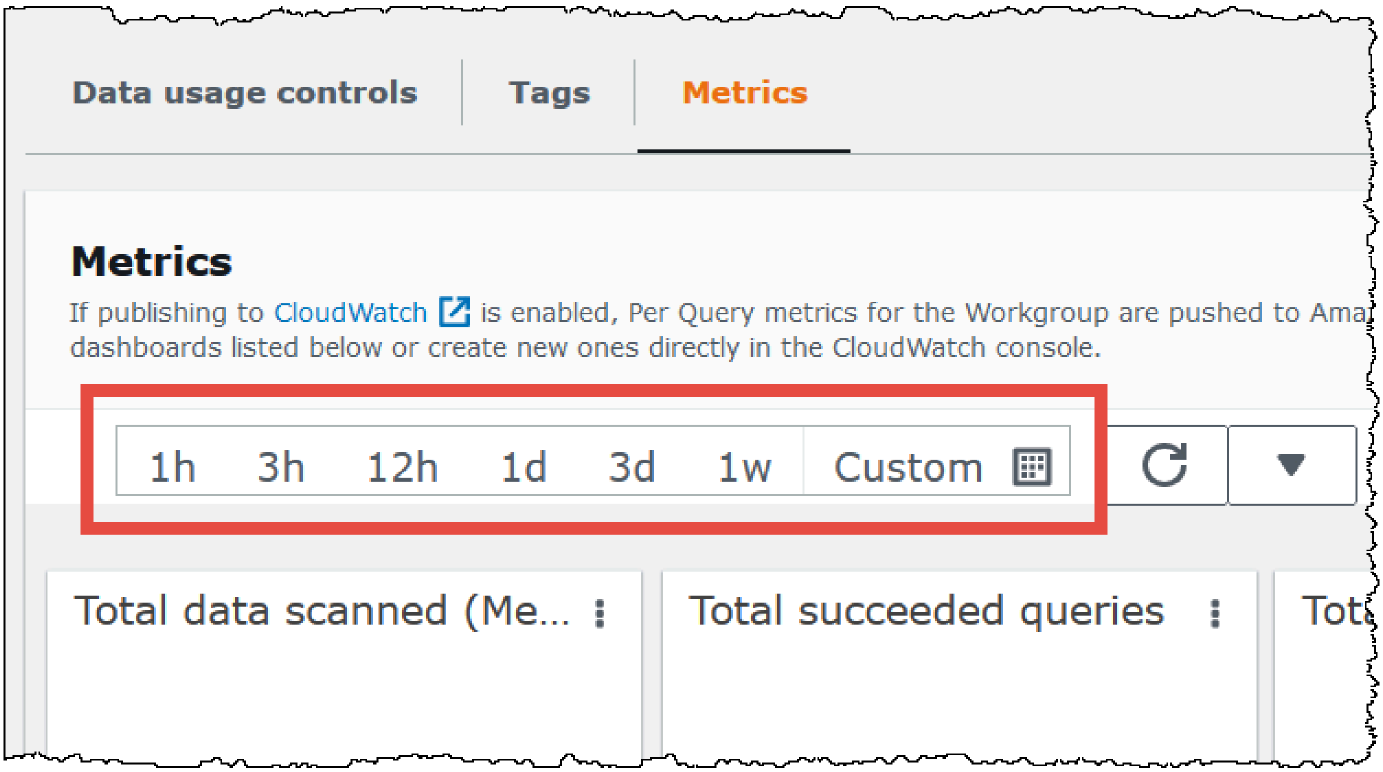
1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Workgroups**.
4. Choose the workgroup that you want from the list, and then choose the **Metrics** tab.

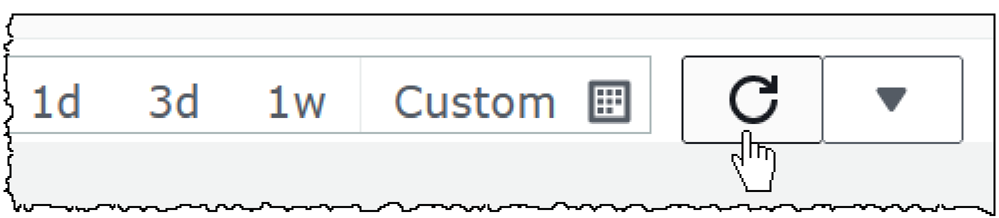
The metrics dashboard displays.

> **Note**
>
> If you just recently enabled metrics for the workgroup and/or there has been no recent query activity, the graphs on the dashboard may be empty. Query activity is retrieved from CloudWatch depending on the interval that you specify in the next step.

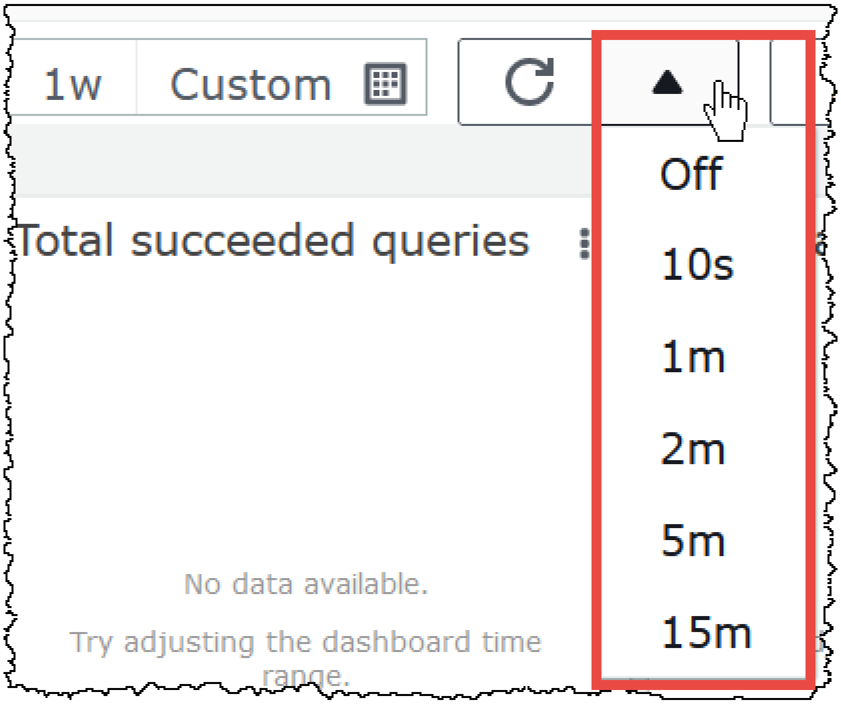
5. In the **Metrics** section, choose the metrics interval that Athena should use to fetch the query metrics from CloudWatch, or specify a custom interval.



6. To refresh the displayed metrics, choose the refresh icon.



7. Click the arrow next to the refresh icon to choose how frequently you want the metrics display to be updated.



View query metrics in the CloudWatch console

To view metrics in the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select the **AWS/Athena** namespace.

View query metrics with the AWS CLI

To view metrics with the AWS CLI

- Do one of the following:
  - To list the metrics for Athena, open a command prompt, and use the following command:

    ```
    aws cloudwatch list-metrics --namespace "AWS/Athena"
    ```

  - To list all available metrics, use the following command:

    ```
    aws cloudwatch list-metrics"
    ```

List of CloudWatch metrics and dimensions for Athena

If you've enabled CloudWatch metrics in Athena, it sends the following metrics to CloudWatch per workgroup. The following metrics use the `AWS/Athena` namespace.

Metric name	Description
DPUAllocated	The total number of DPUs (data processing units) provisioned in a capacity reservation to run queries.
DPUConsumed	The number of DPUs actively consumed by queries in a `RUNNING` state at a given time in a reservation. This metric is emitted only when workgroup is associated with a capacity reservation and includes all workgroups associated with a reservation. If you move a workgroup from one reservation to another, the metric includes data from the time when the workgroup belonged to the first reservation. For more information about capacity reservations, see [Manage query processing capacity]().
DPUCount	The maximum number of DPUs consumed by your query, published exactly once as the query completes. This metric is emitted only for workgroups that are attached to a capacity reservation.
EngineExecutionTime	The number of milliseconds that the query took to run.
ProcessedBytes	The number of bytes that Athena scanned per DML query. For queries that were canceled (either by the users, or automatically, if they reached the limit), this includes the amount of data scanned before the cancellation time. This metric is not reported for DDL queries.
QueryPlanningTime	The number of milliseconds that Athena took to plan the query processing flow. This includes the time spent retrieving table partitions from the data source. Note that because the query engine performs the query planning, query planning time is a subset of EngineExecutionTime.

Metric name	Description
QueryQueueTime	The number of milliseconds that the query was in the query queue waiting for resources. Note that if transient errors occur, the query can be automatically added back to the queue.
ServicePreProcessingTime	The number of milliseconds that Athena took to preprocess the query before submitting the query to the query engine.
ServiceProcessingTime	The number of milliseconds that Athena took to process the query results after the query engine finished running the query.
TotalExecutionTime	The number of milliseconds that Athena took to run a DDL or DML query. TotalExecutionTime includes QueryQueueTime, QueryPlanningTime, EngineExecutionTime, and ServicePr ocessingTime.

These metrics for Athena have the following dimensions.

Dimension	Description
CapacityReservation	The name of the capacity reservation that was used to execute the query, if applicable. When a capacity reservation is not used, this dimension returns no data.
QueryState	The query state. Valid statistics: SUCCEEDED, FAILED, or CANCELED.
QueryType	The query type. Valid statistics: `DDL`, `DML`, or `UTILITY`. The type of query statement that was run. `DDL` indicates DDL (Data Definitio n Language) query statements. `DML` indicates DML (Data Manipulation Language) query statements, such as `CREATE TABLE AS SELECT`. `UTILITY` indicates query statements other than DDL and DML, such as `SHOW CREATE TABLE`, or `DESCRIBE TABLE`.

Dimension	Description
WorkGroup	The name of the workgroup.

Monitor Athena usage metrics with CloudWatch

You can use CloudWatch usage metrics to provide visibility into your how your account uses resources by displaying your current service usage on CloudWatch graphs and dashboards.

For Athena, usage availability metrics correspond to AWS service quotas for Athena. You can configure alarms that alert you when your usage approaches a service quota. For more information about Athena service quotas, see Service Quotas. For more information about AWS usage metrics, see AWS usage metrics in the *Amazon CloudWatch User Guide*.

Athena publishes the following metrics in the `AWS/Usage` namespace.

Metric name	Description
`ResourceCount`	The sum of all queued and executing queries per AWS Region per account, separated by query type (DML or DDL). Maximum is the only useful statistic for this metric. This metric publishes periodically every minute. If you are not running any queries, the metric reports nothing (not even 0). The metric publishes only if active queries are running at the time the metric is taken.

The following dimensions are used to refine the usage metrics that are published by Athena.

Dimension	Description
`Service`	The name of the AWS service containing the resource. For Athena, the value for this dimension is `Athena`.
`Resource`	The type of resource that is running. The resource value for Athena query usage is `ActiveQueryCount` .

Dimension	Description
`Type`	The type of entity that's being reported. Currently, the only valid value for Athena usage metrics is `Resource`.
`Class`	The class of resource being tracked. For Athena, `Class` can be `DML` or `DDL`.

View Athena resource usage metrics in the CloudWatch console

You can use the CloudWatch console to see a graph of Athena usage metrics and configure alarms that alert you when your usage approaches a service quota.

To view Athena resource usage metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose **Usage**, and then choose **By AWS Resource**.

   The list of service quota usage metrics appears.
4. Select the check box that is next to **Athena** and **ActiveQueryCount**.
5. Choose the **Graphed metrics** tab.

   The graph above displays your current usage of the AWS resource.

For information about adding service quotas to the graph and setting an alarm that notifies you if you approach the service quota, see Visualizing your service quotas and setting alarms in the *Amazon CloudWatch User Guide*. For information about setting usage limits per workgroup, see Configure per-query and per-workgroup data usage controls.

Monitor Athena query events with EventBridge

You can use Amazon Athena with Amazon EventBridge to receive real-time notifications regarding the state of your queries. When a query you have submitted transitions states, Athena publishes an event to EventBridge containing information about that query state transition. You can write simple rules for events that are of interest to you and take automated actions when an event matches a rule. For example, you can create a rule that invokes an AWS Lambda function when a query reaches a terminal state. Events are emitted on a best effort basis.

Before you create event rules for Athena, you should do the following:

- Familiarize yourself with events, rules, and targets in EventBridge. For more information, see What Is Amazon EventBridge? For more information about how to set up rules, see Getting started with Amazon EventBridge.
- Create the target or targets to use in your event rules.

> **Note**
>
> Athena currently offers one type of event, Athena Query State Change, but may add other event types and details. If you are programmatically deserializing event JSON data, make sure that your application is prepared to handle unknown properties if additional properties are added.

Athena event format

The following is the basic pattern for an Amazon Athena event.

```
{
    "source":[
        "aws.athena"
    ],
    "detail-type":[
        "Athena Query State Change"
    ],
    "detail":{
        "currentState":[
            "SUCCEEDED"
        ]
    }
}
```

Athena query state change event

The following example shows an Athena Query State Change event with the `currentState` value of `SUCCEEDED`.

```
{
    "version":"0",
```

```
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[

    ],
    "detail":{
        "versionId":"0",
        "currentState":"SUCCEEDED",
        "previousState":"RUNNING",
        "statementType":"DDL",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

The following example shows an Athena Query State Change event with the `currentState` value of `FAILED`. The `athenaError` block appears only when `currentState` is `FAILED`. For information about the values for `errorCategory` and `errorType`, see [Athena error catalog](#).

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[
    ],
    "detail":{
        "athenaError": {
            "errorCategory": 2.0, //Value depends on nature of exception
            "errorType": 1306.0, //Type depends on nature of exception
            "errorMessage": "Amazon S3 bucket not found", //Message depends on nature
 of exception
            "retryable":false //Retryable value depends on nature of exception
        },
        "versionId":"0",
```

```
        "currentState": "FAILED",
        "previousState": "RUNNING",
        "statementType":"DML",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

Output properties

The JSON output includes the following properties.

Property	Description
`athenaError`	Appears only when `currentState` is `FAILED`. Contains information about the error that occurred, including the error category, error type, error message, and whether the action that led to the error can be retried. Values for each of these fields depend on the nature of the error. For information about the values for `errorCategory` and `errorType`, see Athena error catalog.
`versionId`	The version number for the detail object's schema.
`currentState`	The state that the query transitioned to at the time of the event.
`previousState`	The state that the query transitioned from at the time of the event.
`statementType`	The type of query statement that was run.
`queryExecutionId`	The unique identifier for the query that ran.
`workgroupName`	The name of the workgroup in which the query ran.
`sequenceNumber`	A monotonically increasing number that allows for deduplication and ordering of incoming events that involve a single query that ran. When duplicate events are published for the same state transition, the `sequenceNumber` value is the same. When a query experiences a state transition more than once, such as queries that experience

Property	Description
	rare requeuing, you can use `sequenceNumber` to order events with identical `currentState` and `previousState` values.

Example

The following example publishes events to an Amazon SNS topic to which you have subscribed. When Athena is queried, you receive an email. The example assumes that the Amazon SNS topic exists and that you have subscribed to it.

To publish Athena events to an Amazon SNS topic

1. Create the target for your Amazon SNS topic. Give the EventBridge events Service Principal `events.amazonaws.com` permission to publish to your Amazon SNS topic, as in the following example.

```
{
    "Effect":"Allow",
    "Principal":{
        "Service":"events.amazonaws.com"
    },
    "Action":"sns:Publish",
    "Resource":"arn:aws:sns:us-east-1:111111111111:your-sns-topic"
}
```

2. Use the AWS CLI `events put-rule` command to create a rule for Athena events, as in the following example.

```
aws events put-rule --name {ruleName} --event-pattern '{"source": ["aws.athena"]}'
```

3. Use the AWS CLI `events put-targets` command to attach the Amazon SNS topic target to the rule, as in the following example.

```
aws events put-targets --rule {ruleName} --targets Id=1,Arn=arn:aws:sns:us-
east-1:111111111111:your-sns-topic
```

4. Query Athena and observe the target being invoked. You should receive corresponding emails from the Amazon SNS topic.

Use AWS User Notifications with Amazon Athena

You can use AWS User Notifications to set up delivery channels to get notified about Amazon Athena events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, Amazon Q Developer in chat applications chat notifications, or AWS Console Mobile Application push notifications. You can also see notifications in the Console Notifications Center. User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

For more information, see the *AWS User Notifications User Guide*.

Configure per-query and per-workgroup data usage controls

Athena allows you to set two types of cost controls: per-query limit and per-workgroup limit. For each workgroup, you can set only one per-query limit and multiple per-workgroup limits.

- The **per-query control limit** specifies the total amount of data scanned per query. If any query that runs in the workgroup exceeds the limit, it is canceled. You can create only one per-query control limit in a workgroup and it applies to each query that runs in it. Edit the limit if you need to change it. For detailed steps, see To create a per-query data usage control.
- The **workgroup-wide data usage control limit** specifies the total amount of data scanned for all queries that run in this workgroup during the specified time period. You can create multiple limits per workgroup. The workgroup-wide query limit allows you to set multiple thresholds on hourly or daily aggregates on data scanned by queries running in the workgroup.

  If the aggregate amount of data scanned exceeds the threshold, you can push a notification to an Amazon SNS topic. To do this, you configure an Amazon SNS alarm and an action in the Athena console to notify an administrator when the limit is breached. For detailed steps, see To create a per-workgroup data usage control. You can also create an alarm and an action on any metric that Athena publishes from the CloudWatch console. For example, you can set an alert on a number of failed queries. This alert can trigger an email to an administrator if the number crosses a certain threshold. If the limit is exceeded, an action sends an Amazon SNS alarm notification to the specified users.

  Other actions you can take:

  - Invoke a Lambda function. For more information, see Invoking Lambda functions using Amazon SNS notifications in the *Amazon Simple Notification Service Developer Guide*.
  - Disable the workgroup to stop any further queries from running. For steps, see Enable or disable a workgroup.

The per-query and per-workgroup limits are independent of each other. A specified action is taken whenever either limit is exceeded. If two or more users run queries at the same time in the same workgroup, it is possible that each query does not exceed any of the specified limits, but the total sum of data scanned exceeds the data usage limit per workgroup. In this case, an Amazon SNS alarm is sent to the user.

Create a per-query data usage control

To create a per-query data usage control

The per-query control limit specifies the total amount of data scanned per query. If any query that runs in the workgroup exceeds the limit, it is canceled. Canceled queries are charged according to [Amazon Athena pricing](#).

> **Note**
>
> In the case of canceled or failed queries, Athena may have already written partial results to Amazon S3. In such cases, Athena does not delete partial results from the Amazon S3 prefix where results are stored. You must remove the Amazon S3 prefix with partial results. Athena uses Amazon S3 multipart uploads to write data Amazon S3. We recommend that you set the bucket lifecycle policy to end multipart uploads in cases when queries fail. For more information, see [Aborting incomplete multipart uploads using a bucket lifecycle policy](#) in the *Amazon Simple Storage Service User Guide*.

You can create only one per-query control limit in a workgroup and it applies to each query that runs in it. Edit the limit if you need to change it.

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Workgroups**.
4. Choose the name of the workgroup from the list.
5. On the **Data usage controls** tab, in the **Per query data usage control** section, choose **Manage**.
6. On the **Manage per query data usage control** page, specify the following values:

   - For **Data limit**, specify a value between 10 MB (minimum) and 7 EB (maximum).

   > **Note**
   >
   > These are limits imposed by the console for data usage controls within workgroups. They do not represent any query limits in Athena.

   - For units, select the unit value from the drop-down list (for example, **Kilobytes KB** or **Exabytes EB**).

   The default action is to cancel the query if it exceeds the limit. This setting cannot be changed.
7. Choose **Save**.

Create or edit a per-workgroup data usage alert

To create or edit a per-workgroup data usage alert

You can set multiple alert thresholds when queries running in a workgroup scan a specified amount of data within a specific period. Alerts are implemented using Amazon CloudWatch alarms and apply to all queries in the workgroup. When a threshold is reached, you can have Amazon SNS send an email to users that you specify. Queries are not automatically canceled when a threshold is reached.

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. In the navigation pane, choose **Workgroups**.
4. Choose the name of the workgroup from the list.
5. Choose **Edit** to edit the workgroup's settings.
6. Scroll down to and expand **Workgroup data usage alerts - optional**.
7. Choose **Add alert**.
8. For **Data usage threshold configuration**, specify values as follows:
   - For **Data threshold**, specify a number, and then select a unit value from the drop-down list.
   - **For Time period**, choose a time period from the drop-down list.
   - For **SNS topic selection**, choose an Amazon SNS topic from the drop-down list. Or, choose **Create SNS topic** to go directly to the Amazon SNS console, create the Amazon SNS topic, and set up a subscription for it for one of the users in your Athena account. For more information, see Getting started with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.
9. Choose **Add alert** if you are creating a new alert, or **Save** to save an existing alert.

Use Athena workgroup APIs

The following are some of the REST API operations used for Athena workgroups. In all of the following operations except for `ListWorkGroups`, you must specify a workgroup. In other operations, such as `StartQueryExecution`, the workgroup parameter is optional and the operations are not listed here. For the full list of operations, see Amazon Athena API Reference.

- CreateWorkGroup
- DeleteWorkGroup
- GetWorkGroup
- ListWorkGroups
- UpdateWorkGroup

Troubleshoot workgroup errors

Use the following tips to troubleshoot workgroups.

- Check permissions for individual users in your account. They must have access to the location for query results, and to the workgroup in which they want to run queries. If they want to switch workgroups, they too need permissions to both workgroups. For information, see Use IAM policies to control workgroup access.
- Pay attention to the context in the Athena console, to see in which workgroup you are going to run queries. If you use the driver, make sure to set the workgroup to the one you need. For information, see the section called "Specify a workgroup for queries".
- If you use the API or the drivers to run queries, you must specify the query results location using one of the following ways: for individual queries, use OutputLocation (client-side). In the workgroup, use WorkGroupConfiguration. If the location is not specified in either way, Athena issues an error at query runtime.
- If you override client-side settings with workgroup settings, you may encounter errors with query result location. For example, a workgroup's user may not have permissions to the workgroup's location in Amazon S3 for storing query results. In this case, add the necessary permissions.
- Workgroups introduce changes in the behavior of the API operations. Calls to the following existing API operations require that users in your account have resource-based permissions in IAM to the workgroups in which they make them. If no permissions to the workgroup and to workgroup actions exist, the following API actions throw `AccessDeniedException`: **CreateNamedQuery**, **DeleteNamedQuery**, **GetNamedQuery**, **ListNamedQueries**, **StartQueryExecution**, **StopQueryExecution**, **ListQueryExecutions**, **GetQueryExecution**, **GetQueryResults**, and **GetQueryResultsStream** (this API action is only available for use with the driver and is not exposed otherwise for public use). For more information, see Actions, resources, and condition keys for Amazon Athena in the *Service Authorization Reference*.

  Calls to the **BatchGetQueryExecution** and **BatchGetNamedQuery** API operations return information only about queries that run in workgroups to which users have access. If the user has no access to the workgroup, these API operations return the unauthorized query IDs as part of the unprocessed IDs list. For more information, see the section called "Use Athena workgroup APIs".
- If the workgroup in which a query will run is configured with an enforced query results location, do not specify an `external_location` for the CTAS query. Athena issues an error and fails a query that specifies an `external_location` in this case. For example, this query

fails, if you override client-side settings for query results location, enforcing the workgroup to use its own location: `CREATE TABLE <DB>.<TABLE1> WITH (format='Parquet', external_location='s3://amzn-s3-demo-bucket/test/') AS SELECT * FROM <DB>.<TABLE2> LIMIT 10;`

You may see the following errors. This table provides a list of some of the errors related to workgroups and suggests solutions.

Workgroup errors

Error	Occurs when...
query state CANCELED. Bytes scanned limit was exceeded.	A query hits a per-query data limit and is canceled. Consider rewriting the query so that it reads less data, or contact your account administrator.
User: *arn:aws:iam::123456789012:user/abc* is not authorized to perform: athena:StartQueryExecution on resource: *arn:aws:athena:us-east-1:123456789012:workgroup/workgroupname*	A user runs a query in a workgroup, but does not have access to it. Update your policy to have access to the workgroup.
INVALID_INPUT. WorkGroup <name> is disabled.	A user runs a query in a workgroup, but the workgroup is disabled. Your workgroup could be disabled by your administrator. It is possible also that you don't have access to it. In both cases, contact an administrator who has access to modify workgroups.
INVALID_INPUT. WorkGroup <name> is not found.	A user runs a query in a workgroup, but the workgroup does not exist. This could happen if the workgroup was deleted. Switch to another workgroup to run your query.
InvalidRequestException: when calling the StartQueryExecution operation: No output	A user runs a query with the API without specifying the location for query results. You

Error	Occurs when...
location provided. An output location is required either through the Workgroup result configuration setting or as an API input.	must set the output location for query results using one of the two ways: either for individua l queries, using OutputLocation (client-s ide), or in the workgroup, using WorkGroup Configuration.
The Create Table As Select query failed because it was submitted with an 'external _location' property to an Athena Workgroup that enforces a centralized output location for all queries. Please remove the 'external _location' property and resubmit the query.	If the workgroup in which a query runs is configured with an enforced query results location, and you specify an `external_location` for the CTAS query. In this case, remove the `external_location` and rerun the query.
Cannot create prepared statement *`prepared_statement_name`* . The number of prepared statements in this workgroup exceeds the limit of 1000.	The workgroup contains more than the limit of 1000 prepared statements. To work around this issue, use DEALLOCATE PREPARE to remove one or more prepared statements from the workgroup. Alternatively, create a new workgroup.

Manage query processing capacity

You can use capacity reservations to get dedicated processing capacity for the queries you run in Athena. With capacity reservations, you can take advantage of workload management capabilities that help you prioritize, control, and scale your most important interactive workloads. For example, you can add capacity at any time to increase the number of queries you can run concurrently, control which workloads can use the capacity, and share capacity among workloads. Capacity is fully-managed by Athena and held for you as long as you require. Setup is easy and no changes to your SQL statements are required.

To get processing capacity for your queries, you create a capacity reservation, specify the number of Data Processing Units (DPUs) that you require, and assign one or more workgroups to the reservation.

Workgroups play an important role when you use capacity reservations. Workgroups allow you to organize queries into logical groupings. With capacity reservations, you selectively assign capacity to workgroups so that you control how the queries for each workgroup behave and how they are billed. For more information about workgroups, see Use workgroups to control query access and costs.

Assigning workgroups to reservations lets you give priority to the queries that you submit to the assigned workgroups. For example, you could allocate capacity to a workgroup used for time-sensitive financial reporting queries to isolate those queries from less critical queries in another workgroup. This enables consistent query execution for critical workloads while allowing other workloads to run independently.

You can use capacity reservations and workgroups together to meet different requirements. The following are some example scenarios:

- **Isolation –** To isolate an important workload, you assign a single workgroup to one reservation. Only queries from the assigned workgroup use the processing capacity from the chosen reservation.
- **Sharing –** Multiple workloads can share capacity from one reservation. For example, if you want a predictable monthly cost for a specific set of workloads, you can assign multiple workgroups to a single reservation. The assigned workgroups share the reservation's capacity.
- **Mixed model –** You can use capacity reservations and per-query billing at the same time in the same account. For example, to ensure reliable execution of queries that support a production application, you assign a workgroup for those queries to a capacity reservation. When developing the queries before you move them to the production workgroup, you use a separate workgroup that is not associated with a reservation and therefore uses per-query billing.

Understand DPUs

Capacity is measured in Data Processing Units (DPUs). DPUs represent the compute and memory resources used by Athena to access and process data on your behalf. One DPU provides 4 vCPUs and 16 GB of memory. The number of DPUs that you specify influences the number of queries that you can run concurrently. For example, a reservation with 256 DPUs can support approximately twice the number of concurrent queries than a reservation with 128 DPUs.

You can create up to 100 capacity reservations with up to 1,000 total DPUs per account and region. The minimum number of DPUs that you can request is 24. If you require more than 1,000 DPUs for your use case, please reach out to athena-feedback@amazon.com.

For information about estimating your capacity requirements, see Determine capacity requirements. For pricing information, see Amazon Athena pricing.

Considerations and limitations

- The feature requires Athena engine version 3.
- A single workgroup can be assigned to at most one reservation at a time, and you can add a maximum of 20 workgroups to a reservation.
- You cannot add Spark enabled workgroups to a capacity reservation.
- To delete a workgroup that has been assigned to a reservation, remove the workgroup from the reservation first.
- The minimum number of DPUs you can provision is 24.
- You can create up to 100 capacity reservations with up to 1,000 total DPUs per account and region.
- Requests for capacity are not guaranteed and can take up to 30 minutes to complete.
- There is a minimum billing period of 1 hour per reservation. After 1 hour, capacity is billed per minute. For pricing information, see Amazon Athena pricing.
- Reserved capacity is not transferable to another capacity reservation, AWS account, or AWS Region.
- DDL queries on capacity reservations consume DPUs.
- Queries that run on provisioned capacity do not count against your active query limits for DDL and DML.
- If all DPUs are in use, submitted queries are queued. Such queries are not rejected and do not go to on-demand capacity.
- The `DPUConsumed` CloudWatch metric is per-workgroup rather than per-reservation. Thus, if you move a workgroup from one reservation to another, the `DPUConsumed` metric includes data from the time when the workgroup belonged to the first reservation. For more information about using CloudWatch metrics in Athena, see Monitor Athena query metrics with CloudWatch.
- Currently, the feature is available in the following AWS Regions:
  - US East (N. Virginia)
  - US East (Ohio)
  - US West (Oregon)
  - Asia Pacific (Mumbai)

- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)
- South America (São Paulo)

Topics

Determine capacity requirements

Before you create a capacity reservation, you can estimate the capacity required so that you can assign it the correct number of DPUs. And, after a reservation is in use, you might want to check the reservation for insufficient or excess capacity. This topic describes techniques that you can use to make these estimates and also describes some AWS tools for assessing usage and cost.

Topics

Estimate required capacity

When estimating capacity requirements, it is useful to consider two perspectives: how much capacity a particular query might require, and how much capacity you might need in general.

Estimate per-query capacity requirements

To determine the number of DPUs that a query might would require, you can use the following guidelines:

- DDL queries consume 4 DPUs.
- DML queries consume between 4 and 124 DPUs.

Athena determines the number of DPUs required by a DML query when the query is submitted. The number varies based on data size, storage format, query construction, and other factors. Generally, Athena tries to select the lowest, most efficient DPU number. If Athena determines that more computational power is required for the query to complete successfully, it increases the number of DPUs assigned to the query.

Estimate workload specific capacity requirements

To determine how much capacity you might require to run multiple queries at the same time, consider the general guidelines in the following table:

Concurrent queries	DPUs required
10	40 or more
20	96 or more
30 or more	240 or more

Note that the actual number of DPUs that you need depends on your goals and analysis patterns. For example, if you want queries to start immediately without queuing, determine your peak concurrent query demand, and then provision the number of DPUs accordingly.

You can provision fewer DPUs than your peak demand, but queuing may result when peak demand occurs. When queuing occurs, Athena holds your queries in a queue and runs them when capacity becomes available.

If your goal is to run queries within a fixed budget, you can use the AWS Pricing Calculator to determine the number of DPUs that fit your budget.

Lastly, remember that data size, storage format, and how a query is written influence the DPUs that a query requires. To increase query performance, you can compress or partition your data or convert it into columnar formats. For more information, see Optimize Athena performance.

Signs that more capacity is required

Insufficient capacity error messages and query queuing are two indications that your assigned capacity is inadequate.

If your queries fail with an insufficient capacity error message, your capacity reservation's DPU count is too low for your query. For example, if you have a reservation with 24 DPUs and run a query that requires more than 24 DPUs, the query will fail. To monitor for this query error, you can use Athena's EventBridge events. Try adding more DPUs and re-running your query.

If many queries are queued, it means your capacity is fully utilized by other queries. To reduce the queuing, do one of the following:

- Add DPUs to your reservation to increase query concurrency.
- Remove workgroups from your reservation to free up capacity for other queries.

To check for excessive query queuing, use the Athena query queue time CloudWatch metric for the workgroups in your capacity reservation. If the value is above your preferred threshold, you can add DPUs to the capacity reservation.

Check for idle capacity

To check for idle capacity, you can either decrease the number of DPUs in the reservation or increase its workload, and then observe the results.

To check for idle capacity

1. Do one of the following:
   - Reduce the number of DPUs in your reservation (reduce the resources available)
   - Add workgroups to your reservation (increase the workload)
2. Use CloudWatch to measure the query queue time.
3. If the queue time increases beyond a desirable level, do one of the following
   - Remove workgroups

- Add DPUs to your capacity reservation

4. After each change, check the performance and query queue time.
5. Continue to adjust the workload and/or DPU count to attain the desired balance.

If you do not want to maintain capacity outside a preferred time period, you can cancel the reservation and create another reservation later. However, even if you recently cancelled capacity from another reservation, requests for new capacity are not guaranteed, and new reservations take time to create.

Tools for assessing capacity requirements and cost

You can use the following services and features in AWS to measure your Athena usage and costs.

CloudWatch metrics

You can configure Athena to publish query-related metrics to Amazon CloudWatch at the workgroup level. After you enable metrics for the workgroup, the metrics for the workgroup's queries are displayed in the Athena console on the workgroup's details page.

For information about the Athena metrics published to CloudWatch and their dimensions, see Monitor Athena query metrics with CloudWatch.

CloudWatch usage metrics

You can use CloudWatch usage metrics to provide visibility into your how your account uses resources by displaying your current service usage on CloudWatch graphs and dashboards. For Athena, usage availability metrics correspond to AWS service quotas for Athena. You can configure alarms that alert you when your usage approaches a service quota.

For more information, see Monitor Athena usage metrics with CloudWatch.

Amazon EventBridge events

You can use Amazon Athena with Amazon EventBridge to receive real-time notifications regarding the state of your queries. When a query you have submitted changes states, Athena publishes an event to EventBridge that contains information about the query state transition. You can write simple rules for events that are of interest to you and take automated actions when an event matches a rule.

For more information, see the following resources.

- Monitor Athena query events with EventBridge
- What Is Amazon EventBridge?
- Amazon EventBridge events

Tags

In Athena, capacity reservations support tags. A tag consists of a key and a value. To track your costs in Athena, you can use AWS-generated cost allocation tags. AWS uses the cost allocation tags to organize your resource costs on your Cost and Usage Report. This makes it easier for you to categorize and track your AWS costs. To activate cost allocation tags for Athena, you use the AWS Billing and Cost Management console.

For more information, see the following resources.

- Tag Athena resources
- Activating the AWS-generated cost allocation tags
- Using AWS cost allocation tags

Create capacity reservations

To get started, you create a capacity reservation that has the number of DPUs that you require, and then assign one or more workgroups that will use that capacity for their queries. You can adjust your capacity later as needed to provide more consistent performance or better manage costs. For information about estimating your capacity requirements, see Determine capacity requirements.

> **Important**
>
> Requests for capacity are not guaranteed and can take up to 30 minutes to complete.

To create a capacity reservation

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. Choose **Administration**, **Capacity reservations**.

4. Choose **Create capacity reservation**.

5. On the **Create capacity reservation** page, for **Capacity reservation name**, enter name. The name must be unique, from 1 to 128 characters long, and use only the characters a-z, A-Z, 0-9, _(underscore), .(period) and -(hyphen).You cannot change the name after you create the reservation.

6. For **DPU**, choose or enter the number of data processing units (DPUs) that you want in increments of 4. For more information, see Understand DPUs.

7. (Optional) Expand the **Tags** option, and then choose **Add new tag** to add one or more custom key/value pairs to associate with the capacity reservation resource. For more information, see Tag Athena resources.

8. Choose **Review**.

9. At the **Confirm create capacity reservation** prompt, confirm the number of DPUs, AWS Region, and other information. If you accept, choose **Submit**.

   On the details page, your capacity reservation's **Status** shows as **Pending**. When your reservation capacity is available to run queries, its status shows as **Active**.

At this point, you are ready to add one or more workgroups to your reservation. For steps, see Add workgroups to a reservation.

Manage reservations

You can view and manage your capacity reservations on the **Capacity reservations** page. You can perform management tasks like adding or reducing DPUs, modifying workgroup assignments, and tagging or cancelling reservations.

To view and manage capacity reservations

1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. Choose **Administration**, **Capacity reservations**.

4. On the capacity reservations page, you can perform the following tasks:

   - To create a capacity reservation, choose **Create capacity reservation**.
   - Use the search box to filter reservations by name or number of DPUs.

- Choose the status drop-down menu to filter by capacity reservation status (for example, **Active** or **Cancelled**). For information about reservation status, see Understand reservation status.
- To view details for a capacity reservation, choose the link for the reservation. The details page for the reservation includes options for editing capacity, adding workgroups, removing workgroups, and for cancelling the reservation.
- To edit a reservation (for example, by adding or removing DPUs), select the button for the reservation, and then choose **Edit**.
- To cancel a reservation, select the button for the reservation, and then choose **Cancel**.

Understand reservation status

The following table describes the possible status values for a capacity reservation.

Status	Description
Pending	Athena is processing your capacity request. Capacity is not ready to run queries.
Active	Capacity is available to run queries.
Failed	Your request for capacity was not completed successfully. Note that fulfillment of capacity requests is not guaranteed. Failed reservations count towards your account DPU limits. To release the usage, you must cancel the reservation.
Update pending	Athena is processing a change to the reservation. For example, this status occurs after you edit the reservation to add or remove DPUs.
Cancelling	Athena is processing a request to cancel the reservation. Queries that are still running in the workgroups that were using the reservation are allowed to finish, but other queries in the workgroup will use on-demand (non-provisioned) capacity.
Cancelled	The capacity reservation cancellation is complete. Cancelled reservati ons remain in the console for 45 days. After 45 days, Athena will delete the reservation. During the 45 days, you cannot re-purpose or

Status	Description
	reuse the reservation, but you can refer to its tags and view its details for historical reference. Cancelled capacity is not guaranteed to be re-reservable at a future date. Capacity cannot be transferred to another reservation, AWS account or AWS Region.

Understand Active DPUs and Target DPUs

In the list of capacity reservations in the Athena console, your reservation displays two DPU values: **Active DPU** and **Target DPU**.

- **Active DPU** – The number of DPUs that are available in your reservation to run queries. For example, if you request 100 DPUs, and your request is fulfilled, **Active DPU** displays **100**.
- **Target DPU** – The number of DPUs that your reservation is in the process of moving to. **Target DPU** displays a value different than **Active DPU** when a reservation is being created, or an increase or decrease in the number of DPUs is pending.

For example, after you submit a request to create a reservation with 24 DPUs, the reservation **Status** will be **Pending**, **Active DPU** will be **0**, and the **Target DPU** will be **24**.

If you have a reservation with 100 DPUs, and edit your reservation to request an increase of 20 DPUs, the **Status** will be **Update pending**, **Active DPU** will be **100**, and **Target DPU** will be **120**.

If you have a reservation with 100 DPUs, and edit your reservation to request a decrease of 20 DPUs, the **Status** will be **Update pending**, **Active DPU** will be **100**, and **Target DPU** will be **80**.

During these transitions, Athena is actively working to acquire or reduce the number of DPUs based on your request. When **Active DPU** becomes equal to **Target DPU**, the target number has been reached and no changes are pending.

To retrieve these values programmatically, you can call the [GetCapacityReservation](https://docs.aws.amazon.com/athena/latest/APIReference/API_GetCapacityReservation.html) API action. The API refers to **Active DPU** and **Target DPU** as `AllocatedDpus` and `TargetDpus`.

Topics

- Edit capacity reservations

- Add workgroups to a reservation
- Remove a workgroup from a reservation
- Cancel a capacity reservation
- Delete a capacity reservation

Edit capacity reservations

After you create a capacity reservation, you can adjust its number of DPUs and add or remove its custom tags.

To edit a capacity reservation

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. Choose **Administration**, **Capacity reservations**.
4. In the list of capacity reservations, do one of the following:
   - Select the button next to the reservation, and then choose **Edit**.
   - Choose the reservation link, and then choose **Edit**.
5. For **DPU**, choose or enter the number of data processing units that you want in increments of 4. The minimum number of DPUs that you can have is 24. For more information, see Understand DPUs.

   > **Note**
   >
   > - You can add DPUs to an existing capacity reservation at any time. However, you cannot decrease the number of DPUs until 1 hour after you create the reservation or add DPUs to it.
   > - When you request to decrease DPUs while queries are running, the system waits for the queries to complete before updating the capacity reservation with the new target DPUs.

6. (Optional) For **Tags**, choose **Remove** to remove a tag, or choose **Add new tag** to add a new tag.
7. Choose **Submit**. The details page for the reservation shows the updated configuration.

Add workgroups to a reservation

After you create a capacity reservation, you can add up to 20 workgroups to the reservation. Adding a workgroup to a reservation tells Athena which queries should execute on your reserved capacity. Queries from workgroups that are not associated with a reservation continue to run using the default per terabyte (TB) scanned pricing model.

When a reservation has two or more workgroups, queries from those workgroups can use the reservation's capacity. You can add and remove workgroups at any time. When you add or remove workgroups, queries that are running are not interrupted.

When your reservation is in a pending state, queries from workgroups that you have added continue to run using the default per terabyte (TB) scanned pricing model until the reservation becomes active.

To add one or more workgroups to your capacity reservation

1. On the details page for the capacity reservation, choose **Add workgroups**.
2. On the **Add workgroups** page, select the workgroups that you want to add, and then choose **Add workgroups**. You cannot assign a workgroup to more than one reservation.

   The details page for your capacity reservation lists the workgroups that you added. Queries that run in those workgroups will use the capacity that you reserved when the reservation is active.

Remove a workgroup from a reservation

If you no longer require dedicated capacity for a workgroup or want to move a workgroup to its own reservation, you can remove it at any time. Removing a workgroup from a reservation is a straightforward process. After you remove a workgroup from a reservation, queries from the removed workgroup default to using on-demand (non-provisioned) capacity and are billed based on terabytes (TB) scanned.

To remove one or more workgroups from a reservation

1. On the details page for the capacity reservation, select the workgroups that you want to remove.
2. Choose **Remove workgroups**. The **Remove workgroups?** prompt informs you that all currently active queries will finish before the workgroup is removed from the reservation..

3. Choose **Remove**. The details page for your capacity reservation show that the removed workgroups are no longer present.

Cancel a capacity reservation

If you no longer want to use a capacity reservation, you can cancel it. Queries that are still running in the workgroups that were using the reservation will be allowed to finish, but other queries in the workgroup will no longer use the reservation.

> **Note**
>
> Cancelled capacity is not guaranteed to be re-reservable at a future date. Capacity cannot be transferred to another reservation, AWS account or AWS Region.

To cancel a capacity reservation

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. Choose **Administration**, **Capacity reservations**.
4. In the list of capacity reservations, do one of the following:
   - Select the button next to the reservation, and then choose **Cancel**.
   - Choose the reservation link, and then choose **Cancel capacity reservation**.
5. At the **Cancel capacity reservation?** prompt, enter **cancel**, and then choose **Cancel capacity reservation**.

   The reservation's status changes to **Cancelling**, and a progress banner informs you that the cancellation is in progress.

   When the cancellation is complete, the capacity reservation remains, but its status shows as **Cancelled**. The reservation will be deleted 45 days after cancellation. During the 45 days, you cannot re-purpose or reuse a reservation that has been cancelled, but you can refer to its tags and view it for historical reference.

Delete a capacity reservation

If you want to remove all references to a cancelled capacity reservation, you can delete the reservation. A reservation must be cancelled before it can be deleted. A deleted reservation is immediately removed from your account and can no longer be referenced, including by its ARN.

To delete a capacity reservation

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. Choose **Administration**, **Capacity reservations**.
4. In the list of capacity reservations, do one of the following:
   - Select the button next to the cancelled reservation, and then choose **Actions**, **Delete**.
   - Choose the reservation link, and then choose **Delete**.
5. At the **Delete capacity reservation?** prompt, choose **Delete**.

   A banner informs you that the capacity reservation has been successfully deleted. The deleted reservation no longer appears in the list of capacity reservations.

IAM policies for capacity reservations

To control access to capacity reservations, use resource-level IAM permissions or identity-based IAM policies. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

The following procedure is specific to Athena.

For IAM-specific information, see the links listed at the end of this section. For information about example JSON capacity reservations policies, see Example capacity reservation policies.

To use the visual editor in the IAM console to create a capacity reservation policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**, and then choose **Create policy**.
3. On the **Visual editor** tab, choose **Choose a service**. Then choose Athena to add to the policy.

4. Choose **Select actions**, and then choose the actions to add to the policy. The visual editor shows the actions available in Athena. For more information, see [Actions, resources, and condition keys for Amazon Athena] in the *Service Authorization Reference*.
5. Choose **add actions** to type a specific action or use wild card characters (*) to specify multiple actions.

   By default, the policy that you are creating allows the actions that you choose. If you chose one or more actions that support resource-level permissions to the `capacity-reservation` resource in Athena, then the editor lists the `capacity-reservation` resource.
6. Choose **Resources** to specify the specific capacity reservations for your policy. For example JSON capacity reservation policies, see Example capacity reservation policies.
7. Specify the `capacity-reservation` resource as follows:

```
arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>
```

8. Choose **Review policy**, and then type a **Name** and a **Description** (optional) for the policy that you are creating. Review the policy summary to make sure that you granted the intended permissions.
9. Choose **Create policy** to save your new policy.
10. Attach this identity-based policy to a user, a group, or role.

For more information, see the following topics in the *Service Authorization Reference* and *IAM User Guide*:

- Actions, resources, and condition keys for Amazon Athena
- Creating policies with the visual editor
- Adding and removing IAM policies
- Controlling access to resources

For example JSON capacity reservation policies, see Example capacity reservation policies.

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference.

Example capacity reservation policies

This section includes example policies you can use to enable various actions on capacity reservations. Whenever you use IAM policies, make sure that you follow IAM best practices. For more information, see Security best practices in IAM in the *IAM User Guide*.

A capacity reservation is an IAM resource managed by Athena. Therefore, if your capacity reservation policy uses actions that take `capacity-reservation` as an input, you must specify the capacity reservation's ARN as follows:

```
"Resource": [arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>]
```

Where *`<capacity-reservation-name>`* is the name of your capacity reservation. For example, for a capacity reservation named `test_capacity_reservation`, specify it as a resource as follows:

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:capacity-reservation/test_capacity_reservation"]
```

For a complete list of Amazon Athena actions, see the API action names in the Amazon Athena API Reference. For more information about IAM policies, see Creating policies with the visual editor in the *IAM User Guide*.

Example Example policy to list capacity reservations

The following policy allows all users to list all capacity reservations.

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

Example Example policy for management operations

The following policy allows a user to create, cancel, obtain details, and update the capacity reservation `test_capacity_reservation`. The policy also allows a user to assign `workgroupA` and `workgroupB` to `test_capacity_reservation`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect": "Allow",
         "Action": [
             "athena:CreateCapacityReservation",
             "athena:GetCapacityReservation",
             "athena:CancelCapacityReservation",
             "athena:UpdateCapacityReservation",
             "athena:GetCapacityAssignmentConfiguration",
             "athena:PutCapacityAssignmentConfiguration"
         ],
         "Resource": [
             "arn:aws:athena:us-east-1:123456789012:capacity-
reservation/test_capacity_reservation",
             "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA",
             "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupB"
         ]
      }
   ]
}
```

Athena capacity reservation APIs

The following list contains reference links to Athena capacity reservation API actions. For data structures and other Athena API actions, see the *Amazon Athena API Reference*.

- CancelCapacityReservation
- CreateCapacityReservation
- DeleteCapacityReservation
- GetCapacityAssignmentConfiguration
- GetCapacityReservation
- ListCapacityReservations

- PutCapacityAssignmentConfiguration
- UpdateCapacityReservation

Optimize Athena performance

This topic provides general information and specific suggestions for improving the performance of your Athena queries, and how to work around errors related to limits and resource usage.

Broadly speaking, optimizations can be grouped into service, query, and data structure categories. Decisions made at the service level, on how you write your queries, and on how you structure your data and tables can all influence performance.

Topics

Optimize service use

Service level considerations include the number of workloads you run per account, service quotas not only for Athena, but across services, and thinking about how to reduce 'out of resource' errors.

Topics

Run one workload per account to avoid service quota limits

Athena enforces quotas for metrics like query running time, the number of concurrent queries in an account, and API request rates. For more information about these quotas, see Service Quotas. Exceeding these quotas causes a query to fail — either when it is submitted, or during query execution.

Many of the performance optimization tips on this page can help reduce the running time of queries. Optimization frees up capacity so that you can run more queries within the concurrency quota and keeps queries from being cancelled for running too long.

Quotas on the number of concurrent queries and API requests are per AWS account and AWS Region. We recommend running one workload per AWS account (or using separate provisioned capacity reservations) to keep workloads from competing for the same quota.

If you run two workloads in the same account, one of the workloads can run a burst of queries. This can cause the remaining workload to be throttled or blocked from running queries. To avoid this, you can move the workloads to separate accounts to give each workload its own concurrency quota. Creating a provisioned capacity reservation for one or both of the workloads accomplishes the same goal.

Consider quotas in other services

When Athena runs a query, it can call other services that enforce quotas. During query execution, Athena can make API calls to the AWS Glue Data Catalog, Amazon S3, and other AWS services like IAM and AWS KMS. If you use federated queries, Athena also calls AWS Lambda. All of these services have their own limits and quotas that can be exceeded. When a query execution encounters errors from these services, it fails and includes the error from the source service. Recoverable errors are retried, but queries can still fail if the issue does not resolve itself in time. Make sure to read error messages thoroughly to determine if they come from Athena or from another service. Some of the relevant errors are covered in this performance tuning section.

For more information about working around errors caused by Amazon S3 service quotas, see Avoid having too many files later in this document. For more information about Amazon S3 performance optimization, see Best practices design patterns: optimizing Amazon S3 performance in the *Amazon S3 User Guide*.

Reduce 'out of resource' errors

Athena runs queries in a distributed query engine. When you submit a query, the Athena engine query planner estimates the compute capacity required to run the query and prepares a cluster of compute nodes accordingly. Some queries like DDL queries run on only one node. Complex queries over large data sets run on much bigger clusters. The nodes are uniform, with the same memory, CPU, and disk configurations. Athena scales out, not up, to process more demanding queries.

Sometimes the demands of a query exceed the resources available to the cluster running the query. When this happens, the query fails with the error Query exhausted resources at this scale factor.

The resource most commonly exhausted is memory, but in rare cases it can also be disk space. Memory errors commonly occur when the engine performs a join or a window function, but they can also occur in distinct counts and aggregations.

Even if a query fails with an 'out of resource' error once, it might succeed when you run it again. Query execution is not deterministic. Factors such as how long it takes to load data and how intermediate datasets are distributed over the nodes can result in different resource usage. For example, imagine a query that joins two tables and has a heavy skew in the distribution of the values for the join condition. Such a query can succeed most of the time but occasionally fail when the most common values end up being processed by the same node.

To prevent your queries from exceeding available resources, use the performance tuning tips mentioned in this document. In particular, for tips on how to optimize queries that exhaust the resources available, see Optimize joins, Reduce the scope of window functions, or remove them, and Optimize queries by using approximations.

Optimize queries

Use the query optimization techniques described in this section to make queries run faster or as workarounds for queries that exceed resource limits in Athena.

Optimize joins

There are many different strategies for executing joins in a distributed query engine. Two of the most common are distributed hash joins and queries with complex join conditions.

In a distributed hash join, place large tables on the left, small tables on the right

The most common type of join uses an equality comparison as the join condition. Athena runs this type of join as a distributed hash join.

In a distributed hash join, the engine builds a lookup table (hash table) from one of the sides of the join. This side is called the *build side*. The records of the build side are distributed across the nodes.

Each node builds a lookup table for its subset. The other side of the join, called the *probe side*, is then streamed through the nodes. The records from the probe side are distributed over the nodes in the same way as the build side. This enables each node to perform the join by looking up the matching records in its own lookup table.

When the lookup tables created from the build side of the join don't fit into memory, queries can fail. Even if the total size of the build side is less than the available memory, queries can fail if the distribution of the records has significant skew. In an extreme case, all records could have the same value for the join condition and have to fit into memory on a single node. Even a query with less skew can fail if a set of values gets sent to the same node and the values add up to more than the available memory. Nodes do have the ability to spill records to disk, but spilling slows query execution and can be insufficient to prevent the query from failing.

Athena attempts to reorder joins to use the larger relation as the probe side, and the smaller relation as the build side. However, because Athena does not manage the data in tables, it has limited information and often must assume that the first table is the larger and the second table is the smaller.

When writing joins with equality-based join conditions, assume that the table to the left of the `JOIN` keyword is the probe side and the table to the right is the build side. Make sure that the right table, the build side, is the smaller of the tables. If it is not possible to make the build side of the join small enough to fit into memory, consider running multiple queries that join subsets of the build table.

Use EXPLAIN to analyze queries with complex joins

Queries with complex join conditions (for example, queries that use `LIKE` , >, or other operators), are often computationally demanding. In the worst case, every record from one side of the join must be compared to every record on the other side of the join. Because the execution time grows with the square of the number of records, such queries run the risk of exceeding the maximum execution time.

To find out how Athena will execute your query in advance, you can use the `EXPLAIN` statement. For more information, see [Using EXPLAIN and EXPLAIN ANALYZE in Athena]() and [Understand Athena EXPLAIN statement results]().

Reduce the scope of window functions, or remove them

Because window functions are resource intensive operations, they can make queries run slow or even fail with the message Query exhausted resources at this scale factor. Window functions keep all the records that they operate on in memory in order to calculate their result. When the window is very large, the window function can run out of memory.

To make sure your queries run within the available memory limits, reduce the size of the windows that your window functions operate over. To do so, you can add a PARTITIONED BY clause or narrow the scope of existing partitioning clauses.

Use non-window functions

Sometimes queries with window functions can be rewritten without window functions. For example, instead of using row_number to find the top N records, you can use ORDER BY and LIMIT. Instead of using row_number or rank to deduplicate records, you can use aggregate functions like max_by, min_by, and arbitrary.

For example, suppose you have a dataset with updates from a sensor. The sensor periodically reports its battery status and includes some metadata like location. If you want to know the last battery status for each sensor and its location, you can use this query:

```
SELECT sensor_id,
       arbitrary(location) AS location,
       max_by(battery_status, updated_at) AS battery_status
FROM sensor_readings
GROUP BY sensor_id
```

Because metadata like location is the same for every record, you can use the arbitrary function to pick any value from the group.

To get the last battery status, you can use the max_by function. The max_by function picks the value for a column from the record where the maximum value of another column was found. In this case, it returns the battery status for the record with the last update time within the group. This query runs faster and uses less memory than an equivalent query with a window function.

Optimize aggregations

When Athena performs an aggregation, it distributes the records across worker nodes using the columns in the GROUP BY clause. To make the task of matching records to groups as efficient as possible, the nodes attempt to keep records in memory but spill them to disk if necessary.

It is also a good idea to avoid including redundant columns in GROUP BY clauses. Because fewer columns require less memory, a query that describes a group using fewer columns is more efficient.

Numeric columns also use less memory than strings. For example, when you aggregate a dataset that has both a numeric category ID and a category name, use only the category ID column in the GROUP BY clause.

Sometimes queries include columns in the GROUP BY clause to work around the fact that a column must either be part of the GROUP BY clause or an aggregate expression. If this rule is not followed, you can receive an error message like the following:

EXPRESSION_NOT_AGGREGATE: line 1:8: 'category' must be an aggregate expression or appear in GROUP BY clause

To avoid having to add a redundant columns to the GROUP BY clause, you can use the arbitrary function, as in the following example.

```
SELECT country_id,
       arbitrary(country_name) AS country_name,
       COUNT(*) AS city_count
FROM world_cities
GROUP BY country_id
```

The ARBITRARY function returns an arbitrary value from the group. The function is useful when you know all records in the group have the same value for a column, but the value does not identify the group.

Optimize top N queries

The ORDER BY clause returns the results of a query in sorted order. Athena uses distributed sort to run the sort operation in parallel on multiple nodes.

If you don't strictly need your result to be sorted, avoid adding an ORDER BY clause. Also, avoid adding ORDER BY to inner queries if they are not strictly necessary. In many cases, the query planner can remove redundant sorting, but this is not guaranteed. An exception to this rule is if an inner query is doing a top N operation, such as finding the N most recent, or N most common values.

When Athena sees ORDER BY together with LIMIT, it understands that you are running a top N query and uses dedicated operations accordingly.

> **Note**
>
> Although Athena can also often detect window functions like `row_number` that use top `N`, we recommend the simpler version that uses `ORDER BY` and `LIMIT`. For more information, see Reduce the scope of window functions, or remove them.

Include only required columns

If you don't strictly need a column, don't include it in your query. The less data a query has to process, the faster it will run. This reduces both the amount of memory required and the amount of data that has to be sent between nodes. If you are using a columnar file format, reducing the number columns also reduces the amount of data that is read from Amazon S3.

Athena has no specific limit on the number of columns in a result, but how queries are executed limits the possible combined size of columns. The combined size of columns includes their names and types.

For example, the following error is caused by a relation that exceeds the size limit for a relation descriptor:

GENERIC_INTERNAL_ERROR: io.airlift.bytecode.CompilationException

To work around this issue, reduce the number of columns in the query, or create subqueries and use a `JOIN` that retrieves a smaller amount of data. If you have queries that do `SELECT *` in the outermost query, you should change the `*` to a list of only the columns that you need.

Optimize queries by using approximations

Athena has support for approximation aggregate functions for counting distinct values, the most frequent values, percentiles (including approximate medians), and creating histograms. Use these functions whenever exact values are not needed.

Unlike `COUNT(DISTINCT col)` operations, approx_distinct uses much less memory and runs faster. Similarly, using numeric_histogram instead of histogram uses approximate methods and therefore less memory.

Optimize LIKE

You can use `LIKE` to find matching strings, but with long strings, this is compute intensive. The regexp_like function is in most cases a faster alternative, and also provides more flexibility.

Often you can optimize a search by anchoring the substring that you are looking for. For example, if you're looking for a prefix, it is much better to use '*substr*%' instead of '%*substr*%'. Or, if you're using `regexp_like`, '^*substr*'.

Use UNION ALL instead of UNION

`UNION ALL` and `UNION` are two ways to combine the results of two queries into one result. `UNION ALL` concatenates the records from the first query with the second, and `UNION` does the same, but also removes duplicates. `UNION` needs to process all the records and find the duplicates, which is memory and compute intensive, but `UNION ALL` is a relatively quick operation. Unless you need to deduplicate records, use `UNION ALL` for the best performance.

Use UNLOAD for large result sets

When the results of a query are expected to be large (for example, tens of thousands of rows or more), use UNLOAD to export the results. In most cases, this is faster than running a regular query, and using `UNLOAD` also gives you more control over the output.

When a query finishes executing, Athena stores the result as a single uncompressed CSV file on Amazon S3. This takes longer than `UNLOAD`, not only because the result is uncompressed, but also because the operation cannot be parallelized. In contrast, `UNLOAD` writes results directly from the worker nodes and makes full use of the parallelism of the compute cluster. In addition, you can configure `UNLOAD` to write the results in compressed format and in other file formats such as JSON and Parquet.

For more information, see UNLOAD.

Use CTAS or Glue ETL to materialize frequently used aggregations

'Materializing' a query is a way of accelerating query performance by storing pre-computed complex query results (for example, aggregations and joins) for reuse in subsequent queries.

If many of your queries include the same joins and aggregations, you can materialize the common subquery as a new table and then run queries against that table. You can create the new table with Create a table from query results (CTAS), or a dedicated ETL tool like Glue ETL.

For example, suppose you have a dashboard with widgets that show different aspects of an orders dataset. Each widget has its own query, but the queries all share the same joins and filters. An order table is joined with a line items table, and there is a filter to show only the last three months. If you identify the common features of these queries, you can create a new table that the widgets can use. This reduces duplication and improves performance. The disadvantage is that you must keep the new table up to date.

Reuse query results

It's common for the same query to run multiple times within a short duration. For example, this can occur when multiple people open the same data dashboard. When you run a query, you can tell Athena to reuse previously calculated results. You specify the maximum age of the results to be reused. If the same query was previously run within that time frame, Athena returns those results instead of running the query again. For more information, see Reuse query results in Athena here in the *Amazon Athena User Guide* and Reduce cost and improve query performance with Amazon Athena Query Result Reuse in the *AWS Big Data Blog*.

Optimize data

Performance depends not only on queries, but also importantly on how your dataset is organized and on the file format and compression that it uses.

Partition your data

Partitioning divides your table into parts and keeps the related data together based on properties such as date, country, or region. Partition keys act as virtual columns. You define partition keys at table creation and use them for filtering your queries. When you filter on partition key columns, only data from matching partitions is read. For example, if your dataset is partitioned by date and your query has a filter that matches only the last week, only the data for the last week is read. For more information about partitioning, see Partition your data.

Pick partition keys that will support your queries

Because partitioning has a significant impact on query performance, be sure to consider how you partition carefully when you design your dataset and tables. Having too many partition keys can result in fragmented datasets with too many files and files that are too small. Conversely, having too few partition keys, or no partitioning at all, leads to queries that scan more data than necessary.

Avoid optimizing for rare queries

A good strategy is to optimize for the most common queries and avoid optimizing for rare queries. For example, if your queries look at time spans of days, don't partition by hour, even if some queries filter to that level. If your data has a granular timestamp column, the rare queries that filter by hour can use the timestamp column. Even if rare cases scan a little more data than necessary, reducing overall performance for the sake of rare cases is usually not a good tradeoff.

To reduce the amount of data that queries have to scan, and thereby improve performance, use a columnar file format and keep the records sorted. Instead of partitioning by hour, keep the records sorted by timestamp. For queries on shorter time windows, sorting by timestamp is almost as efficient as partitioning by hour. Furthermore, sorting by timestamp does not typically hurt the performance of queries on time windows counted in days. For more information, see Use columnar file formats.

Note that queries on tables with tens of thousands of partitions perform better if there are predicates on all partition keys. This is another reason to design your partitioning scheme for the most common queries. For more information, see Query partitions by equality.

Use partition projection

Partition projection is an Athena feature that stores partition information not in the AWS Glue Data Catalog, but as rules in the properties of the table in AWS Glue. When Athena plans a query on a table configured with partition projection, it reads the table's partition projection rules. Athena computes the partitions to read in memory based on the query and the rules instead of looking up partitions in the AWS Glue Data Catalog.

Besides simplifying partition management, partition projection can improve performance for datasets that have large numbers of partitions. When a query includes ranges instead of specific values for partition keys, looking up matching partitions in the catalog takes longer the more partitions there are. With partition projection, the filter can be computed in memory without going to the catalog, and can be much faster.

In certain circumstances, partition projection can result in worse performance. One example occurs when a table is "sparse." A sparse table does not have data for every permutation of the partition key values described by the partition projection configuration. With a sparse table, the set of partitions calculated from the query and the partition projection configuration are all listed on Amazon S3 even when they have no data.

When you use partition projection, make sure to include predicates on all partition keys. Narrow the scope of possible values to avoid unnecessary Amazon S3 listings. Imagine a partition key that has a range of one million values and a query that does not have any filters on that partition key. To run the query, Athena must perform at least one million Amazon S3 list operations. Queries are fastest when you query on specific values, regardless of whether you use partition projection or store partition information in the catalog. For more information, see Query partitions by equality.

When you configure a table for partition projection, make sure that the ranges that you specify are reasonable. If a query doesn't include a predicate on a partition key, all the values in the range for that key are used. If your dataset was created on a specific date, use that date as the starting point for any date ranges. Use `NOW` as the end of date ranges. Avoid numeric ranges that have large number of values, and consider using the injected type instead.

For more information about partition projection, see Use partition projection with Amazon Athena.

Use partition indexes

Partition indexes are a feature in the AWS Glue Data Catalog that improves partition lookup performance for tables that have large numbers of partitions.

The list of partitions in the catalog is like a table in a relational database. The table has columns for the partition keys and an additional column for the partition location. When you query a partitioned table, the partition locations are looked up by scanning this table.

Just as with relational databases, you can increase the performance of queries by adding indexes. You can add multiple indexes to support different query patterns. The AWS Glue Data Catalog partition index supports both equality and comparison operators like >, >=, and < combined with the `AND` operator. For more information, see Working with partition indexes in AWS Glue in the *AWS Glue Developer Guide* and Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes in the *AWS Big Data Blog*.

Always use STRING as the type for partition keys

When you query on partition keys, remember that Athena requires partition keys to be of type `STRING` in order to push down partition filtering into AWS Glue. If the number of partitions is not small, using other types can lead to worse performance. If your partition key values are date-like or number-like, cast them to the appropriate type in your query.

Remove old and empty partitions

If you remove data from a partition on Amazon S3 (for example, by using Amazon S3 lifecycle), you should also remove the partition entry from the AWS Glue Data Catalog. During query planning, any partition matched by the query is listed on Amazon S3. If you have many empty partitions, the overhead of listing these partitions can be detrimental.

Also, if you have many thousands of partitions, consider removing partition metadata for old data that is no longer relevant. For example, if queries never look at data older than a year, you can periodically remove partition metadata for the older partitions. If the number of partitions grows into the tens of thousands, removing unused partitions can speed up queries that don't include predicates on all partition keys. For information about including predicates on all partition keys in your queries, see Query partitions by equality.

Query partitions by equality

Queries that include equality predicates on all partition keys run faster because the partition metadata can be loaded directly. Avoid queries in which one or more of the partition keys does not have a predicate, or the predicate selects a range of values. For such queries, the list of all partitions has to be filtered to find matching values. For most tables, the overhead is minimal, but for tables with tens of thousands or more partitions, the overhead can become significant.

If it is not possible to rewrite your queries to filter partitions by equality, you can try partition projection. For more information, see Use partition projection.

Avoid using MSCK REPAIR TABLE for partition maintenance

Because `MSCK REPAIR TABLE` can take a long time to run, only adds new partitions, and does not remove old partitions, it is not an efficient way to manage partitions (see Considerations and limitations).

Partitions are better managed manually using the AWS Glue Data Catalog APIs, ALTER TABLE ADD PARTITION, or AWS Glue crawlers. As an alternative, you can use partition projection, which removes the need to manage partitions altogether. For more information, see Use partition projection with Amazon Athena.

Validate that your queries are compatible with the partitioning scheme

You can check in advance which partitions a query will scan by using the `EXPLAIN` statement. Prefix your query with the `EXPLAIN` keyword, then look for the source fragment (for example, `Fragment 2 [SOURCE]`) for each table near the bottom of the `EXPLAIN` output. Look for assignments where the right side is defined as a partition key. The line underneath includes a list of all the values for that partition key that will be scanned when the query is run.

For example, suppose you have a query on a table with a `dt` partition key and prefix the query with `EXPLAIN`. If the values in the query are dates, and a filter selects a range of three days, the `EXPLAIN` output might look something like this:

```
dt := dt:string:PARTITION_KEY
    :: [[2023-06-11], [2023-06-12], [2023-06-13]]
```

The EXPLAIN output shows that the planner found three values for this partition key that matched the query. It also shows you what those values are. For more information about using EXPLAIN, see Using EXPLAIN and EXPLAIN ANALYZE in Athena and Understand Athena EXPLAIN statement results.

Use columnar file formats

Columnar file formats like Parquet and ORC are designed for distributed analytics workloads. They organize data by column instead of by row. Organizing data in columnar format offers the following advantages:

- Only the columns needed for the query are loaded
- The overall amount of data that needs to be loaded is reduced
- Column values are stored together, so data can be compressed efficiently
- Files can contain metadata that allow the engine to skip loading unneeded data

As an example of how file metadata can be used, file metadata can contain information about the minimum and maximum values in a page of data. If the values queried are not in the range noted in the metadata, the page can be skipped.

One way to use this metadata to improve performance is to ensure that data within the files are sorted. For example, suppose you have queries that look for records where the `created_at` entry is within a short time span. If your data is sorted by the `created_at` column, Athena can use the minimum and maximum values in the file metadata to skip the unneeded parts of the data files.

When using columnar file formats, make sure that your files aren't too small. As noted in Avoid having too many files, datasets with many small files cause performance issues. This is particularly true with columnar file formats. For small files, the overhead of the columnar file format outweighs the benefits.

Note that Parquet and ORC are internally organized by row groups (Parquet) and stripes (ORC). The default size for row groups is 128 MB, and for stripes, 64 MB. If you have many columns, you can increase the row group and stripe size for better performance. Decreasing the row group or stripe size to less than their default values is not recommended.

To convert other data formats to Parquet or ORC, you can use AWS Glue ETL or Athena. For more information, see Convert to columnar formats.

Compress data

Athena supports a wide range of compression formats. Querying compressed data is faster and also cheaper because you pay for the number of bytes scanned before decompression.

The gzip format provides good compression ratios and has wide range support across other tools and services. The zstd (Zstandard) format is a newer compression format with a good balance between performance and compression ratio.

When compressing text files such as JSON and CSV data, try to achieve a balance between the number of files and the size of the files. Most compression formats require the reader to read files from the beginning. This means that compressed text files cannot, in general, be processed in parallel. Big uncompressed files are often split between workers to achieve higher parallelism during query processing, but this is not possible with most compression formats.

As discussed in Avoid having too many files, it's better to have neither too many files nor too few. Because the number of files is the limit for how many workers can process the query, this rule is especially true for compressed files.

For more information about using compression in Athena, see Use compression in Athena.

Use bucketing for lookups on keys with high cardinality

Bucketing is a technique for distributing records into separate files based on the value of one of the columns. This ensures that all records with the same value will be in the same file. Bucketing is useful when you have a key with high cardinality and many of your queries look up specific values of the key.

For example, suppose you query a set of records for a specific user. If the data is bucketed by user ID, Athena knows in advance which files contain records for a specific ID and which files do not. This enables Athena to read only the files that can contain the ID, greatly reducing the amount of data read. It also reduces the compute time that otherwise would be required to search through the data for the specific ID.

Avoid bucketing when queries frequently search for multiple values in a column

Bucketing is less valuable when queries frequently search for multiple values in the column that the data is bucketed by. The more values queried, the higher the likelihood that all or most files will have to be read. For example, if you have three buckets, and a query looks for three different values, all files might have to be read. Bucketing works best when queries look up single values.

For more information, see Use partitioning and bucketing.

Avoid having too many files

Datasets that consist of many small files result in poor overall query performance. When Athena plans a query, it lists all partition locations, which takes time. Handling and requesting each file also has a computational overhead. Therefore, loading a single bigger file from Amazon S3 is faster than loading the same records from many smaller files.

In extreme cases, you might encounter Amazon S3 service limits. Amazon S3 supports up to 5,500 requests per second to a single index partition. Initially, a bucket is treated as a single index partition, but as request loads increase, it can be split into multiple index partitions.

Amazon S3 looks at request patterns and splits based on key prefixes. If your dataset consists of many thousands of files, the requests coming from Athena can exceed the request quota. Even with fewer files, the quota can be exceeded if multiple concurrent queries are made against the same dataset. Other applications that access the same files can contribute to the total number of requests.

When the request rate `limit` is exceeded, Amazon S3 returns the following error. This error is included in the status information for the query in Athena.

SlowDown: Please reduce your request rate

To troubleshoot, start by determining if the error is caused by a single query or by multiple queries that read the same files. If the latter, coordinate the running of queries so that they don't run at the same time. To achieve this, add a queuing mechanism or even retries in your application.

If running a single query triggers the error, try combining data files or modifying the query to read fewer files. The best time to combine small files is before they are written. To do so, consider the following techniques:

- Change the process that writes the files to write larger files. For example, you could buffer records for a longer time before they are written.
- Put files in a location on Amazon S3 and use a tool like Glue ETL to combine them into larger files. Then, move the larger files into the location that the table points to. For more information, see Reading input files in larger groups in the *AWS Glue Developer Guide* and How can I configure an AWS Glue ETL job to output larger files? in the *AWS re:Post Knowledge Center*.

- Reduce the number of partition keys. When you have too many partition keys, each partition might have only a few records, resulting in an excessive number of small files. For information about deciding which partitions to create, see Pick partition keys that will support your queries.

Avoid additional storage hierarchies beyond the partition

To avoid query planning overhead, store files in a flat structure in each partition location. Do not use any additional directory hierarchies.

When Athena plans a query, it lists all files in all partitions matched by the query. Although Amazon S3 doesn't have directories per se, the convention is to interpret the / forward slash as a directory separator. When Athena lists partition locations, it recursively lists any directory it finds. When files within a partition are organized into a hierarchy, multiple rounds of listings occur.

When all files are directly in the partition location, most of the time only one list operation has to be performed. However, multiple sequential list operations are required if you have more than 1000 files in a partition because Amazon S3 returns only 1000 objects per list operation. Having more than 1000 files in a partition can also create other, more serious performance issues. For more information, see Avoid having too many files.

Use SymlinkTextInputFormat only when necessary

Using the SymlinkTextInputFormat technique can be a way to work around situations when the files for a table are not neatly organized into partitions. For example, symlinks can be useful when all files are in the same prefix or files with different schemas are in the same location.

However, using symlinks adds levels of indirection to the query execution. These levels of indirection impact overall performance. The symlink files have to be read, and the locations they define have to be listed. This adds multiple round trips to Amazon S3 that usual Hive tables do not require. In conclusion, you should use `SymlinkTextInputFormat` only when better options like reorganizing files are not available.

Use columnar storage formats

Apache Parquet and ORC are columnar storage formats that are optimized for fast retrieval of data and used in AWS analytical applications.

Columnar storage formats have the following characteristics that make them suitable for using with Athena:

- *Compression by column, with compression algorithm selected for the column data type* to save storage space in Amazon S3 and reduce disk space and I/O during query processing.
- *Predicate pushdown* in Parquet and ORC enables Athena queries to fetch only the blocks it needs, improving query performance. When an Athena query obtains specific column values from your data, it uses statistics from data block predicates, such as max/min values, to determine whether to read or skip the block.
- *Splitting of data* in Parquet and ORC allows Athena to split the reading of data to multiple readers and increase parallelism during its query processing.

To convert your existing raw data from other storage formats to Parquet or ORC, you can run CREATE TABLE AS SELECT (CTAS) queries in Athena and specify a data storage format as Parquet or ORC, or use the AWS Glue Crawler.

Choose between Parquet and ORC

The choice between ORC (Optimized Row Columnar) and Parquet depends on your specific usage requirements.

Apache Parquet provides efficient data compression and encoding schemes and is ideal for running complex queries and processing large amounts of data. Parquet is optimized for use with Apache Arrow, which can be advantageous if you use tools that are Arrow related.

ORC provides an efficient way to store Hive data. ORC files are often smaller than Parquet files, and ORC indexes can make querying faster. In addition, ORC supports complex types such as structs, maps, and lists.

When choosing between Parquet and ORC, consider the following:

Query performance – Because Parquet supports a wider range of query types, Parquet might be a better choice if you plan to perform complex queries.

Complex data types – If you are using complex data types, ORC might be a better choice as it supports a wider range of complex data types.

File size – If disk space is a concern, ORC usually results in smaller files, which can reduce storage costs.

Compression – Both Parquet and ORC provide good compression, but the best format for you can depend on your specific use case.

Evolution – Both Parquet and ORC support schema evolution, which means you can add, remove, or modify columns over time.

Both Parquet and ORC are good choices for big data applications, but consider the requirements of your scenario before choosing. You might want to perform benchmarks on your data and queries to see which format performs better for your use case.

Convert to columnar formats

Options for easily converting source data such as JSON or CSV into a columnar format include using CREATE TABLE AS queries or running jobs in AWS Glue.

- You can use `CREATE TABLE AS` (CTAS) queries to convert data into Parquet or ORC in one step. For an example, see Example: Writing query results to a different format on the Examples of CTAS queries page.
- For information about using Athena for ETL to transform data from CSV to Parquet, see Use CTAS and INSERT INTO for ETL and data analysis.
- For information about running an AWS Glue job to transform CSV data to Parquet, see the section "Transform the data from CSV to Parquet format" in the AWS Big Data blog post Build a data lake foundation with AWS Glue and Amazon S3. AWS Glue supports using the same technique to convert CSV data to ORC, or JSON data to either Parquet or ORC.

Use partitioning and bucketing

Partitioning and bucketing are two ways to reduce the amount of data Athena must scan when you run a query. Partitioning and bucketing are complementary and can be used together. Reducing the amount of data scanned leads to improved performance and lower cost. For general guidelines about Athena query performance, see Top 10 performance tuning tips for Amazon Athena.

Topics

What is partitioning?

Partitioning means organizing data into directories (or "prefixes") on Amazon S3 based on a particular property of the data. Such properties are called *partition keys*. A common partition key is the date or some other unit of time such as the year or month. However, a dataset can be partitioned by more than one key. For example, data about product sales might be partitioned by date, product category, and market.

Deciding how to partition

Good candidates for partition keys are properties that are always or frequently used in queries and have low cardinality. There is a trade-off between having too many partitions and having too few. With too many partitions, the increased number of files creates overhead. There is also some overhead from filtering the partitions themselves. With too few partitions, queries often have to scan more data.

Create a partitioned table

When a dataset is partitioned, you can create a partitioned table in Athena. A partitioned table is a table that has partition keys. When you use `CREATE TABLE`, you add partitions to the table. When you use `CREATE TABLE AS`, the partitions that are created on Amazon S3 are automatically added to the table.

In a `CREATE TABLE` statement, you specify the partition keys in the `PARTITIONED BY (column_name data_type)` clause. In a `CREATE TABLE AS` statement, you specify the partition keys in a `WITH (partitioned_by = ARRAY['partition_key'])` clause, or `WITH (partitioning = ARRAY['partition_key'])` for Iceberg tables. For performance reasons, partition keys should always be of type `STRING`. For more information, see Use string as the data type for partition keys.

For additional `CREATE TABLE` and `CREATE TABLE AS` syntax details, see CREATE TABLE and CTAS table properties.

Query partitioned tables

When you query a partitioned table, Athena uses the predicates in the query to filter the list of partitions. Then it uses the locations of the matching partitions to process the files found. Athena can efficiently reduce the amount of data scanned by simply not reading data in the partitions that don't match the query predicates.

Examples

Suppose you have a table partitioned by `sales_date` and `product_category` and want to know the total revenue over a week in a specific category. You include predicates on the `sales_date` and `product_category` columns to ensure that Athena scans only the minimum amount of data, as in the following example.

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND product_category = 'Toys'
```

Suppose you have a dataset that is partitioned by date but also has a fine-grained timestamp.

With Iceberg tables, you can declare a partition key to have a relationship to a column, but with Hive tables the query engine has no knowledge of relationships between columns and partition keys. For this reason, you must include a predicate on both the column and the partition key in your query to make sure the query does not scan more data than necessary.

For example, suppose the `sales` table in the previous example also has a `sold_at` column of the `TIMESTAMP` data type. If you want the revenue only for a specific time range, you would write the query like this:

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date = '2023-02-28'
AND sold_at BETWEEN TIMESTAMP '2023-02-28 10:00:00' AND TIMESTAMP '2023-02-28
 12:00:00'
AND product_category = 'Toys'
```

For more information about this difference between querying Hive and Iceberg tables, see [How to write queries for timestamp fields that are also time-partitioned].

What is bucketing?

Bucketing is a way to organize the records of a dataset into categories called *buckets*.

This meaning of *bucket* and *bucketing* is different from, and should not be confused with, Amazon S3 buckets. In data bucketing, records that have the same value for a property go into the same bucket. Records are distributed as evenly as possible among buckets so that each bucket has roughly the same amount of data.

In practice, the buckets are files, and a hash function determines the bucket that a record goes into. A bucketed dataset will have one or more files per bucket per partition. The bucket that a file belongs to is encoded in the file name.

Bucketing benefits

Bucketing is useful when a dataset is bucketed by a certain property and you want to retrieve records in which that property has a certain value. Because the data is bucketed, Athena can use the value to determine which files to look at. For example, suppose a dataset is bucketed by `customer_id` and you want to find all records for a specific customer. Athena determines the bucket that contains those records and only reads the files in that bucket.

Good candidates for bucketing occur when you have columns that have high cardinality (that is, have many distinct values), are uniformly distributed, and that you frequently query for specific values.

> **Note**
>
> Athena does not support using `INSERT INTO` to add new records to bucketed tables.

Data types supported for filtering on bucketed columns

You can add filters on bucketed columns with certain data types. Athena supports filtering on bucketed columns with the following data types:

- BOOLEAN
- BYTE
- DATE
- DOUBLE
- FLOAT
- INT
- LONG
- SHORT
- STRING
- VARCHAR

Hive and Spark support

Athena engine version 2 supports datasets bucketed using the Hive bucket algorithm, and Athena engine version 3 also supports the Apache Spark bucketing algorithm. Hive bucketing is the default. If your dataset is bucketed using the Spark algorithm, use the TBLPROPERTIES clause to set the bucketing_format property value to spark.

> **Note**
>
> Athena has a limit of 100 partitions in a CREATE TABLE AS SELECT (CTAS) query. Similarly, you can add only a maximum of 100 partitions to a destination table with an INSERT INTO statement.
>
> If you exceed this limitation, you may receive the error message HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets. To work around this limitation, you can use a CTAS statement and a series of INSERT INTO statements that create or insert up to 100 partitions each. For more information, see Use CTAS and INSERT INTO to work around the 100 partition limit.

Bucketing CREATE TABLE example

To create a table for an existing bucketed dataset, use the CLUSTERED BY (*column*) clause followed by the INTO *N* BUCKETS clause. The INTO *N* BUCKETS clause specifies the number of buckets the data is bucketed into.

In the following CREATE TABLE example, the sales dataset is bucketed by customer_id into 8 buckets using the Spark algorithm. The CREATE TABLE statement uses the CLUSTERED BY and TBLPROPERTIES clauses to set the properties accordingly.

```
CREATE EXTERNAL TABLE sales (...)
...
CLUSTERED BY (`customer_id`) INTO 8 BUCKETS
...
TBLPROPERTIES (
  'bucketing_format' = 'spark'
)
```

Bucketing CREATE TABLE AS (CTAS) example

To specify bucketing with CREATE TABLE AS, use the bucketed_by and bucket_count parameters, as in the following example.

```
CREATE TABLE sales
WITH (
```

```
  ...
  bucketed_by = ARRAY['customer_id'],
  bucket_count = 8
)
AS SELECT ...
```

Bucketing query example

The following example query looks for the names of products that a specific customer has purchased over the course of a week.

```
SELECT DISTINCT product_name
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND customer_id = 'c123'
```

If this table is partitioned by `sales_date` and bucketed by `customer_id`, Athena can calculate the bucket that the customer records are in. At most, Athena reads one file per partition.

Additional resources

- For a `CREATE TABLE AS` example that creates both bucketed and partitioned tables, see Example: Creating bucketed and partitioned tables.
- For information on implementing bucketing on AWS data lakes, including using an Athena CTAS statement, AWS Glue for Apache Spark, and bucketing for Apache Iceberg tables, see the AWS Big Data Blog post Optimize data layout by bucketing with Amazon Athena and AWS Glue to accelerate downstream queries.

Partition your data

By partitioning your data, you can restrict the amount of data scanned by each query, thus improving performance and reducing cost. You can partition your data by any key. A common practice is to partition the data based on time, often leading to a multi-level partitioning scheme. For example, a customer who has data coming in every hour might decide to partition by year, month, date, and hour. Another customer, who has data coming from many different sources but that is loaded only once per day, might partition by a data source identifier and date.

Athena can use Apache Hive style partitions, whose data paths contain key value pairs connected by equal signs (for example, `country=us/...` or `year=2021/month=01/day=26/...`). Thus, the paths include both the names of the partition keys and the values that each path represents. To load new Hive partitions into a partitioned table, you can use the MSCK REPAIR TABLE command, which works only with Hive-style partitions.

Athena can also use non-Hive style partitioning schemes. For example, CloudTrail logs and Firehose delivery streams use separate path components for date parts such as `data/2021/01/26/us/6fc7845e.json`. For such non-Hive style partitions, you use ALTER TABLE ADD PARTITION to add the partitions manually.

Considerations and limitations

When using partitioning, keep in mind the following points:

- If you query a partitioned table and specify the partition in the `WHERE` clause, Athena scans the data only from that partition.
- If you issue queries against Amazon S3 buckets with a large number of objects and the data is not partitioned, such queries may affect the `GET` request rate limits in Amazon S3 and lead to Amazon S3 exceptions. To prevent errors, partition your data. Additionally, consider tuning your Amazon S3 request rates. For more information, see Best practices design patterns: Optimizing Amazon S3 performance .
- Partition locations to be used with Athena must use the `s3` protocol (for example, `s3://amzn-s3-demo-bucket/folder/`). In Athena, locations that use other protocols (for example, `s3a://amzn-s3-demo-bucket/folder/`) will result in query failures when `MSCK REPAIR TABLE` queries are run on the containing tables.
- Make sure that the Amazon S3 path is in lower case instead of camel case (for example, `userid` instead of `userId`). If the S3 path is in camel case, `MSCK REPAIR TABLE` doesn't add the partitions to the AWS Glue Data Catalog. For more information, see MSCK REPAIR TABLE.
- Because `MSCK REPAIR TABLE` scans both a folder and its subfolders to find a matching partition scheme, be sure to keep data for separate tables in separate folder hierarchies. For example, suppose you have data for table 1 in `s3://amzn-s3-demo-bucket1` and data for table 2 in `s3://amzn-s3-demo-bucket1/table-2-data`. If both tables are partitioned by string, `MSCK REPAIR TABLE` will add the partitions for table 2 to table 1. To avoid this, use separate folder structures like `s3://amzn-s3-demo-bucket1` and `s3://amzn-s3-demo-bucket2` instead. Note that this behavior is consistent with Amazon EMR and Apache Hive.
- If you are using the AWS Glue Data Catalog with Athena, see AWS Glue endpoints and quotas for service quotas on partitions per account and per table.
  - Although Athena supports querying AWS Glue tables that have 10 million partitions, Athena cannot read more than 1 million partitions in a single scan. In such scenarios, partition

indexing can be beneficial. For more information, see the AWS Big Data Blog article [Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes].

- To request a partitions quota increase if you are using the AWS Glue Data Catalog, visit the [Service Quotas console for AWS Glue].

Create and load a table with partitioned data

To create a table that uses partitions, use the `PARTITIONED BY` clause in your [CREATE TABLE] statement. The `PARTITIONED BY` clause defines the keys on which to partition data, as in the following example. The `LOCATION` clause specifies the root location of the partitioned data.

```
CREATE EXTERNAL TABLE users (
first string,
last string,
username string
)
PARTITIONED BY (id string)
STORED AS parquet
LOCATION 's3://amzn-s3-demo-bucket'
```

After you create the table, you load the data in the partitions for querying. For Hive style partitions, you run [MSCK REPAIR TABLE]. For non-Hive style partitions, you use [ALTER TABLE ADD PARTITION] to add the partitions manually.

Prepare Hive style and non-Hive style data for querying

The following sections show how to prepare Hive style and non-Hive style data for querying in Athena.

Scenario 1: Data stored on Amazon S3 in Hive format

In this scenario, partitions are stored in separate folders in Amazon S3. For example, here is the partial listing for sample ad impressions output by the [`aws s3 ls`] command, which lists the S3 objects under a specified prefix:

```
aws s3 ls s3://elasticmapreduce/samples/hive-ads/tables/impressions/

    PRE dt=2009-04-12-13-00/
    PRE dt=2009-04-12-13-05/
```

```
    PRE dt=2009-04-12-13-10/
    PRE dt=2009-04-12-13-15/
    PRE dt=2009-04-12-13-20/
    PRE dt=2009-04-12-14-00/
    PRE dt=2009-04-12-14-05/
    PRE dt=2009-04-12-14-10/
    PRE dt=2009-04-12-14-15/
    PRE dt=2009-04-12-14-20/
    PRE dt=2009-04-12-15-00/
    PRE dt=2009-04-12-15-05/
```

Here, logs are stored with the column name (dt) set equal to date, hour, and minute increments. When you give a DDL with the location of the parent folder, the schema, and the name of the partitioned column, Athena can query data in those subfolders.

Create the table

To make a table from this data, create a partition along 'dt' as in the following Athena DDL statement:

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string,
    hostname string,
    sessionId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://elasticmapreduce/samples/hive-ads/tables/impressions/' ;
```

This table uses Hive's native JSON serializer-deserializer to read JSON data stored in Amazon S3. For more information about the formats supported, see Choose a SerDe for your data.

Run MSCK REPAIR TABLE

After you run the CREATE TABLE query, run the MSCK REPAIR TABLE command in the Athena query editor to load the partitions, as in the following example.

```
MSCK REPAIR TABLE impressions
```

After you run this command, the data is ready for querying.

Query the data

Query the data from the impressions table using the partition column. Here's an example:

```
SELECT dt,impressionid FROM impressions WHERE dt<'2009-04-12-14-00' and
 dt>='2009-04-12-13-00' ORDER BY dt DESC LIMIT 100
```

This query should show results similar to the following:

```
2009-04-12-13-20    ap3HcVKAWfXtgIPu6WpuUfAfL0DQEc
2009-04-12-13-20    17uchtodoS9kdeQP1x0XThKl5IuRsV
2009-04-12-13-20    JOUf1SCtRwviGw8sVcghqE5h0nkgtp
2009-04-12-13-20    NQ2XP0J0dvVbCXJ0pb4XvqJ5A4QxxH
2009-04-12-13-20    fFAItiBMsgqro9kRdIwbeX60SROaxr
2009-04-12-13-20    V4og4R9W6G3QjHHwF7gI1cSqig5D1G
2009-04-12-13-20    hPEPtBwk45msmwWTxPVVo1kVu4v11b
2009-04-12-13-20    v0SkfxegheD90gp31UCr6FplnKpx6i
2009-04-12-13-20    1iD9odVgOIi4QWkwHMcOhmwTkWDKfj
2009-04-12-13-20    b31tJiIA25CK8eDHQrHnbcknfSndUk
```

Scenario 2: Data is not partitioned in Hive format

In the following example, the aws s3 ls command shows [ELB](https://) logs stored in Amazon S3. Note how the data layout does not use key=value pairs and therefore is not in Hive format. (The --recursive option for the aws s3 ls command specifies that all files or objects under the specified directory or prefix be listed.)

```
aws s3 ls s3://athena-examples-myregion/elb/plaintext/ --recursive

2016-11-23 17:54:46   11789573 elb/plaintext/2015/01/01/part-r-00000-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    8776899 elb/plaintext/2015/01/01/part-r-00001-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:46    9309800 elb/plaintext/2015/01/01/part-r-00002-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9412570 elb/plaintext/2015/01/01/part-r-00003-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47   10725938 elb/plaintext/2015/01/01/part-r-00004-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9439710 elb/plaintext/2015/01/01/part-r-00005-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47          0 elb/plaintext/2015/01/01_$folder$
2016-11-23 17:54:47    9012723 elb/plaintext/2015/01/02/part-r-00006-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    7571816 elb/plaintext/2015/01/02/part-r-00007-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9673393 elb/plaintext/2015/01/02/part-r-00008-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11979218 elb/plaintext/2015/01/02/part-r-00009-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    9546833 elb/plaintext/2015/01/02/part-r-00010-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   10960865 elb/plaintext/2015/01/02/part-r-00011-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48          0 elb/plaintext/2015/01/02_$folder$
2016-11-23 17:54:48   11360522 elb/plaintext/2015/01/03/part-r-00012-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11211291 elb/plaintext/2015/01/03/part-r-00013-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    8633768 elb/plaintext/2015/01/03/part-r-00014-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11891626 elb/plaintext/2015/01/03/part-r-00015-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49    9173813 elb/plaintext/2015/01/03/part-r-00016-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11899582 elb/plaintext/2015/01/03/part-r-00017-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49          0 elb/plaintext/2015/01/03_$folder$
2016-11-23 17:54:50    8612843 elb/plaintext/2015/01/04/part-r-00018-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50   10731284 elb/plaintext/2015/01/04/part-r-00019-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9984735 elb/plaintext/2015/01/04/part-r-00020-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9290089 elb/plaintext/2015/01/04/part-r-00021-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:50    7896339 elb/plaintext/2015/01/04/part-r-00022-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8321364 elb/plaintext/2015/01/04/part-r-00023-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/04_$folder$
2016-11-23 17:54:51    7641062 elb/plaintext/2015/01/05/part-r-00024-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   10253377 elb/plaintext/2015/01/05/part-r-00025-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8502765 elb/plaintext/2015/01/05/part-r-00026-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   11518464 elb/plaintext/2015/01/05/part-r-00027-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7945189 elb/plaintext/2015/01/05/part-r-00028-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7864475 elb/plaintext/2015/01/05/part-r-00029-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/05_$folder$
2016-11-23 17:54:51   11342140 elb/plaintext/2015/01/06/part-r-00030-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8063755 elb/plaintext/2015/01/06/part-r-00031-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9387508 elb/plaintext/2015/01/06/part-r-00032-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9732343 elb/plaintext/2015/01/06/part-r-00033-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11510326 elb/plaintext/2015/01/06/part-r-00034-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9148117 elb/plaintext/2015/01/06/part-r-00035-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52          0 elb/plaintext/2015/01/06_$folder$
2016-11-23 17:54:52    8402024 elb/plaintext/2015/01/07/part-r-00036-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    8282860 elb/plaintext/2015/01/07/part-r-00037-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11575283 elb/plaintext/2015/01/07/part-r-00038-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53    8149059 elb/plaintext/2015/01/07/part-r-00039-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10037269 elb/plaintext/2015/01/07/part-r-00040-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10019678 elb/plaintext/2015/01/07/part-r-00041-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53          0 elb/plaintext/2015/01/07_$folder$
```

```
2016-11-23 17:54:53          0 elb/plaintext/2015/01_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015_$folder$
```

Run ALTER TABLE ADD PARTITION

Because the data is not in Hive format, you cannot use the `MSCK REPAIR TABLE` command to add the partitions to the table after you create it. Instead, you can use the ALTER TABLE ADD PARTITION command to add each partition manually. For example, to load the data in s3://athena-examples-*myregion*/elb/plaintext/2015/01/01/, you can run the following query. Note that a separate partition column for each Amazon S3 folder is not required, and that the partition key value can be different from the Amazon S3 key.

```
ALTER TABLE elb_logs_raw_native_part ADD PARTITION (dt='2015-01-01') location 's3://
athena-examples-us-west-1/elb/plaintext/2015/01/01/'
```

If a partition already exists, you receive the error Partition already exists. To avoid this error, you can use the `IF NOT EXISTS` clause. For more information, see ALTER TABLE ADD PARTITION. To remove a partition, you can use ALTER TABLE DROP PARTITION.

Consider partition projection

To avoid having to manage partitions yourself, you can use partition projection. Partition projection is an option for highly partitioned tables whose structure is known in advance. In partition projection, partition values and locations are calculated from table properties that you configure rather than read from a metadata repository. Because the in-memory calculations are faster than remote look-up, the use of partition projection can significantly reduce query runtimes.

For more information, see Use partition projection with Amazon Athena.

Additional resources

- For information about partitioning options for Firehose data, see Amazon Data Firehose example.
- You can automate adding partitions by using the JDBC driver.
- You can use CTAS and INSERT INTO to partition a dataset. For more information, see Use CTAS and INSERT INTO for ETL and data analysis.

Use partition projection with Amazon Athena

You can use partition projection in Athena to speed up query processing of highly partitioned tables and automate partition management.

In partition projection, Athena calculates partition values and locations using the table properties that you configure directly on your table in AWS Glue. The table properties allow Athena to 'project', or determine, the necessary partition information instead of having to do a more time-consuming metadata lookup in the AWS Glue Data Catalog. Because in-memory operations are often faster than remote operations, partition projection can reduce the runtime of queries against highly partitioned tables. Depending on the specific characteristics of the query and underlying data, partition projection can significantly reduce query runtime for queries that are constrained on partition metadata retrieval.

Understand partition pruning vs. partition projection

Partition pruning gathers metadata and "prunes" it to only the partitions that apply to your query. This often speeds up queries. Athena uses partition pruning for all tables with partition columns, including those tables configured for partition projection.

Normally, when processing queries, Athena makes a `GetPartitions` call to the AWS Glue Data Catalog before performing partition pruning. If a table has a large number of partitions, using `GetPartitions` can affect performance negatively. To avoid this, you can use partition projection. Partition projection allows Athena to avoid calling `GetPartitions` because the partition projection configuration gives Athena all of the necessary information to build the partitions itself.

How to use partition projection

To use partition projection, you specify the ranges of partition values and projection types for each partition column in the table properties in the AWS Glue Data Catalog or in your external Hive metastore. These custom properties on the table allow Athena to know what partition patterns to expect when it runs a query on the table. During query execution, Athena uses this information to project the partition values instead of retrieving them from the AWS Glue Data Catalog or external Hive metastore. This not only reduces query execution time but also automates partition management because it removes the need to manually create partitions in Athena, AWS Glue, or your external Hive metastore.

> **Important**
>
> Enabling partition projection on a table causes Athena to ignore any partition metadata registered to the table in the AWS Glue Data Catalog or Hive metastore.

Some use cases

Scenarios in which partition projection is useful include the following:

- Queries against a highly partitioned table do not complete as quickly as you would like.
- You regularly add partitions to tables as new date or time partitions are created in your data. With partition projection, you configure relative date ranges that can be used as new data arrives.
- You have highly partitioned data in Amazon S3. The data is impractical to model in your AWS Glue Data Catalog or Hive metastore, and your queries read only small parts of it.

Projectable partition structures

Partition projection is most easily configured when your partitions follow a predictable pattern such as, but not limited to, the following:

- **Integers** – Any continuous sequence of integers such as `[1, 2, 3, 4, ..., 1000]` or `[0500, 0550, 0600, ..., 2500]`.
- **Dates** – Any continuous sequence of dates or datetimes such as `[20200101, 20200102, ..., 20201231]` or `[1-1-2020 00:00:00, 1-1-2020 01:00:00, ..., 12-31-2020 23:00:00]`.
- **Enumerated values** – A finite set of enumerated values such as airport codes or AWS Regions.
- **AWS service logs** – AWS service logs typically have a known structure whose partition scheme you can specify in AWS Glue and that Athena can therefore use for partition projection.

How to customize the partition path template

By default, Athena builds partition locations using the form `s3://amzn-s3-demo-bucket/`*`<table-root>`*`/partition-col-1=`*`<partition-col-1-val>`*`/partition-col-2=`*`<partition-col-2-val>`*`/`, but if your data is organized differently, Athena offers a mechanism for customizing this path template. For steps, see [How to specify custom S3 storage locations](#).

Considerations and limitations

The following considerations apply:

- Partition projection eliminates the need to specify partitions manually in AWS Glue or an external Hive metastore.
- When you enable partition projection on a table, Athena ignores any partition metadata in the AWS Glue Data Catalog or external Hive metastore for that table.
- If a projected partition does not exist in Amazon S3, Athena will still project the partition. Athena does not throw an error, but no data is returned. However, if too many of your partitions are empty, performance can be slower compared to traditional AWS Glue partitions. If more than half of your projected partitions are empty, it is recommended that you use traditional partitions.
- Queries for values that are beyond the range bounds defined for partition projection do not return an error. Instead, the query runs, but returns zero rows. For example, if you have time-related data that starts in 2020 and is defined as `'projection.timestamp.range'='2020/01/01,NOW'`, a query like `SELECT * FROM table-name WHERE timestamp = '2019/02/02'` will complete successfully, but return zero rows.
- Partition projection is usable only when the table is queried through Athena. If the same table is read through another service such as Amazon Redshift Spectrum, Athena for Spark, or Amazon EMR, the standard partition metadata is used.
- Because partition projection is a DML-only feature, `SHOW PARTITIONS` does not list partitions that are projected by Athena but not registered in the AWS Glue catalog or external Hive metastore.
- Athena does not use the table properties of views as configuration for partition projection. To work around this limitation, configure and enable partition projection in the table properties for the tables that the views reference.

Video

The following video shows how to use partition projection to improve the performance of your queries in Athena.

[Partition projection with Amazon Athena](#)

Topics

- Set up partition projection
- Supported types for partition projection
- Use dynamic ID partitioning
- Amazon Data Firehose example

Set up partition projection

Setting up partition projection in a table's properties is a two-step process:

1. Specify the data ranges and relevant patterns for each partition column, or use a custom template.
2. Enable partition projection for the table.

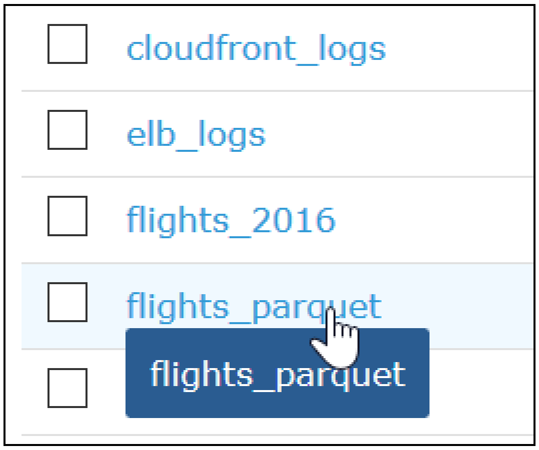
> **Note**
>
> Before you add partition projection properties to an existing table, the partition column for which you are setting up partition projection properties must already exist in the table schema. If the partition column does not yet exist, you must add a partition column to the existing table manually. AWS Glue does not perform this step for you automatically.

This section shows how to set the table properties for AWS Glue. To set them, you can use the AWS Glue console, Athena CREATE TABLE queries, or AWS Glue API operations. The following procedure shows how to set the properties in the AWS Glue console.

To configure and enable partition projection using the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. Choose the **Tables** tab.

   On the **Tables** tab, you can edit existing tables, or choose **Add tables** to create new ones. For information about adding tables manually or with a crawler, see Working with tables on the AWS Glue console in the *AWS Glue Developer Guide*.
3. In the list of tables, choose the link for the table that you want to edit.



4. Choose **Actions**, **Edit table**.

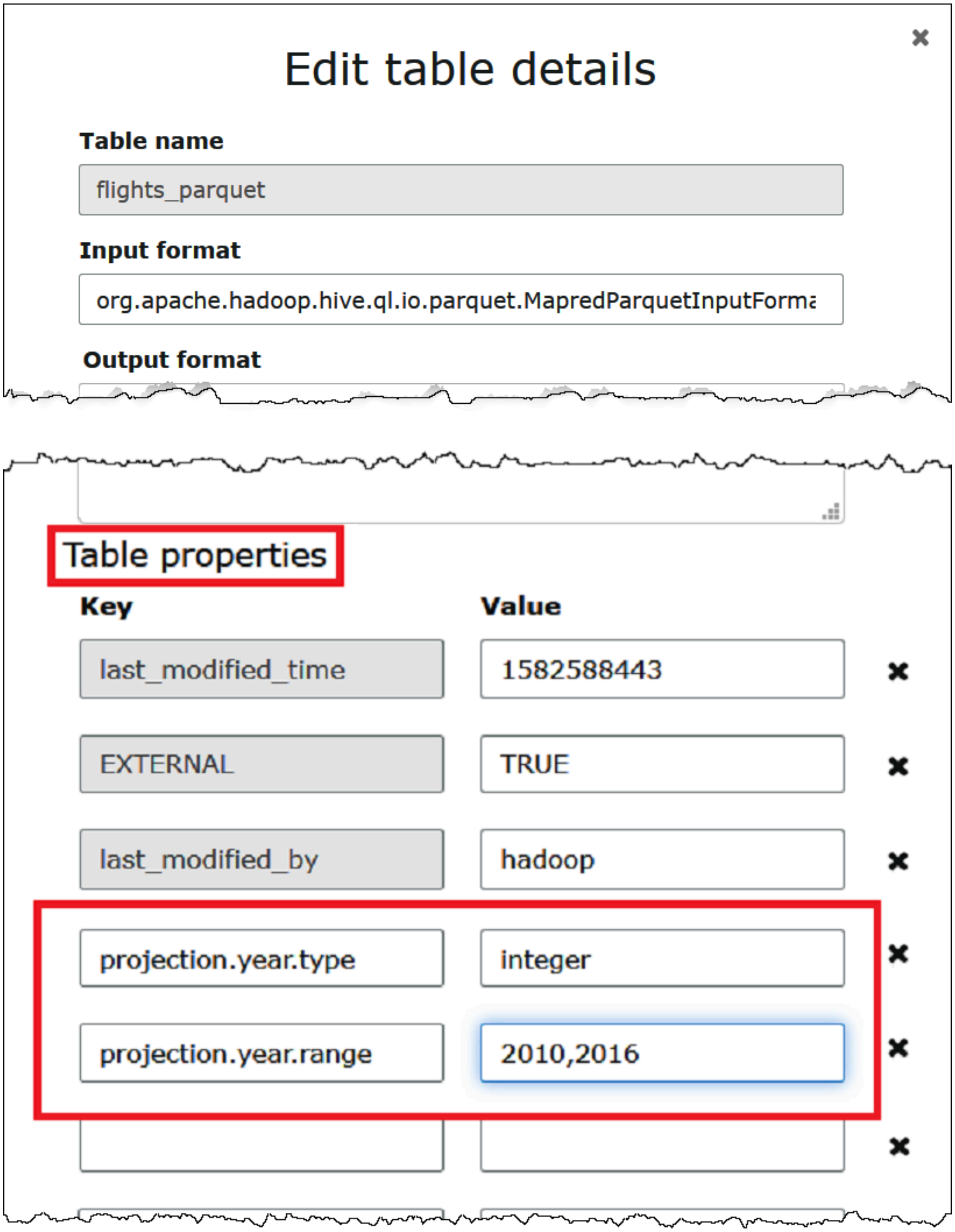
5. On the **Edit table** page, in the **Table properties** section, for each partitioned column, add the following key-value pair:

   a. For **Key**, add `projection.`*`columnName`*`.type`.

   b. For **Value**, add one of the supported types: `enum`, `integer`, `date`, or `injected`. For more information, see [Supported types for partition projection](#).

6. Following the guidance in [Supported types for partition projection](#), add additional key-value pairs according to your configuration requirements.

   The following example table configuration configures the `year` column for partition projection, restricting the values that can be returned to a range from 2010 through 2016.

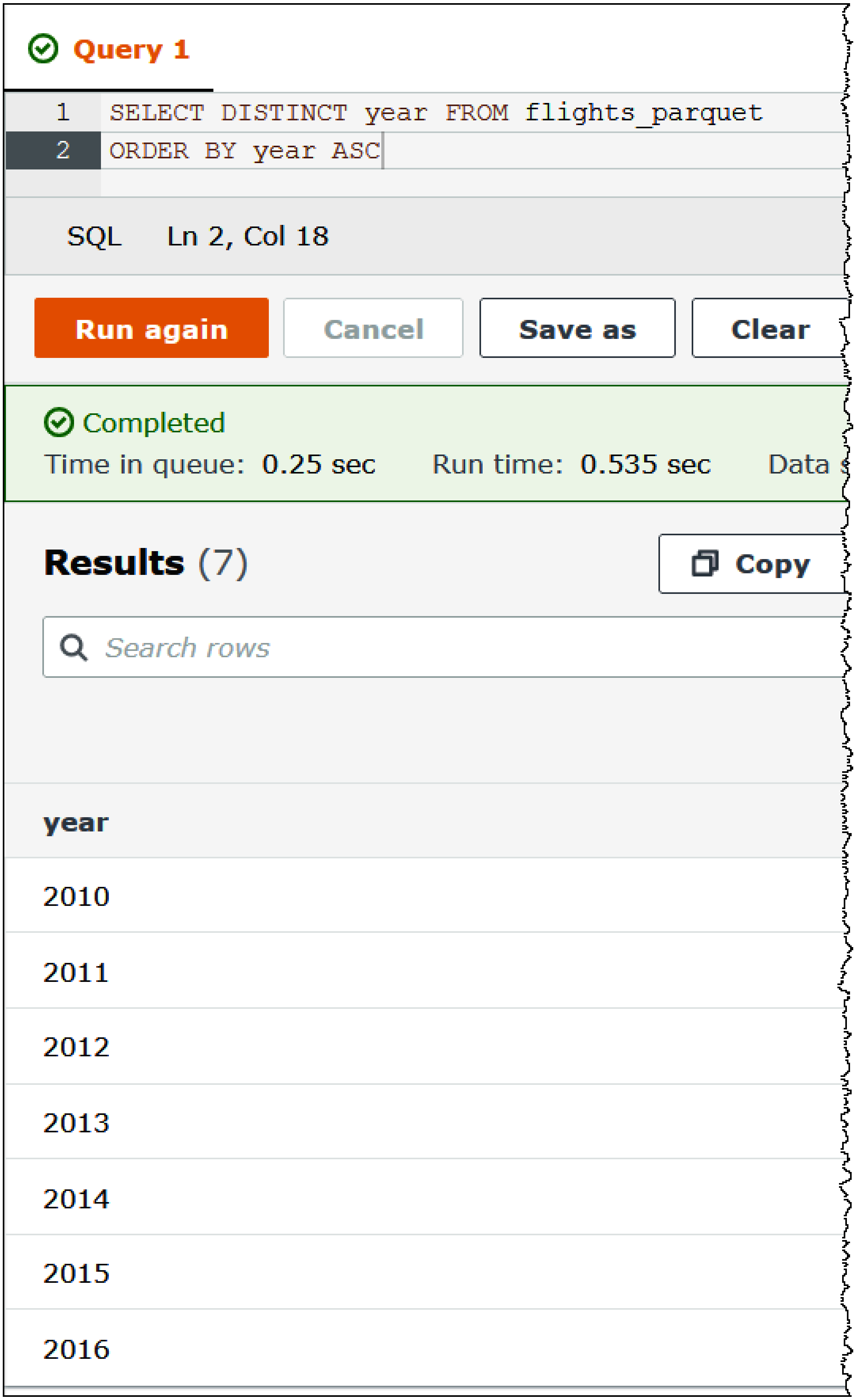


7. Add a key-value pair to enable partition projection. For **Key**, enter `projection.enabled`, and for its **Value**, enter `true`.

> **Note**
>
> You can disable partition projection on this table at any time by setting `projection.enabled` to `false`.

8. When you are finished, choose **Save**.
9. In the Athena Query Editor, test query the columns that you configured for the table.

   The following example query uses `SELECT DISTINCT` to return the unique values from the `year` column. The database contains data from 1987 to 2016, but the `projection.year.range` property restricts the values returned to the years 2010 to 2016.


Query 1
1 SELECT DISTINCT year FROM flights_parquet
2 ORDER BY year ASC
SQL Ln 2, Col 18
Run again
Cancel
Save as
Clear
Completed
Time in queue: 0.25 sec
Run time: 0.535 sec
Data
Results (7)
Copy
Search rows
year
2010
2011
2012
2013
2014
2015
2016

> **Note**
>
> If you set `projection.enabled` to `true` but fail to configure one or more partition columns, you receive an error message like the following:
> `HIVE_METASTORE_ERROR: Table *database_name*.*table_name* is configured for partition projection, but the following partition columns are missing projection configuration: [*column_name*] (table *database_name*.*table_name*).`

How to specify custom S3 storage locations

When you edit table properties in AWS Glue, you can also specify a custom Amazon S3 path template for the projected partitions. A custom template enables Athena to properly map partition values to custom Amazon S3 file locations that do not follow a typical `.../column=value/...` pattern.

Using a custom template is optional. However, if you use a custom template, the template must contain a placeholder for each partition column. Templated locations must end with a forward slash so that the partitioned data files live in a "folder" per partition.

To specify a custom partition location template

1. Following the steps to [configure and enable partition projection using the AWS Glue console](), add an additional a key-value pair that specifies a custom template as follows:

   a. For **Key**, enter `storage.location.template`.

   b. For **Value**, specify a location that includes a placeholder for every partition column. Make sure that each placeholder (and the S3 path itself) is terminated by a single forward slash.

      The following example template values assume a table with partition columns a, b, and c.

      ```
      s3://amzn-s3-demo-bucket/table_root/a=${a}/${b}/some_static_subdirectory/${c}/
      ```

      ```
      s3://amzn-s3-demo-bucket/table_root/c=${c}/${b}/some_static_subdirectory/${a}/
      ${b}/${c}/${c}/
      ```

For the same table, the following example template value is invalid because it contains no placeholder for column `c`.

```
s3://amzn-s3-demo-bucket/table_root/a=${a}/${b}/some_static_subdirectory/
```

2. Choose **Apply**.

Supported types for partition projection

A table can have any combination of `enum`, `integer`, `date,` or `injected` partition column types.

Enum type

Use the `enum` type for partition columns whose values are members of an enumerated set (for example, airport codes or AWS Regions).

Define the partition properties in the table as follows:

Property name	Example values	Description
`projection.` *`columnName`* `.type`	`enum`	Required. The projection type to use for column *`columnName`* . The value must be `enum` (case insensitive) to signal the use of the enum type. Leading and trailing white space is allowed.
`projection.` *`columnName`* `.values`	`A,B,C,D,E,F,G,Unknown`	Required. A comma-separated list of enumerated partition values for column *`columnName`* . Any white space is considered part of an enum value.

> **Note**
>
> As a best practice we recommend limiting the use of `enum` based partition projections to a few dozen or less. Although there is no specific limit for `enum` projections, the total size of your table's metadata cannot exceed the AWS Glue limit of about 1 MB when gzip compressed. Note that this limit is shared across key parts of your table like column names, location, storage format, and others. If you find yourself using more than a few dozen unique IDs in your `enum` projection, consider an alternative approach such as bucketing into a smaller number of unique values in a surrogate field. By trading off cardinality, you can control the number of unique values in your `enum` field.

Integer type

Use the integer type for partition columns whose possible values are interpretable as integers within a defined range. Projected integer columns are currently limited to the range of a Java signed long (-2^{63} to $2^{63}-1$ inclusive).

Property name	Example values	Description
`projection.` *`columnName`* `.type`	`integer`	Required. The projection type to use for column *`columnName`* . The value must be `integer` (case insensitive) to signal the use of the integer type. Leading and trailing white space is allowed.

`projection.` *`columnName`* `.range`	`0,10`
	`-1,8675309`
	`0001,9999`
		Required. A two-element comma-separated list that provides the minimum and maximum range values to be returned by queries on the column *`columnName`* . Note that the values must be separated by a comma, not

Property name	Example values	Description
		a hyphen. These values are inclusive, can be negative, and can have leading zeroes. Leading and trailing white space is allowed.
`projection. columnName .interval`	1 5	Optional. A positive integer that specifies the interval between successive partition values for the column *columnName* . For example, a `range` value of "1,3" with an `interval` value of "1" produces the values 1, 2, and 3. The same `range` value with an `interval` value of "2" produces the values 1 and 3, skipping 2. Leading and trailing white space is allowed. The default is 1.

Property name	Example values	Description
projection. *columnName* .digits	1 5	Optional. A positive integer that specifies the number of digits to include in the partition value's final representation for column *columnName* . For example, a `range` value of "1,3" that has a `digits` value of "1" produces the values 1, 2, and 3. The same `range` value with a `digits` value of "2" produces the values 01, 02, and 03. Leading and trailing white space is allowed. The default is no static number of digits and no leading zeroes.

Date type

Use the date type for partition columns whose values are interpretable as dates (with optional times) within a defined range.

> **Important**
>
> Projected date columns are generated in Coordinated Universal Time (UTC) at query execution time.

Property name	Example values	Description
projection. *columnName* .type	date	Required. The projection type to use for column *columnName* . The value must be `date` (case insensitive) to signal the use of the

Property name	Example values	Description
		date type. Leading and trailing white space is allowed.

`projection. columnName .range`	`201701,201812`
	`01-01-2010,12-31-2018`
	`NOW-3YEARS,NOW`
	`201801,NOW+1MONTH`
		Required. A two-element, comma-separated list which provides the minimum and maximum `range` values for the column *columnName* . These values are inclusive and can use any format compatible with the Java `java.time.*` date types. Both the minimum and maximum values must use the same format. The format specified in the `.format` property must be the format used for these values.
		
		This column can also contain relative date strings, formatted in this regular expression pattern:
		
		`\s*NOW\s*(([\+\-])\s*([0-9]+)\s*(YEARS?|MONTHS?|WEEKS?|DAYS?|HOURS?|MINUTES?|SECONDS?)\s*)?`
		
		White spaces are allowed, but in date literals are considered part of the date strings themselves.

`projection. columnName .format`	`yyyyMM`
	`dd-MM-yyyy`
	`dd-MM-yyyy-HH-mm-ss`
		Required. A date format string based on the Java date format [DateTimeFormatter](). Can be any supported `Java.time.*` type.

Property name	Example values	Description
projection. *columnName* .interval	1 5	A positive integer that specifies the interval between successive partition values for column *columnName* . For example, a `range` value of `2017-01,2018-12` with an `interval` value of 1 and an `interval.unit` value of `MONTHS` produces the values 2017-01, 2017-02, 2017-03, and so on. The same `range` value with an `interval` value of 2 and an `interval.unit` value of `MONTHS` produces the values 2017-01, 2017-03, 2017-05, and so on. Leading and trailing white space is allowed. When the provided dates are at single-day or single-month precision, the `interval` is optional and defaults to 1 day or 1 month, respectively. Otherwise, `interval` is required.

projection. *columnName* .interval.unit	`YEARS`
	
	`MONTHS`
	
	`WEEKS`
	
	`DAYS`
	
	`HOURS`
	
	`MINUTES`
	
	`SECONDS`
	
	`MILLIS`
		A time unit word that represents the serialized form of a ChronoUnit. Possible values are `YEARS`, `MONTHS`, `WEEKS`, `DAYS`, `HOURS`, `MINUTES`, `SECONDS`, or `MILLIS`. These values are case insensitive.
		
		When the provided dates are at single-day or single-month precision, the `interval.unit` is optional and defaults to 1 day or 1 month, respectively. Otherwise, the `interval.unit` is required.

Example – Partitioning by month

The following example table configuration partitions data by month from 2015 to the present.

```
'projection.month.type'='date',
'projection.month.format'='yyyy-MM',
'projection.month.interval'='1',
'projection.month.interval.unit'='MONTHS',
'projection.month.range'='2015-01,NOW',
...
```

Injected type

Use the injected type for partition columns with possible values that cannot be procedurally generated within some logical range but that are provided in a query's `WHERE` clause as a single value.

It is important to keep in mind the following points:

- Queries on injected columns fail if a filter expression is not provided for each injected column.
- Queries with multiple values for a filter expression on an injected column succeed only if the values are disjunct.
- Only columns of `string` type are supported.
- When you use the `WHERE IN` clause with an injected partition column, there is a limit of 1,000 values that you can specify in the `IN` list. To query a dataset with more than 1,000 partitions for an injected column, split the query into multiple smaller queries, each with up to 1,000 values in the `WHERE IN` clause, and then aggregate the results.

Property name	Value	Description
`projection.` *`columnName`* `.type`	`inje`	Required. The projection type to use for the column *`columnName`* . Only the `string` type is supported. The value specified must be `injected` (case insensitive). Leading and trailing white space is allowed.

For more information, see [When to use the `injected` projection type](#).

Use dynamic ID partitioning

When your data is partitioned by a property with high cardinality or when the values cannot be known in advance, you can use the `injected` projection type. Examples of such properties are user names, and IDs of devices or products. When you use the `injected` projection type to configure a partition key, Athena uses values from the query itself to compute the set of partitions that will be read.

For Athena to be able to run a query on a table that has a partition key configured with the `injected` projection type, the following must be true:

- Your query must include at least one value for the partition key.
- The value(s) must be literals or expressions that can be evaluated without reading any data.

If any of these criteria are not met, your query fails with the following error:

CONSTRAINT_VIOLATION: For the injected projected partition column *`column_name`*, the WHERE clause must contain only static equality conditions, and at least one such condition must be present.

When to use the `injected` projection type

Imagine you have a data set that consists of events from IoT devices, partitioned on the devices' IDs. This data set has the following characteristics:

- The device IDs are generated randomly.
- New devices are provisioned frequently.
- There are currently hundreds of thousands of devices, and in the future there will be millions.

This data set is difficult to manage using traditional metastores. It is difficult to keep the partitions in sync between the data storage and the metastore, and filtering partitions can be slow during query planning. But if you configure a table to use partition projection and use the `injected` projection type, you have two advantages: you don't have to manage partitions in the metastore, and your queries don't have to look up partition metadata.

The following `CREATE TABLE` example creates a table for the device event data set just described. The table uses the injected projection type.

```
CREATE EXTERNAL TABLE device_events (
  event_time TIMESTAMP,
  data STRING,
  battery_level INT
)
```

```
PARTITIONED BY (
  device_id STRING
)
LOCATION "s3://amzn-s3-demo-bucket/prefix/"
TBLPROPERTIES (
  "projection.enabled" = "true",
  "projection.device_id.type" = "injected",
  "storage.location.template" = "s3://amzn-s3-demo-bucket/prefix/${device_id}"
)
```

The following example query looks up the number of events received from three specific devices over the course of 12 hours.

```
SELECT device_id, COUNT(*) AS events
FROM device_events
WHERE device_id IN (
  '4a770164-0392-4a41-8565-40ed8cec737e',
  'f71d12cf-f01f-4877-875d-128c23cbde17',
  '763421d8-b005-47c3-ba32-cc747ab32f9a'
)
AND event_time BETWEEN TIMESTAMP '2023-11-01 20:00' AND TIMESTAMP '2023-11-02 08:00'
GROUP BY device_id
```

When you run this query, Athena sees the three values for the `device_id` partition key and uses them to compute the partition locations. Athena uses the value for the `storage.location.template` property to generate the following locations:

- `s3://amzn-s3-demo-bucket/prefix/4a770164-0392-4a41-8565-40ed8cec737e`
- `s3://amzn-s3-demo-bucket/prefix/f71d12cf-f01f-4877-875d-128c23cbde17`
- `s3://amzn-s3-demo-bucket/prefix/763421d8-b005-47c3-ba32-cc747ab32f9a`

If you leave out the `storage.location.template` property from the partition projection configuration, Athena uses Hive-style partitioning to project partition locations based on the value in `LOCATION` (for example, `s3://amzn-s3-demo-bucket/prefix/device_id=4a770164-0392-4a41-8565-40ed8cec737e`).

Amazon Data Firehose example

When you use Firehose to deliver data to Amazon S3, the default configuration writes objects with keys that look like the following example:

```
s3://amzn-s3-demo-bucket/prefix/yyyy/MM/dd/HH/file.extension
```

To create an Athena table that finds the partitions automatically at query time, instead of having to add them to the AWS Glue Data Catalog as new data arrives, you can use partition projection.

The following `CREATE TABLE` example uses the default Firehose configuration.

```
CREATE EXTERNAL TABLE my_ingested_data (
 ...
)
...
PARTITIONED BY (
 datehour STRING
)
LOCATION "s3://amzn-s3-demo-bucket/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.datehour.type" = "date",
 "projection.datehour.format" = "yyyy/MM/dd/HH",
 "projection.datehour.range" = "2021/01/01/00,NOW",
 "projection.datehour.interval" = "1",
 "projection.datehour.interval.unit" = "HOURS",
 "storage.location.template" = "s3://amzn-s3-demo-bucket/prefix/${datehour}/"
)
```

The `TBLPROPERTIES` clause in the `CREATE TABLE` statement tells Athena the following:

- Use partition projection when querying the table
- The partition key `datehour` is of type `date` (which includes an optional time)
- How the dates are formatted
- The range of date times. Note that the values must be separated by a comma, not a hyphen.
- Where to find the data on Amazon S3.

When you query the table, Athena calculates the values for `datehour` and uses the storage location template to generate a list of partition locations.

Topics

- [How to use the date type](#)

- How to choose partition keys
- How to use custom prefixes and dynamic partitioning

How to use the date type

When you use the date type for a projected partition key, you must specify a range. Because you have no data for dates before the Firehose delivery stream was created, you can use the date of creation as the start. And because you do not have data for dates in the future, you can use the special token NOW as the end.

In the CREATE TABLE example, the start date is specified as January 1, 2021 at midnight UTC.

> **Note**
>
> Configure a range that matches your data as closely as possible so that Athena looks only for existing partitions.

When a query is run on the sample table, Athena uses the conditions on the datehour partition key in combination with the range to generate values. Consider the following query:

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2020/12/15/00'
AND datehour < '2021/02/03/15'
```

The first condition in the SELECT query uses a date that is before the start of the date range specified by the CREATE TABLE statement. Because the partition projection configuration specifies no partitions for dates before January 1, 2021, Athena looks for data only in the following locations, and ignores the earlier dates in the query.

```
s3://amzn-s3-demo-bucket/prefix/2021/01/01/00/
s3://amzn-s3-demo-bucket/prefix/2021/01/01/01/
s3://amzn-s3-demo-bucket/prefix/2021/01/01/02/
...
s3://amzn-s3-demo-bucket/prefix/2021/02/03/12/
s3://amzn-s3-demo-bucket/prefix/2021/02/03/13/
s3://amzn-s3-demo-bucket/prefix/2021/02/03/14/
```

Similarly, if the query ran at a date and time before February 3, 2021 at 15:00, the last partition would reflect the current date and time, not the date and time in the query condition.

If you want to query for the most recent data, you can take advantage of the fact that Athena does not generate future dates and specify only a beginning `datehour`, as in the following example.

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2021/11/09/00'
```

How to choose partition keys

You can specify how partition projection maps the partition locations to partition keys. In the `CREATE TABLE` example in the previous section, the date and hour were combined into one partition key called datehour, but other schemes are possible. For example, you could also configure a table with separate partition keys for the year, month, day, and hour.

However, splitting dates into year, month, and day means that the `date` partition projection type can't be used. An alternative is to separate the date from the hour to still leverage the `date` partition projection type, but make queries that specify hour ranges easier to read.

With that in mind, the following `CREATE TABLE` example separates the date from the hour. Because `date` is a reserved word in SQL, the example uses `day` as the name for the partition key that represents the date.

```
CREATE EXTERNAL TABLE my_ingested_data2 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 hour INT
)
LOCATION "s3://amzn-s3-demo-bucket/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy/MM/dd",
 "projection.day.range" = "2021/01/01,NOW",
 "projection.day.interval" = "1",
```

```
  "projection.day.interval.unit" = "DAYS",
  "projection.hour.type" = "integer",
  "projection.hour.range" = "0,23",
  "projection.hour.digits" = "2",
  "storage.location.template" = "s3://amzn-s3-demo-bucket/prefix/${day}/${hour}/"
)
```

In the example `CREATE TABLE` statement, the hour is a separate partition key, configured as an integer. The configuration for the hour partition key specifies the range 0 to 23, and that the hour should be formatted with two digits when Athena generates the partition locations.

A query for the `my_ingested_data2` table might look like this:

```
SELECT *
FROM my_ingested_data2
WHERE day = '2021/11/09'
AND hour > 3
```

Understand partition key and partition projection data types

Note that `datehour` key in the first `CREATE TABLE` example is configured as `date` in the partition projection configuration, but the type of the partition key is `string`. The same is true for `day` in the second example. The types in the partition projection configuration only tell Athena how to format the values when it generates the partition locations. The types that you specify do not change the type of the partition key — in queries, `datehour` and `day` are of type `string`.

When a query includes a condition like `day = '2021/11/09'`, Athena parses the string on the right side of the expression using the date format specified in the partition projection configuration. After Athena verifies that the date is within the configured range, it uses the date format again to insert the date as a string into the storage location template.

Similarly, for a query condition like `day > '2021/11/09'`, Athena parses the right side and generates a list of all matching dates within the configured range. It then uses the date format to insert each date into the storage location template to create the list of partition locations.

Writing the same condition as `day > '2021-11-09'` or `day > DATE '2021-11-09'` does not work. In the first case, the date format does not match (note the hyphens instead of the forward slashes), and in the second case, the data types do not match.

How to use custom prefixes and dynamic partitioning

Firehose can be configured with custom prefixes and dynamic partitioning. Using these features, you can configure the Amazon S3 keys and set up partitioning schemes that better support your use case. You can also use partition projection with these partitioning schemes and configure them accordingly.

For example, you could use the custom prefix feature to get Amazon S3 keys that have ISO formatted dates instead of the default `yyyy/MM/dd/HH` scheme.

You can also combine custom prefixes with dynamic partitioning to extract a property like `customer_id` from Firehose messages, as in the following example.

```
prefix/!{timestamp:yyyy}-!{timestamp:MM}-!{timestamp:dd}/!
{partitionKeyFromQuery:customer_id}/
```

With that Amazon S3 prefix, the Firehose delivery stream would write objects to keys such as `s3://amzn-s3-demo-bucket/prefix/2021-11-01/customer-1234/file.extension`. For a property like `customer_id`, where the values may not be known in advance, you can use the partition projection type `injected` and use a `CREATE TABLE` statement like the following:

```
CREATE EXTERNAL TABLE my_ingested_data3 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 customer_id STRING
)
LOCATION "s3://amzn-s3-demo-bucket/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy-MM-dd",
 "projection.day.range" = "2021-01-01,NOW",
 "projection.day.interval" = "1",
 "projection.day.interval.unit" = "DAYS",
 "projection.customer_id.type" = "injected",
 "storage.location.template" = "s3://amzn-s3-demo-bucket/prefix/${day}/${customer_id}/"
)
```

When you query a table that has a partition key of type `injected`, your query must include a value for that partition key. A query for the `my_ingested_data3` table might look like this:

```
SELECT *
FROM my_ingested_data3
WHERE day BETWEEN '2021-11-01' AND '2021-11-30'
AND customer_id = 'customer-1234'
```

Use the DATE type for the day partition key

Because the values for the `day` partition key are ISO formatted, you can also use the `DATE` type for the day partition key instead of `STRING`, as in the following example:

```
PARTITIONED BY (day DATE, customer_id STRING)
```

When you query, this strategy allows you to use date functions on the partition key without parsing or casting, as in the following example:

```
SELECT *
FROM my_ingested_data3
WHERE day > CURRENT_DATE - INTERVAL '7' DAY
AND customer_id = 'customer-1234'
```

> **Note**
>
> Specifying a partition key of the `DATE` type assumes that you have used the [custom prefix] feature to create Amazon S3 keys that have ISO formatted dates. If you are using the default Firehose format of `yyyy/MM/dd/HH`, you must specify the partition key as type `string` even though the corresponding table property is of type `date`, as in the following example:
>
> ```
> PARTITIONED BY (
>   `mydate` string)
> TBLPROPERTIES (
>   'projection.enabled'='true',
>    ...
>   'projection.mydate.type'='date',
>   'storage.location.template'='s3://amzn-s3-demo-bucket/prefix/${mydate}')
> ```

Prevent Amazon S3 throttling

Throttling is the process of limiting the rate at which you use a service, an application, or a system. In AWS, you can use throttling to prevent overuse of the Amazon S3 service and increase the availability and responsiveness of Amazon S3 for all users. However, because throttling limits the rate at which the data can be transferred to or from Amazon S3, it's important to consider preventing your interactions from being throttled.

As pointed out in the performance tuning chapter, optimizations can depend on your service level decisions, on how you structure your tables and data, and on how you write your queries.

Topics

- Reduce throttling at the service level
- Optimize your tables
- Optimize your queries

Reduce throttling at the service level

To avoid Amazon S3 throttling at the service level, you can monitor your usage and adjust your service quotas, or you use certain techniques like partitioning. The following are some of the conditions that can lead to throttling:

- **Exceeding your account's API request limits** – Amazon S3 has default API request limits that are based on account type and usage. If you exceed the maximum number of requests per second for a single prefix, your requests may be throttled to prevent overload of the Amazon S3 service.
- **Insufficient partitioning of data** – If you do not properly partition your data and transfer a large amount of data, Amazon S3 can throttle your requests. For more information about partitioning, see the Use partitioning section in this document.
- **Large number of small objects** – If possible, avoid having a large number of small files. Amazon S3 has a limit of 5500 GET requests per second per partitioned prefix, and your Athena queries share this same limit. If you scan millions of small objects in a single query, your query will likely be throttled by Amazon S3.

To avoid excess scanning, you can use AWS Glue ETL to periodically compact your files, or you partition the table and add partition key filters. For more information, see the following resources.

- How can I configure an AWS Glue ETL job to output larger files? (*AWS Knowledge Center*)

- Reading input files in larger groups (*AWS Glue Developer Guide*)

Optimize your tables

Structuring your data is important if you encounter throttling issues. Although Amazon S3 can handle large amounts of data, throttling sometimes occurs because of the way the data is structured.

The following sections offer some suggestions on how to structure your data in Amazon S3 to avoid throttling issues.

Use partitioning

You can use partitioning to reduce throttling by limiting the amount of data that has to be accessed at any given time. By partitioning data on specific columns, you can distribute requests evenly across multiple objects and reduce the number of requests for a single object. Reducing the amount of data that must be scanned improves query performance and lowers cost.

You can define partitions, which act as virtual columns, when you create a table. To create a table with partitions in a `CREATE TABLE` statement, you use the `PARTITIONED BY (column_name data_type)` clause to define the keys to partition your data.

To restrict the partitions scanned by a query, you can specify them as predicates in a `WHERE` clause of the query. Thus, columns that are frequently used as filters are good candidates for partitioning. A common practice is to partition the data based on time intervals, which can lead to a multi-level partitioning scheme.

Note that partitioning also has a cost. When you increase the number of partitions in your table, the time required to retrieve and process partition metadata also increases. Thus, over-partitioning can remove the benefits you gain by partitioning more judiciously. If your data is heavily skewed to one partition value, and most queries use that value, then you may incur the additional overhead.

For more information about partitioning in Athena, see What is partitioning?

Bucket your data

Another way to partition your data is to bucket the data within a single partition. With bucketing, you specify one or more columns that contain rows that you want to group together. Then, you put those rows into multiple buckets. This way, you query only the bucket that must be read, which reduces the number of rows of data that must be scanned.

When you select a column to use for bucketing, select the column that has high cardinality (that is, that has many distinct values), is uniformly distributed, and is frequently used to filter the data. An example of a good column to use for bucketing is a primary key, such as an ID column.

For more information about bucketing in Athena, see What is bucketing?

Use AWS Glue partition indexes

You can use AWS Glue partition indexes to organize data in a table based on the values of one or more partitions. AWS Glue partition indexes can reduce the number of data transfers, the amount of data processing, and the time for queries to process.

An AWS Glue partition index is a metadata file that contains information about the partitions in the table, including the partition keys and their values. The partition index is stored in an Amazon S3 bucket and is updated automatically by AWS Glue as new partitions are added to the table.

When an AWS Glue partition index is present, queries attempt to fetch a subset of the partitions instead of loading all the partitions in the table. Queries only run on the subset of data that is relevant to the query.

When you create a table in AWS Glue, you can create a partition index on any combination of partition keys defined on the table. After you have created one or more partition indexes on a table, you must add a property to the table that enables partition filtering. Then, you can query the table from Athena.

For information about creating partition indexes in AWS Glue, see Working with partition indexes in AWS Glue in the *AWS Glue Developer Guide*. For information about adding a table property to enable partition filtering, see Optimize queries with AWS Glue partition indexing and filtering.

Use data compression and file splitting

Data compression can speed up queries significantly if files are at their optimal size or if they can be split into logical groups. Generally, higher compression ratios require more CPU cycles to compress and decompress the data. For Athena, we recommend that you use either Apache Parquet or Apache ORC, which compress data by default. For information about data compression in Athena, see Use compression in Athena.

Splitting files increases parallelism by allowing Athena to distribute the task of reading a single file among multiple readers. If a single file is not splittable, only a single reader can read the file while other readers are idle. Apache Parquet and Apache ORC also support splittable files.

Use optimized columnar data stores

Athena query performance improves significantly if you convert your data into a columnar format. When you generate columnar files, one optimization technique to consider is ordering the data based on partition key.

Apache Parquet and Apache ORC are commonly used open source columnar data stores. For information on converting existing Amazon S3 data source to one of these formats, see Convert to columnar formats.

Use a larger Parquet block size or ORC stripe size

Parquet and ORC have data storage parameters that you can tune for optimization. In Parquet, you can optimize for block size. In ORC, you can optimize for stripe size. The larger the block or stripe, the more rows that you can store in each. By default, the Parquet block size is 128 MB, and the ORC stripe size is 64 MB.

If an ORC stripe is less than 8 MB (the default value of `hive.orc.max_buffer_size`), Athena reads the whole ORC stripe. This is the tradeoff Athena makes between column selectivity and input/output operations per second for smaller stripes.

If you have tables with a very large number of columns, a small block or stripe size can cause more data to be scanned than necessary. In these cases, a larger block size can be more efficient.

Use ORC for complex types

Currently, when you query columns stored in Parquet that have complex data types (for example, `array`, `map`, or `struct`), Athena reads an entire row of data instead of selectively reading only the specified columns. This is a known issue in Athena. As a workaround, consider using ORC.

Choose a compression algorithm

Another parameter that you can configure is the compression algorithm on data blocks. For information about the compression algorithms supported for Parquet and ORC in Athena, see Athena compression support.

For more information about optimization of columnar storage formats in Athena, see the section "Optimize columnar data store generation" in the AWS Big Data Blog post Top 10 Performance Tuning Tips for Amazon Athena.

Use Iceberg tables

Apache Iceberg is an open table format for very large analytic datasets that is designed for optimized usage on Amazon S3. You can use Iceberg tables to help reduce throttling in Amazon S3.

Iceberg tables offer you the following advantages:

- You can partition Iceberg tables on one or more columns. This optimizes data access and reduces the amount of data that must be scanned by queries.
- Because Iceberg object storage mode optimizes Iceberg tables to work with Amazon S3, it can process large volumes of data and heavy query workloads.
- Iceberg tables in object storage mode are scalable, fault tolerant, and durable, which can help reduce throttling.
- ACID transaction support means that multiple users can add and delete Amazon S3 objects in an atomic manner.

For more information about Apache Iceberg, see Apache Iceberg. For more information about using Apache Iceberg tables in Athena, see Using Iceberg tables.

Optimize your queries

Use the suggestions in this section for optimizing your SQL queries in Athena.

Use LIMIT with the ORDER BY clause

The `ORDER BY` clause returns data in a sorted order. This requires Athena to send all rows of data to a single worker node and then sort the rows. This type of query can run for a long time or even fail.

For greater efficiency in your queries, look at the top or bottom *N* values, and then also use a `LIMIT` clause. This significantly reduces the cost of the sort by pushing both sorting and limiting to individual worker nodes rather than to a single worker.

Optimize JOIN clauses

When you join two tables, Athena distributes the table on the right to worker nodes, and then streams the table on the left to perform the join.

For this reason, specify the larger table on the left side of the join and the smaller table on the right side of the join. This way, Athena uses less memory and runs the query with lower latency.

Also note the following points:

- When you use multiple `JOIN` commands, specify tables from largest to smallest.
- Avoid cross joins unless they are required by the query.

Optimize GROUP BY clauses

The `GROUP BY` operator distributes rows based on the `GROUP BY` columns to the worker nodes. These columns are referenced in memory and the values are compared as the rows are ingested. The values are aggregated together when the `GROUP BY` column matches. In consideration of the way this process works, it is advisable to order the columns from the highest cardinality to the lowest.

Use numbers instead of strings

Because numbers require less memory and are faster to process compared to strings, use numbers instead of strings when possible.

Limit the number of columns

To reduce the total amount of memory required to store your data, limit the number of columns specified in your `SELECT` statement.

Use regular expressions instead of LIKE

Queries that include clauses such as `LIKE '%string%'` on large strings can be very computationally intensive. When you filter for multiple values on a string column, use the [regexp_like()](#) function and a regular expression instead. This is particularly useful when you compare a long list of values.

Use the LIMIT clause

Instead of selecting all columns when you run a query, use the `LIMIT` clause to return only the columns that you require. This reduces the size of the dataset that is processed through the query execution pipeline. `LIMIT` clauses are more helpful when you query tables that have a large of number of columns that are string-based. `LIMIT` clauses are also helpful when you perform multiple joins or aggregations on any query.

Additional resources

For additional information about performance tuning in Athena, consider the following resources:

- AWS Big Data blog post: Top 10 performance tuning tips for Amazon Athena.
- AWS Big Data blog post: Run queries 3x faster with up to 70% cost savings on the latest Amazon Athena engine in the *AWS Big Data Blog*.
- AWS Big Data blog post: Improve federated queries with predicate pushdown in Amazon Athena.
- Amazon Simple Storage Service User Guide: Best practices design patterns: optimizing Amazon S3 performance.
- Other Athena posts in the AWS big data blog.
- Ask a question on AWS re:Post using the **Amazon Athena** tag.
- Consult the Athena topics in the AWS knowledge center.
- Contact AWS Support (in the AWS Management Console, click **Support**, **Support Center**)

Use compression in Athena

Athena supports a variety of compression formats for reading and writing data, including reading from a table that uses multiple compression formats. For example, Athena can successfully read the data in a table that uses Parquet file format when some Parquet files are compressed with Snappy and other Parquet files are compressed with GZIP. The same principle applies for ORC, text file, and JSON storage formats.

Supported compression formats

Athena supports the following compression formats:

- **BZIP2** – Format that uses the Burrows-Wheeler algorithm.
- **DEFLATE** – Compression algorithm based on LZSS and Huffman coding. Deflate is relevant only for the Avro file format.
- **GZIP** – Compression algorithm based on Deflate. For Hive tables in Athena engine versions 2 and 3, and Iceberg tables in Athena engine version 2, GZIP is the default write compression format for files in the Parquet and text file storage formats. Files in the `tar.gz` format are not supported.
- **LZ4** – This member of the Lempel-Ziv 77 (LZ7) family also focuses on compression and decompression speed rather than maximum compression of data. LZ4 has the following framing formats:
  - **LZ4 Raw/Unframed** – An unframed, standard implementation of the LZ4 block compression format. For more information, see the LZ4 block format description on GitHub.

  - **LZ4 framed** – The usual framing implementation of LZ4. For more information, see the LZ4 frame format description on GitHub.
  - **LZ4 hadoop-compatible** – The Apache Hadoop implementation of LZ4. This implementation wraps LZ4 compression with the BlockCompressorStream.java class.
- **LZO** – Format that uses the Lempel–Ziv–Oberhumer algorithm, which focuses on high compression and decompression speed rather than the maximum compression of data. LZO has two implementations:
  - **Standard LZO** – For more information, see the LZO abstract on the Oberhumer website.
  - **LZO hadoop-compatible** – This implementation wraps the LZO algorithm with the BlockCompressorStream.java class.
- **SNAPPY** – Compression algorithm that is part of the Lempel-Ziv 77 (LZ7) family. Snappy focuses on high compression and decompression speed rather than the maximum compression of data.
- **ZLIB** – Based on Deflate, ZLIB is the default write compression format for files in the ORC data storage format. For more information, see the zlib page on GitHub.
- **ZSTD** – The Zstandard real-time data compression algorithm is a fast compression algorithm that provides high compression ratios. The Zstandard (ZSTD) library is provided as open source software using a BSD license. ZSTD is the default compression for Iceberg tables. When writing ZSTD compressed data, Athena uses ZSTD compression level 3 by default. For more information about using ZSTD compression levels in Athena, see Use ZSTD compression levels.

Note

Athena does not support writing Parquet files compressed with LZ4 or LZO formats. Reads for these compression formats are supported.

Specify compression formats

When you write CREATE TABLE or CTAS statements, you can specify compression properties that specify the compression type to use when Athena writes to those tables.

- For CTAS, see CTAS table properties. For examples, see Examples of CTAS queries.
- For CREATE TABLE, see ALTER TABLE SET TBLPROPERTIES for a list of compression table properties.

Specify no compression

CREATE TABLE statements support writing uncompressed files. To write uncompressed files, use the following syntax:

- CREATE TABLE (text file or JSON) – In `TBLPROPERTIES`, specify `write.compression = NONE`.
- CREATE TABLE (Parquet) – In `TBLPROPERTIES`, specify `parquet.compression = UNCOMPRESSED`.
- CREATE TABLE (ORC) – In `TBLPROPERTIES`, specify `orc.compress = NONE`.

Notes and resources

- Currently, uppercase file extensions such as `.GZ` or `.BZIP2` are not recognized by Athena. Avoid using datasets with uppercase file extensions, or rename the data file extensions to lowercase.
- For data in CSV, TSV, and JSON, Athena determines the compression type from the file extension. If no file extension is present, Athena treats the data as uncompressed plain text. If your data is compressed, make sure the file name includes the compression extension, such as `gz`.
- The ZIP file format is not supported.
- For querying Amazon Data Firehose logs from Athena, supported formats include GZIP compression or ORC files with SNAPPY compression.
- For more information about using compression, see section 3 ("Compress and split files") of the AWS Big Data Blog post [Top 10 performance tuning tips for Amazon Athena](#).

Topics

Use Hive table compression

The compression options for Hive tables in Athena vary by engine version and file format.

Hive compression support in Athena engine version 3

The following table summarizes the compression format support in Athena engine version 3 for storage file formats in Apache Hive. Text file format includes TSV, CSV, JSON, and custom SerDes for text. "Yes" or "No" in a cell apply equally to read and write operations except where noted. For the purposes of this table, CREATE TABLE, CTAS, and INSERT INTO are considered write operations. For more information about using ZSTD compression levels in Athena, see Use ZSTD compression levels.

	Avro	Ion	ORC	Parquet	Text file
BZIP2	Yes	Yes	No	No	Yes
DEFLATE	Yes	No	No	No	No
GZIP	No	Yes	No	Yes	Yes
LZ4	No	Yes	Yes	Write - No Read - Yes	Yes
LZO	No	Write - No Read - Yes	No	Write - No Read - Yes	Write - No Read - Yes
SNAPPY	Yes	Yes	Yes	Yes	Yes
ZLIB	No	No	Yes	No	No
ZSTD	Yes	Yes	Yes	Yes	Yes
NONE	Yes	Yes	Yes	Yes	Yes

Use Iceberg table compression

The compression options for Iceberg tables in Athena vary by engine version and file format.

Iceberg compression support in Athena engine version 3

The following table summarizes the compression format support in Athena engine version 3 for storage file formats in Apache Iceberg. "Yes" or "No" in a cell apply equally to read and write operations except where noted. For the purposes of this table, CREATE TABLE, CTAS, and INSERT

INTO are considered write operations. The default storage format for Iceberg in Athena engine version 3 is Parquet. The default compression format for Iceberg in Athena engine version 3 is ZSTD. For more information about using ZSTD compression levels in Athena, see Use ZSTD compression levels.

	Avro	ORC	Parquet (default)
BZIP2	No	No	No
GZIP	Yes	No	Yes
LZ4	No	Yes	No
SNAPPY	Yes	Yes	Yes
ZLIB	No	Yes	No
ZSTD	Yes	Yes	Yes (default)
NONE	Yes (specify `None` or `Deflate`)	Yes	Yes (specify `None` or `Uncompressed`)

Use ZSTD compression levels

The Zstandard real-time data compression algorithm is a fast compression algorithm that provides high compression ratios. The Zstandard (ZSTD) library is open source software and uses a BSD license. Athena supports reading and writing ZSTD compressed ORC, Parquet, and text file data.

You can use ZSTD compression levels to adjust the compression ratio and speed according to your requirements. The ZSTD library supports compression levels from 1 to 22. Athena uses ZSTD compression level 3 by default.

Compression levels provide granular trade-offs between compression speed and the amount of compression achieved. Lower compression levels provide faster speed but larger file sizes. For example, you can use level 1 if speed is most important and level 22 if size is most important. Level 3 is suitable for many use cases and is the default. Use levels greater than 19 with caution as they require more memory. The ZSTD library also offers negative compression levels that extend the range of compression speed and ratios. For more information, see the Zstandard Compression RFC.

The abundance of compression levels offers substantial opportunities for fine tuning. However, make sure that you measure your data and consider the tradeoffs when deciding on a compression level. We recommend using the default level of 3 or a level in the range from 6 to 9 for a reasonable tradeoff between compression speed and compressed data size. Reserve levels 20 and greater for cases where size is most important and compression speed is not a concern.

Considerations and limitations

When using ZSTD compression level in Athena, consider the following points.

- The ZSTD `compression_level` property is supported only in Athena engine version 3.
- The ZSTD `compression_level` property is supported for the `ALTER TABLE`, `CREATE TABLE`, `CREATE TABLE AS` (CTAS), and `UNLOAD` statements.
- The `compression_level` property is optional.
- The `compression_level` property is supported only for ZSTD compression.
- Possible compression levels are 1 through 22.
- The default compression level is 3.

For information about Apache Hive ZSTD compression support in Athena, see [Use Hive table compression](). For information about Apache Iceberg ZSTD compression support in Athena, see [Use Iceberg table compression]().

Specify ZSTD compression levels

To specify the ZSTD compression level for the `ALTER TABLE`, `CREATE TABLE`, `CREATE TABLE AS`, and `UNLOAD` statements, use the `compression_level` property. To specify ZSTD compression itself, you must use the individual compression property that the syntax for the statement uses.

ALTER TABLE SET TBLPROPERTIES

In the [ALTER TABLE SET TBLPROPERTIES]() statement `SET TBLPROPERTIES` clause, specify ZSTD compression using `'write.compression' = ' ZSTD'` or `'parquet.compression' = 'ZSTD'`. Then use the `compression_level` property to specify a value from 1 to 22 (for example, `'compression_level' = '5'`). If you do not specify a compression level property, the compression level defaults to 3.

Example

The following example modifies the table `existing_table` to use Parquet file format with ZSTD compression and ZSTD compression level 4. Note that in the `TBLPROPERTIES` clause the compression level value must be entered as a string rather an integer and therefore must be enclosed in either single or double quotes.

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = '4')
```

CREATE TABLE

In the [CREATE TABLE](#) statement `TBLPROPERTIES` clause, specify '`write.compression' = 'ZSTD'` or `'parquet.compression' = 'ZSTD'`, and then use `compression_level = compression_level` and specify a value from 1 to 22 as a string. If the `compression_level` property is not specified, the default compression level is 3.

Example

The following example creates a table in Parquet file format using ZSTD compression and ZSTD compression level 4.

```
CREATE EXTERNAL TABLE new_table (
  `col0` string COMMENT '',
  `col1` string COMMENT ''
)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/'
TBLPROPERTIES ('write.compression' = 'ZSTD', 'compression_level' = '4')
```

CREATE TABLE AS (CTAS)

In the [CREATE TABLE AS](#) statement `WITH` clause, specify `write_compression = 'ZSTD'`, or `parquet_compression = 'ZSTD'`, and then use `compression_level = compression_level` and specify a value from 1 to 22 as an integer. If the `compression_level` property is not specified, the default compression level is 3.

Example

The following CTAS example specifies Parquet as the file format using ZSTD compression with compression level 4. Note that, in the `WITH` clause, the value for compression level must be specified as an integer, not as a string.

```
CREATE TABLE new_table
WITH ( format = 'PARQUET', write_compression = 'ZSTD', compression_level = 4)
AS SELECT * FROM old_table
```

UNLOAD

In the UNLOAD statement `WITH` clause, specify `compression = 'ZSTD'`, and then use `compression_level = compression_level` and specify a value from 1 to 22 as an integer. If the `compression_level` property is not specified, the default compression level is 3.

Example

The following example unloads the query results to the specified location using the Parquet file format, ZSTD compression, and ZSTD compression level 4.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://amzn-s3-demo-bucket/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

Tag Athena resources

A tag consists of a key and a value, both of which you define. When you tag an Athena resource, you assign custom metadata to it. You can use tags to categorize your AWS resources in different ways; for example, by purpose, owner, or environment. In Athena, resources like workgroups, data catalogs, and capacity reservations are taggable resources. For example, you can create a set of tags for workgroups in your account that helps you track workgroup owners, or identify workgroups by their purpose. If you also enable the tags as cost allocation tags in the Billing and Cost Management console, costs associated with running queries appear in your Cost and Usage Report with that cost allocation tag. We recommend that you that you use AWS tagging best practices to create a consistent set of tags to meet your organization requirements.

You can work with tags using the Athena console or the API operations.

Topics

- Tag basics
- Tag restrictions
- Work with tags for workgroups
- Use API and AWS CLI tag operations
- Use tag-based IAM access control policies

Tag basics

A tag is a label that you assign to an Athena resource. Each tag consists of a key and an optional value, both of which you define.

Tags enable you to categorize your AWS resources in different ways. For example, you can define a set of tags for your account's workgroups that helps you track each workgroup owner or purpose.

You can add tags when creating a new Athena workgroup or data catalog, or you can add, edit, or remove tags from them. You can edit a tag in the console. To use API operations to edit a tag, remove the old tag and add a new one. If you delete a resource, any tags for the resource are also deleted.

Athena does not automatically assign tags to your resources. You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. Do not add duplicate tag keys to the same resource. If you do, Athena issues an error message. If you use the **TagResource** action to tag a resource using an existing tag key, the new tag value overwrites the old value.

In IAM, you can control which users in your Amazon Web Services account have permission to create, edit, remove, or list tags. For more information, see Use tag-based IAM access control policies.

For a complete list of Amazon Athena tag actions, see the API action names in the Amazon Athena API Reference.

You can use tags for billing. For more information, see Using tags for billing in the *AWS Billing and Cost Management User Guide*.

For more information, see Tag restrictions.

Tag restrictions

Tags have the following restrictions:

- In Athena, you can tag workgroups and data catalogs. You cannot tag queries.
- The maximum number of tags per resource is 50. To stay within the limit, review and delete unused tags.
- For each resource, each tag key must be unique, and each tag key can have only one value. Do not add duplicate tag keys at the same time to the same resource. If you do, Athena issues an error message. If you tag a resource using an existing tag key in a separate `TagResource` action, the new tag value overwrites the old value.
- Tag key length is 1-128 Unicode characters in UTF-8.
- Tag value length is 0-256 Unicode characters in UTF-8.

  Tagging operations, such as adding, editing, removing, or listing tags, require that you specify an ARN for the workgroup resource.
- Athena allows you to use letters, numbers, spaces represented in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case-sensitive.
- The `"aws:"` prefix in tag keys is reserved for AWS use. You can't edit or delete tag keys with this prefix. Tags with this prefix do not count against your per-resource tags limit.
- The tags you assign are available only to your Amazon Web Services account.

Work with tags for workgroups

Using the Athena console, you can see which tags are in use by each workgroup in your account. You can view tags by workgroup only. You can also use the Athena console to apply, edit, or remove tags from one workgroup at a time.

You can search workgroups using the tags you created.

Topics

- [Display tags for individual workgroups]
- [Add and delete tags on an individual workgroup]

Display tags for individual workgroups

To display tags for an individual workgroup in the Athena console

1. Open the Athena console at https://console.aws.amazon.com/athena/.

2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. On the navigation menu, choose **Workgroups**, and then choose the workgroup that you want.
4. Do one of the following:
   - Choose the **Tags** tab. If the list of tags is long, use the search box.
   - Choose **Edit**, and then scroll down to the **Tags** section.

Add and delete tags on an individual workgroup

You can manage tags for an individual workgroup directly from the **Workgroups** tab.

> **Note**
>
> If you want users to add tags when they create a workgroup in the console or pass in tags when they use the **CreateWorkGroup** action, make sure that you give the users IAM permissions to the **TagResource** and **CreateWorkGroup** actions.

To add a tag when you create a new workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. On the navigation menu, choose **Workgroups**.
3. Choose **Create workgroup** and fill in the values as needed. For detailed steps, see [Create a workgroup](#).

4. In the **Tags** section, add one or more tags by specifying keys and values. Do not add duplicate tag keys at the same time to the same workgroup. If you do, Athena issues an error message. For more information, see [Tag restrictions](#).
5. When you are done, choose **Create workgroup**.

To add or edit a tag to an existing workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Workgroups**.
3. Choose the workgroup that you want to modify.
4. Do one of the following:
   - Choose the **Tags** tab, and then choose **Manage tags**.
   - Choose **Edit**, and then scroll down to the **Tags** section.
5. Specify a key and value for each tag. For more information, see [Tag restrictions](#).
6. Choose **Save**.

To delete a tag from an individual workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the navigation pane, choose **Workgroups**.
3. Choose the workgroup that you want to modify.
4. Do one of the following:
   - Choose the **Tags** tab, and then choose **Manage tags**.
   - Choose **Edit**, and then scroll down to the **Tags** section.
5. In the list of tags, choose **Remove** for the tag that you want to delete, and then choose **Save**.

Use API and AWS CLI tag operations

Use the following tag operations to add, remove, or list tags on a resource.

API	CLI	Action description
TagResource	tag-resource	Add or overwrite one or more tags on the resource that has the specified ARN.
UntagResource	untag-resource	Delete one or more tags from the resource that has the specified ARN.
ListTagsForResource	list-tags-for-resource	List one or more tags for the resource that has the specified ARN.

Add tags when you create a resource

To add tags when you create a workgroup or data catalog, use the `tags` parameter with the `CreateWorkGroup` or `CreateDataCatalog` API operations or with the AWS CLI `create-work-group` or `create-data-catalog` commands.

Manage tags using API actions

The following examples show how to use tag API actions to manage tags on workgroups and data catalogs. The examples are in the Java programming language.

Example – TagResource

The following example adds two tags to the workgroup `workgroupA`:

```
List<Tag> tags = new ArrayList<>();
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTags(tags);

client.tagResource(request);
```

The following example adds two tags to the data catalog `datacatalogA`:

```
List<Tag> tags = new ArrayList<>();
```

```
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTags(tags);

client.tagResource(request);
```

> **Note**
>
> Do not add duplicate tag keys to the same resource. If you do, Athena issues an error message. If you tag a resource using an existing tag key in a separate `TagResource` action, the new tag value overwrites the old value.

Example – UntagResource

The following example removes `tagKey2` from the workgroup `workgroupA`:

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");

UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

The following example removes `tagKey2` from the data catalog `datacatalogA`:

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");

UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

Example – ListTagsForResource

The following example lists tags for the workgroup `workgroupA`:

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

The following example lists tags for the data catalog `datacatalogA`:

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

Manage tags using the AWS CLI

The following examples show how to use the AWS CLI to create and manage tags on data catalogs.

Add tags to a resource: tag-resource

The `tag-resource` command adds one or more tags to a specified resource.

Syntax

```
aws athena tag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tags
Key=string,Value=string Key=string,Value=string
```

The `--resource-arn` parameter specifies the resource to which the tags are added. The `--tags` parameter specifies a list of space-separated key-value pairs to add as tags to the resource.

Example

The following example adds tags to the `mydatacatalog` data catalog.

```
aws athena tag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tags Key=Color,Value=Orange
 Key=Time,Value=Now
```

To show the result, use the `list-tags-for-resource` command.

For information about adding tags when using the `create-data-catalog` command, see Registering a catalog: Create-data-catalog.

List the tags for a resource: list-tags-for-resource

The `list-tags-for-resource` command lists the tags for the specified resource.

Syntax

`aws athena list-tags-for-resource --resource-arn arn:aws:athena:`*`region`*`:`*`account_id`*`:datacatalog/`*`catalog_name`*

The `--resource-arn` parameter specifies the resource for which the tags are listed.

The following example lists the tags for the `mydatacatalog` data catalog.

```
aws athena list-tags-for-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog
```

The following sample result is in JSON format.

```
{
    "Tags": [
        {
            "Key": "Time",
            "Value": "Now"
        },
        {
            "Key": "Color",
            "Value": "Orange"
        }
    ]
}
```

Remove tags from a resource: untag-resource

The `untag-resource` command removes the specified tag keys and their associated values from the specified resource.

Syntax

```
aws athena untag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tag-keys
key_name [key_name ...]
```

The `--resource-arn` parameter specifies the resource from which the tags are removed. The `--tag-keys` parameter takes a space-separated list of key names. For each key name specified, the `untag-resource` command removes both the key and its value.

The following example removes the `Color` and `Time` keys and their values from the `mydatacatalog` catalog resource.

```
aws athena untag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tag-keys Color Time
```

Use tag-based IAM access control policies

Having tags allows you to write an IAM policy that includes the `Condition` block to control access to a resource based on its tags. This section includes tag policy examples for workgroup and data catalog resources.

Tag policy examples for workgroups

Example – Basic tagging policy

The following IAM policy allows you to run queries and interact with tags for the workgroup named `workgroupA`:

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
```

```
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
        }
    ]
}
```

Example – Policy block that denies actions on a workgroup based on a tag key and tag value pair

Tags that are associated with a resource like a workgroup are referred to as resource tags. Resource tags let you write policy blocks like the following that deny the listed actions on any workgroup tagged with a key-value pair like `stack`, `production`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "athena:GetWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
                }
            }
```

```
        }
    ]
}
```

Example – Policy block that restricts tag-changing action requests to specified tags

Tags that are passed in as parameters to operations that change tags (for example, `TagResource`, `UntagResource`, or `CreateWorkGroup` with tags) are referred to as request tags. The following example policy block allows the `CreateWorkGroup` operation only if one of the tags passed has the key `costcenter` and the value 1, 2, or 3.

> **Note**
>
> If you want to allow an IAM role to pass in tags as part of a `CreateWorkGroup` operation, make sure that you give the role permissions to the `TagResource` and `CreateWorkGroup` actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:TagResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
                }
            }
        }
    ]
}
```

Tag policy examples for data catalogs

Example – Basic tagging policy

The following IAM policy allows you to interact with tags for the data catalog named `datacatalogA`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
```

```
                "arn:aws:athena:us-east-1:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-
east-1:123456789012:datacatalog/datacatalogA"
        }
    ]
}
```

Example – Policy block that denies actions on a Data Catalog based on a tag key and tag value pair

You can use resource tags to write policy blocks that deny specific actions on data catalogs that are tagged with specific tag key-value pairs. The following example policy denies actions on data catalogs that have the tag key-value pair `stack`, `production`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
```

```
                "athena:GetTableMetadata",
                "athena:ListTableMetadata",
                "athena:StartQueryExecution",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
                }
            }
        }
    ]
}
```

Example – Policy block that restricts tag-changing action requests to specified tags

Tags that are passed in as parameters to operations that change tags (for example, `TagResource`, `UntagResource`, or `CreateDataCatalog` with tags) are referred to as request tags. The following example policy block allows the `CreateDataCatalog` operation only if one of the tags passed has the key `costcenter` and the value 1, 2, or 3.

> **Note**
>
> If you want to allow an IAM role to pass in tags as part of a `CreateDataCatalog` operation, make sure that you give the role permissions to the `TagResource` and `CreateDataCatalog` actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:TagResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
                }
            }
        }
    ]
}
```

Service Quotas

> **Note**
>
> The Service Quotas console provides information about Amazon Athena quotas. You can also use the Service Quotas console to request quota increases for the quotas that are adjustable. For AWS Glue related schema limitations, see the AWS Glue endpoints and quotas page. For general information about AWS service quotas, see AWS service quotas in the *AWS General Reference*.

Queries

Your account has the following query-related quotas for Amazon Athena. For details, see the Amazon Athena endpoints and quotas page of the AWS General Reference.

- **Active DDL queries** – The number of active DDL queries. DDL queries include `CREATE TABLE` and `ALTER TABLE ADD PARTITION` queries.
- **DDL query timeout** – The maximum amount of time in minutes a DDL query can run before it gets cancelled.
- **Active DML queries** – The number of active DML queries. DML queries include `SELECT`, `CREATE TABLE AS` (CTAS), and `INSERT INTO` queries. The specific quotas vary by AWS Region.
- **DML query timeout** – The maximum amount of time in minutes a DML query can run before it gets cancelled. You can request an increase in this timeout up to a maximum of 240 minutes.

To request quota increases, you can use the Athena Service Quotas console.

Athena processes queries by assigning resources based on the overall service load and the number of incoming requests. Your queries may be temporarily queued before they run. Asynchronous processes pick up the queries from queues and run them on physical resources as soon as the resources become available and for as long as your account configuration permits.

The Active DML queries and Active DDL queries quotas include both running and queued queries. For example, if your Active DML query quota is 25 and your total of running and queued queries is 26, query 26 will result in a TooManyRequestsException error.

> **Note**
>
> If you would like to control concurrency directly for the queries you run in Athena, you can use capacity reservations. For more information, see Manage query processing capacity.

Query string length

The maximum allowed query string length is 262144 bytes, where the strings are encoded in UTF-8. This is not an adjustable quota. However, you can work around this limitation by splitting long queries into multiple smaller queries. For more information, see How can I increase the maximum query string length in Athena? in the AWS Knowledge Center.

Workgroups

When you work with Athena workgroups, remember the following points:

- Athena service quotas are shared across all workgroups in an account.
- The maximum number of workgroups you can create per Region in an account is 1000.
- The maximum number of prepared statements in a workgroup is 1000.
- The maximum number of tags per workgroup is 50. For more information, see Tag restrictions.

Databases, tables, and partitions

Athena uses the AWS Glue Data Catalog. For service quotas on tables, databases, and partitions (for example, the maximum number of databases or tables per account), see AWS Glue endpoints and quotas. Note that, although Athena supports querying AWS Glue tables that have 10 million partitions, Athena cannot read more than 1 million partitions in a single scan.

Amazon S3 buckets

When you work with Amazon S3 buckets, remember the following points:

- Amazon S3 has a default service quota of 10,000 buckets per account.
- Athena requires a separate bucket to log results.
- You can request a quota increase of up to one million Amazon S3 buckets per AWS account.

Per account API call quotas

Athena APIs have default quotas for the number of calls to the API per account (not per query). For a complete list of the default quotas, see Service quotas table in the AWS General Reference guide.

If you use any of these APIs and exceed the default quota for the number of calls per second, or the burst capacity in your account, the Athena API issues an error similar to the following: ""ClientError: An error occurred (ThrottlingException) when calling the *<API_name>* operation: Rate exceeded." Reduce the number of calls per second, or the burst capacity for the API for this account.

You can change the Athena quota for per account API calls in the Athena Service Quotas console.

Athena engine versioning

Athena occasionally releases a new engine version to provide improved performance, functionality, and code fixes. When a new engine version is available, Athena notifies you through the Athena console and your AWS Health Dashboard. Your AWS Health Dashboard notifies you about events that can affect your AWS services or account. For more information about AWS Health Dashboard, see Getting started with the AWS Health Dashboard.

Engine versioning is configured per workgroup. You can use workgroups to control which query engine your queries use and whether to let Athena automatically upgrade your workgroups. The query engine that is in use is shown in the query editor, on the workgroup details page, and is available through the Athena APIs.

- By default, workgroups are configured to auto upgrade. When a workgroup is set to auto upgrade, Athena upgrades the workgroup for you unless it finds incompatibilities.
- If you configure a workgroup to use a given version, Athena will not change the version of the workgroup.

In both cases, Athena upgrades your workgroups when a version is no longer available. Athena notifies you through AWS Health Dashboard regarding when an engine version will no longer be offered. Your AWS Health Dashboard notifies you about events that can affect your AWS services or account. For more information about AWS Health Dashboard, see Getting started with the AWS Health Dashboard.

When you start using a new engine version, a small subset of queries may break due to incompatibilities. Breaking changes are announced when a new Athena version is released. You should use workgroups to test your queries in advance of the upgrade by creating a test workgroup that uses the new engine or by test upgrading an existing workgroup. For more information, see Test queries in advance of an engine version upgrade.

Topics

Change Athena engine versions

Athena occasionally releases a new engine version to provide improved performance, functionality, and code fixes. When a new engine version is available, Athena notifies you in the console. You can choose to let Athena decide when to upgrade, or manually specify an Athena engine version per workgroup.

Find the engine version for a workgroup

You can use the **Workgroups** page to find the current engine version for any workgroup.

To find the current engine version for any workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the Athena console navigation pane, choose **Workgroups**.
4. On the **Workgroups** page, find the workgroup that you want. The **Query engine version** column for the workgroup displays the query engine version.

Use the Athena console to change the engine version

When a new engine version is available, you can choose to let Athena decide when to upgrade the workgroup, or manually specify the Athena engine version that the workgroup uses. If only one version is currently available, manually specifying a different version is not possible.

> **Note**
>
> To change the engine version for a workgroup, you must have permission to perform the `athena:ListEngineVersions` action on the workgroup. For IAM policy examples, see [Example workgroup policies](#).

To let Athena decide when to upgrade the workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. In the console navigation pane, choose **Workgroups**.
4. In the list of workgroups, choose the link for the workgroup that you want to configure.
5. Choose **Edit**.

6. In the **Query engine version** section, for **Update query engine**, choose **Automatic** to let Athena choose when to upgrade your workgroup. This is the default setting.
7. Choose **Save changes**.

   In the list of workgroups, the **Query engine update status** for the workgroup shows **Automatic**.

To manually choose an engine version

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. In the console navigation pane, choose **Workgroups**.
4. In the list of workgroups, choose the link for the workgroup that you want to configure.
5. Choose **Edit**.
6. In the **Query engine version** section, for **Update query engine**, choose **Manual** to manually choose an engine version.
7. Use the **Query engine version** option to choose the engine version that you want the workgroup to use. If a different engine version is unavailable, a different engine version cannot be specified.
8. Choose **Save changes**.

   In the list of workgroups, the **Query engine update status** for the workgroup shows **Manual**.

Use the AWS CLI to change the engine version

To change the engine version using the AWS CLI, use the syntax in the following example.

```
aws athena update-work-group --work-group workgroup-name --configuration-updates
 EngineVersion={SelectedEngineVersion='Athena engine version 3'}
```

Specify the engine version when you create a workgroup

When you create a workgroup, you can specify the engine version that the workgroup uses or let Athena decide when to upgrade the workgroup. If a new engine version is available, a best practice is to create a workgroup to test the new engine before you upgrade your other workgroups. To specify the engine version for a workgroup, you must have the `athena:ListEngineVersions` permission on the workgroup. For IAM policy examples, see Example workgroup policies.

To specify the engine version when you create a workgroup

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. In the console navigation pane, choose **Workgroups**.
4. On the **Workgroups** page, choose **Create workgroup**.
5. On the **Create workgroup** page, in the **Query engine version** section, do one of the following:
   - Choose **Automatic** to let Athena choose when to upgrade your workgroup. This is the default setting.
   - Choose **Manual** to manually choose a different engine version if one is available.
6. Enter information for the other fields as necessary. For information about the other fields, see Create a workgroup.
7. Choose **Create workgroup**.

Test queries in advance of an engine version upgrade

When a workgroup is upgraded to a new engine version, some of your queries can break due to incompatibilities. To make sure that your engine version upgrade goes smoothly, you can test your queries in advance.

To test your queries prior to an engine version upgrade

1. Verify the engine version of the workgroup that you are using. The engine version that you are using is displayed on the **Workgroups** page in the **Query engine version** column for for the workgroup. For more information, see Find the engine version for a workgroup.
2. Create a test workgroup that uses the new engine version. For more information, see Specify the engine version when you create a workgroup.
3. Use the new workgroup to run the queries that you want to test.
4. If a query fails, check for breaking changes in the new engine that might be affecting the query. Some changes may require you to update the syntax of your queries.
5. If your queries still fail, contact AWS Support for assistance. In the AWS Management Console, choose **Support**, **Support Center**, or ask a question on AWS re:Post using the **Amazon Athena** tag.

Troubleshoot queries that fail after an engine version upgrade

If a query fails after an engine version upgrade, check for breaking changes, including changes that may affect the syntax in your queries.

If your queries still fail, contact AWS Support for assistance. In the AWS Management Console, choose **Support**, **Support Center**, or ask a question on AWS re:Post using the **Amazon Athena** tag.

Athena engine version 3

For engine version 3, Athena has introduced a continuous integration approach to open source software management that improves concurrency with the Trino and Presto projects so that you get faster access to community improvements, integrated and tuned within the Athena engine.

This release of Athena engine version 3 supports all the features of previous engine versions. This document highlights key differences between previous engine versions and Athena engine version 3. For more information, see the the *AWS Big Data Blog* article Upgrade to Athena engine version 3 to increase query performance and access more analytics features.

- Get started
- Improvements and new features
  - Added Features
  - Added Functions
  - Performance improvements
  - Reliability enhancements
  - Query syntax enhancements
  - Data format and data type enhancements
- Breaking changes
  - Query syntax changes
  - Data processing changes
  - Timestamp changes
- Limitations

Get started

To get started, either create a new Athena workgroup that uses Athena engine version 3 or configure an existing workgroup to use version 3.

For more information, see Changing Athena engine versions.

Improvements and new features

The features and updates listed include improvements from Athena itself and from functionality incorporated from open source Trino. For an exhaustive list of SQL query operators and functions, refer to the Trino documentation.

Added Features

Apache Spark bucketing algorithm support

Athena can read buckets generated by the Spark hash algorithm. To specify that data was originally written by the Spark hash algorithm, put `('bucketing_format'='spark')` in the `TBLPROPERTIES` clause of your `CREATE TABLE` statement. If this property is not specified, the Hive hash algorithm is used.

```
CREATE EXTERNAL TABLE `spark_bucket_table`(
  `id` int,
  `name` string
  )
CLUSTERED BY (`name`)
INTO 8 BUCKETS
STORED AS PARQUET
LOCATION
  's3://amzn-s3-demo-bucket/to/bucketed/table/'
TBLPROPERTIES ('bucketing_format'='spark')
```

Added Functions

The functions in this section are new to Athena engine version 3.

Aggregate functions

listagg(x, separator) – Returns the concatenated input values, separated by the separator string.

```
SELECT listagg(value, ',') WITHIN GROUP (ORDER BY value) csv_value
```

```
FROM (VALUES 'a', 'c', 'b') t(value);
```

Array functions

contains_sequence(x, seq) – Returns true if array x contains all array seq as a sequential subset (all values in the same consecutive order).

```
SELECT contains_sequence(ARRAY [1,2,3,4,5,6], ARRAY[1,2]);
```

Binary functions

murmur3(binary) – Computes the 128-bit MurmurHash3 hash of binary.

```
SELECT murmur3(from_base64('aaaaaa'));
```

Conversion functions

format_number(number) – Returns a formatted string using a unit symbol.

```
SELECT format_number(123456); -- '123K'
```

```
SELECT format_number(1000000); -- '1M'
```

Date and time functions

timezone_hour(timestamp) – Returns the hour of the time zone offset from timestamp.

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

timezone_minute(timestamp) – Returns the minute of the time zone offset from timestamp.

```
SELECT EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

Geospatial functions

to_encoded_polyline(Geometry) – Encodes a linestring or multipoint to a polyline.

```
SELECT to_encoded_polyline(ST_GeometryFromText(
```

```
    'LINESTRING (-120.2 38.5, -120.95 40.7, -126.453 43.252)'));
```

from_encoded_polyline(varchar) – Decodes a polyline to a linestring.

```
SELECT ST_AsText(from_encoded_polyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
```

to_geojson_geometry(SphericalGeography) – Returns the specified spherical geography in GeoJSON format.

```
SELECT to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
    'LINESTRING (0 0, 1 2, 3 4)')));
```

from_geojson_geometry(varchar) – Returns the spherical geography type object from the GeoJSON representation, stripping non geometry key/values. `Feature` and `FeatureCollection` are not supported.

```
SELECT
 from_geojson_geometry(to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
    'LINESTRING (0 0, 1 2, 3 4)'))));
```

geometry_nearest_points(Geometry, Geometry) – Returns the points on each geometry that are nearest each other. If either geometry is empty, returns NULL. Otherwise, returns a row of two `Point` objects that have the minimum distance of any two points on the geometries. The first point is from the first Geometry argument, the second from the second Geometry argument. If there are multiple pairs with the same minimum distance, one pair is chosen arbitrarily.

```
SELECT geometry_nearest_points(ST_GeometryFromText(
    'LINESTRING (50 100, 50 200)'), ST_GeometryFromText(
    'LINESTRING (10 10, 20 20)'));
```

Set Digest functions

make_set_digest(x) – Composes all input values of x into a setdigest.

```
SELECT make_set_digest(value) FROM (VALUES 1, 2, 3) T(value);
```

String functions

soundex(char) – Returns a character string that contains the phonetic representation of char.

```
SELECT name
FROM nation
WHERE SOUNDEX(name) = SOUNDEX('CHYNA'); -- CHINA
```

concat_ws(string0, string1, ..., stringN) – Returns the concatenation of `string1, string2, ..., stringN` using `string0` as a separator. If `string0` is null, then the return value is null. Any null values provided in the arguments after the separator are skipped.

```
SELECT concat_ws(',', 'def', 'pqr', 'mno');
```

Window functions

GROUPS – Adds support for window frames based on groups.

```
SELECT array_agg(a) OVER(
   ORDER BY a ASC NULLS FIRST GROUPS BETWEEN 1 PRECEDING AND 2 FOLLOWING)
FROM (VALUES 3, 3, 3, 2, 2, 1, null, null) T(a);
```

Performance improvements

Performance improvements in Athena engine version 3 include the following.

- **Faster AWS Glue table metadata retrieval –** Queries that involve multiple tables will see reduced query planning time.
- **Dynamic filtering for RIGHT JOIN –** Dynamic filtering is now enabled for right joins that have equality join conditions, as in the following example.

  ```
  SELECT *
  FROM lineitem RIGHT JOIN tpch.tiny.supplier
  ON lineitem.suppkey = supplier.suppkey
  WHERE supplier.name = 'abc';
  ```
- **Large prepared statements –** Increased the default HTTP request/response header size to 2 MB to allow large prepared statements.
- **approx_percentile() –** The `approx_percentile` function now uses `tdigest` instead of `qdigest` to retrieve approximate quantile values from distributions. This results in higher performance and lower memory usage. Note that as a result of this change, the function returns different results than it did in previous engine versions. For more information, see [The approx_percentile function returns different results](#).

Reliability enhancements

General engine memory usage and tracking in Athena engine version 3 have been improved. Large queries are less susceptible to failure from node crashes.

Query syntax enhancements

INTERSECT ALL – Added support for `INTERSECT ALL`.

```
SELECT * FROM (VALUES 1, 2, 3, 4) INTERSECT ALL SELECT * FROM (VALUES 3, 4);
```

EXCEPT ALL – Added support for `EXCEPT ALL`.

```
SELECT * FROM (VALUES 1, 2, 3, 4) EXCEPT ALL SELECT * FROM (VALUES 3, 4);
```

RANGE PRECEDING – Added support for `RANGE PRECEDING` in window functions.

```
SELECT sum(x) over (order by x range 1 preceding)
FROM (values (1), (1), (2), (2)) t(x);
```

MATCH_RECOGNIZE – Added support for row pattern matching, as in the following example.

```
SELECT m.id AS row_id, m.match, m.val, m.label
FROM (VALUES(1, 90),(2, 80),(3, 70),(4, 70)) t(id, value)
MATCH_RECOGNIZE (
        ORDER BY id
        MEASURES match_number() AS match,
        RUNNING LAST(value) AS val,
        classifier() AS label
        ALL ROWS PER MATCH
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (() | A) DEFINE A AS true
) AS m;
```

Data format and data type enhancements

Athena engine version 3 has the following data format and data type enhancements.

- **LZ4 and ZSTD** – Added support for reading LZ4 and ZSTD compressed Parquet data. Added support for writing ZSTD compressed ORC data.

- **Symlink-based tables** – Added support for creating symlink-based tables on Avro files. An example follows.

```
CREATE TABLE test_avro_symlink
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
...
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
```

- **SphericalGeography** – The SphericalGeography type provides native support for spatial features represented on geographic coordinates (sometimes called geodetic coordinates, lat/lon, or lon/lat). Geographic coordinates are spherical coordinates expressed in angular units (degrees).

  The `to_spherical_geography` function returns geographic (spherical) coordinates from geometric (planar) coordinates, as in the following example.

```
SELECT to_spherical_geography(ST_GeometryFromText(
   'LINESTRING (-40.2 28.9, -40.2 31.9, -37.2 31.9)'));
```

Breaking changes

When you migrate from previous engine versions to Athena engine version 3, certain changes can affect table schema, syntax, or data type usage. This section lists the associated error messages and provides suggested workarounds.

Query syntax changes

IGNORE NULLS cannot be used with non-value window functions

Error message: Cannot specify null treatment clause for `bool_or` function.

Cause: `IGNORE NULLS` can now be used only with the value functions `first_value`, `last_value`, `nth_value`, `lead`, and `lag`. This change was made to conform to the ANSI SQL specification.

Suggested solution: Remove `IGNORE NULLS` from non-value window functions in query strings.

CONCAT function must have two or more arguments

Error Message: INVALID_FUNCTION_ARGUMENT: There must be two or more concatenation arguments

Cause: Previously, the CONCAT string function accepted a single argument. In Athena engine version 3, the CONCAT function requires a minimum of two arguments.

Suggested solution: Change occurrences of CONCAT(str) to CONCAT(str, '').

In Athena engine version 3, functions can have no more than 127 arguments. For more information, see Too many arguments for function call.

The approx_percentile function returns different results

The approx_percentile function returns different results in Athena engine version 3 than it did in previous engine versions.

Error message: None.

Cause: The approx_percentile function is subject to version changes.

> **Important**
>
> Because the outputs of the approx_percentile function are approximations, and the approximations are subject to change from one version to the next, you should not rely on the approx_percentile function for critical applications.

Suggested Solution: To approximate the previous engine versions behavior of approx_percentile, you can use a different set of functions in Athena engine version 3. For example, suppose you have the following query in previous engine versions:

```
SELECT approx_percentile(somecol, 2E-1)
```

To approximate the same output in Athena engine version 3, you can try the qdigest_agg and value_at_quantile functions, as in the following example. Note that, even with this workaround, the same behavior is not guaranteed.

```
SELECT value_at_quantile(qdigest_agg(somecol, 1), 2E-1)
```

Geospatial function does not support varbinary input

Error message: FUNCTION_NOT_FOUND for st_XXX

Cause: A few geospatial functions no longer support the legacy VARBINARY input type or text related function signatures.

Suggested solution: Use geospatial functions to convert the input types to types that are supported. Supported input types are indicated in the error message.

In GROUP BY clauses, nested columns must be double quoted

Error message: "*column_name*"."*nested_column*" must be an aggregate expression or appear in GROUP BY clause

Cause: Athena engine version 3 requires that nested column names in GROUP BY clauses be double quoted. For example, the following query produces the error because, in the GROUP BY clause, user.name is not double quoted.

```
SELECT "user"."name" FROM dataset
GROUP BY user.name
```

Suggested solution: Place double quotes around nested column names in GROUP BY clauses, as in the following example.

```
SELECT "user"."name" FROM dataset
GROUP BY "user"."name"
```

Unexpected FilterNode error when using OPTIMIZE on an Iceberg table

Error message: Unexpected FilterNode found in plan; probably connector was not able to handle provided WHERE expression.

Cause: The OPTIMIZE statement that was run on the Iceberg table used a WHERE clause that included a non-partition column in its filter expression.

Suggested Solution: The OPTIMIZE statement supports filtering by partitions only. When you run OPTIMIZE on partitioned tables, include only partition columns in the WHERE clause. If you run OPTIMIZE on a non-partitioned table, do not specify a WHERE clause.

Log() function order of arguments

In Athena engine version 3, the order of arguments for the log() function has changed to log(*base*, *value*) in conformance with SQL standards.

Minute() function does not support interval year to month

Error message: Unexpected parameters (interval year to month) for function minute. Expected: minute(timestamp with time zone) , minute(time with time zone) , minute(timestamp) , minute(time) , minute(interval day to second).

Cause: In Athena engine version 3, type checks have been made more precise for `EXTRACT` in accordance with the ANSI SQL specification.

Suggested solution: Update the queries to make sure types are matched with the suggested function signatures.

ORDER BY expressions must appear in SELECT list

Error message: For SELECT DISTINCT, ORDER BY expressions must appear in SELECT list

Cause: Incorrect table aliasing is used in a `SELECT` clause.

Suggested solution: Double check that all columns in the `ORDER BY` expression have proper references in the `SELECT DISTINCT` clause.

Query failure when comparing multiple columns returned from a subquery

Example error message: Value expression and result of subquery must be of the same type: row(varchar, varchar) vs row(row(varchar, varchar))

Cause: Due to a syntax update in Athena engine version 3, this error occurs when a query tries to compare multiple values returned from a subquery, and the subquery `SELECT` statement encloses its list of columns in parentheses, as in the following example.

```
SELECT *
FROM table1
WHERE (t1_col1, t1_col2)
IN (SELECT (t2_col1, t2_col2) FROM table2)
```

Solution: In Athena engine version 3, remove the parenthesis around the list of columns in the subquery `SELECT` statement, as in the following updated example query.

```
SELECT *
FROM table1
```

```
WHERE (t1_col1, t1_col2)
IN (SELECT t2_col1, t2_col2 FROM table2)
```

SKIP is a reserved word for DML queries

The word SKIP is now a reserved word for DML queries like SELECT. To use SKIP as an identifier in a DML query, enclose it in double quotes.

For more information about reserved words in Athena, see [Escape reserved keywords in queries].

SYSTEM_TIME and SYSTEM_VERSION clauses deprecated for time travel

Error message: mismatched input 'SYSTEM_TIME'. Expecting: 'TIMESTAMP', 'VERSION'

Cause: In previous engine versions, Iceberg tables used the FOR SYSTEM_TIME AS OF and FOR SYSTEM_VERSION AS OF clauses for timestamp and version time travel. Athena engine version 3 uses the FOR TIMESTAMP AS OF and FOR VERSION AS OF clauses.

Suggested solution: Update the SQL query to use the TIMESTAMP AS OF and VERSION AS OF clauses for time travel operations, as in the following examples.

Time travel by timestamp:

```
SELECT * FROM TABLE FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

Time travel by version:

```
SELECT * FROM TABLE FOR VERSION AS OF 949530903748831860
```

Too many arguments for an array constructor

Error Message: TOO_MANY_ARGUMENTS: Too many arguments for array constructor.

Cause: The maximum number of elements in an array constructor is now set at 254.

Suggested solution: Break up the elements into multiple arrays that have 254 or fewer elements each, and use the CONCAT function to concatenate the arrays, as in the following example.

```
CONCAT(
ARRAY[x1,x2,x3...x254],
```

```
ARRAY[y1,y2,y3...y254],
...
)
```

Zero-length delimited identifier not allowed

Error message: Zero-length delimited identifier not allowed.

Cause: A query used an empty string as a column alias.

Suggested solution: Update the query to use a non-empty alias for the column.

Data processing changes

Bucket validation

Error Message: HIVE_INVALID_BUCKET_FILES: Hive table is corrupt.

Cause: The table might have been corrupted. To ensure query correctness for bucketed tables, Athena engine version 3 enables additional validation on bucketed tables to ensure query correctness and avoid unexpected failures at runtime.

Suggested solution: Re-create the table using Athena engine version 3.

Casting a struct to JSON now returns field names

When you cast a `struct` to JSON in a `SELECT` query in Athena engine version 3, the cast now returns both the field names and the values (for example `"useragent":null` instead of just the values (for example, `null`).

Iceberg table column level security enforcement change

Error Message: Access Denied: Cannot select from columns

Cause: The Iceberg table was created outside Athena and uses an [Apache Iceberg SDK] version earlier than 0.13.0. Because earlier SDK versions do not populate columns in AWS Glue, Lake Formation could not determine the columns authorized for access.

Suggested solution: Perform an update using the Athena [ALTER TABLE SET TBLPROPERTIES] statement or use the latest Iceberg SDK to fix the table and update the column information in AWS Glue.

Nulls in List data types are now propagated to UDFs

Error message: Null Pointer Exception

Cause: This issue can affect you if you use the UDF connector and have implemented a user defined Lambda function.

The previous engine versions filtered out the nulls in List data types that were passed to a user defined function. In Athena engine version 3, the nulls are now preserved and passed on to the UDF. This can cause a null pointer exception if the UDF attempts to dereference the null element without checking.

For example, if you have the data `[null, 1, null, 2, 3, 4]` in an originating data source like DynamoDB, the following are passed to the user-defined Lambda function:

Athena engine version 3: `[null, 1, null, 2, 3, 4]`

Suggested solution: Ensure that your user-defined Lambda function handles null elements in list data types.

Substrings from character arrays no longer contain padded spaces

Error message: No error is thrown, but the string returned no longer contains padded spaces. For example, `substr(char[20],1,100)` now returns a string with length 20 instead of 100.

Suggested solution: No action is required.

Unsupported decimal column type coercion

Error messages: HIVE_CURSOR_ERROR: Failed to read Parquet file: s3://amzn-s3-demo-bucket/*`path`*/*`file_name`*.parquet or Unsupported column type (varchar) for Parquet column ([*`column_name`*]

Cause: Athena engine version 2 occasionally succeeded (but frequently failed) when attempting data type coercions from `varchar` to decimal. Because Athena engine version 3 has type validation that checks that the type is compatible before it tries to read the value, such attempted coercions now always fail.

Suggested Solution: For Athena engine version 3, modify your schema in AWS Glue to use a numeric data type instead of `varchar` for decimal columns in Parquet files. Either recrawl the data and ensure that the new column data type is a decimal type, or manually re-create the table in Athena and use the syntax `decimal(precision, scale)` to specify a decimal data type for the column.

Float or double NaN values can no longer be cast to bigint

Error Message: INVALID_CAST_ARGUMENT: Cannot cast real/double NaN to bigint

Cause: In Athena engine version 3, `NaN` can no longer be cast to 0 as `bigint`.

Suggested solution: Make sure that `NaN` values are not present in `float` or `double` columns when you cast to `bigint`.

uuid() function return type change

The following issue affects both tables and views.

Error message: Unsupported Hive type: uuid

Cause: In previous engine versions, the `uuid()` function returned a string, but in Athena engine version 3, it returns a pseudo randomly generated UUID (type 4). Because the UUID column data type is not supported in Athena, the `uuid()` function can no longer be used directly in CTAS queries to generate UUID columns in Athena engine version 3.

For example, the following `CREATE TABLE` statement completes successfully in previous engine versions but returns NOT_SUPPORTED: Unsupported Hive type: uuid in Athena engine version 3:

```
CREATE TABLE uuid_table AS
   SELECT uuid() AS myuuid
```

Similarly, the following `CREATE VIEW` statement completed successfully in Athena engine version 2 but returns Invalid column type for column myuuid: Unsupported Hive type: uuid in Athena engine version 3:

```
CREATE VIEW uuid_view AS
   SELECT uuid() AS myuuid
```

When a view so created in previous engine versions is queried in Athena engine version 3, an error like the following occurs:

VIEW_IS_STALE: line 1:15: View 'awsdatacatalog.mydatabase.uuid_view' is stale or in invalid state: column [myuuid] of type uuid projected from query view at position 0 cannot be coerced to column [myuuid] of type varchar stored in view definition

Suggested Solution: When you create the table or view, use the `cast()` function to convert the output of `uuid()` to a `varchar`, as in the following examples:

```
CREATE TABLE uuid_table AS
   SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

```
CREATE VIEW uuid_view AS
   SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

CHAR and VARCHAR coercion issues

Use the workarounds in this section if you encounter coercion issues with `varchar` and `char` in Athena engine version 3. If you are unable to use these workarounds, please contact Support.

CONCAT function failure with mixed CHAR and VARCHAR inputs

Issue: The following query succeeds on Athena engine version 2.

```
SELECT concat(CAST('abc' AS VARCHAR(20)), '12', CAST('a' AS CHAR(1)))
```

However, on Athena engine version 3, the same query fails with the following:

Error message: FUNCTION_NOT_FOUND: line 1:8: Unexpected parameters (varchar(20), varchar(2), char(1)) for function concat. Expected: concat(char(x), char(y)), concat(array(E), E) E, concat(E, array(E)) E, concat(array(E)) E, concat(varchar), concat(varbinary)

Suggested Solution: When using the `concat` function, cast to `char` or `varchar`, but not to a mix of both.

SQL || concatenation failure with CHAR and VARCHAR inputs

In Athena engine version 3, the double vertical bar `||` concatenation operator requires `varchar` as inputs. The inputs cannot be a combination of `varchar` and `char` types.

Error message: TYPE_NOT_FOUND: line 1:26: Unknown type: char(65537)

Cause: A query that uses **||** to concatenate a `char` and a `varchar` can produce the error, as in the following example.

```
SELECT CAST('a' AS CHAR) || CAST('b' AS VARCHAR)
```

Suggested Solution: Concatenate `varchar` with `varchar`, as in the following example.

```
SELECT CAST('a' AS VARCHAR) || CAST('b' AS VARCHAR)
```

CHAR and VARCHAR UNION query failure

Error message: NOT_SUPPORTED: Unsupported Hive type: char(65536). Supported CHAR types: CHAR(<=255)

Cause: A query that attempts to combine `char` and `varchar`, as in the following example:

```
CREATE TABLE t1 (c1) AS SELECT CAST('a' as CHAR) as c1 UNION ALL SELECT CAST('b' AS
 VARCHAR) AS c1
```

Suggested Solution: In the example query, cast `'a'` as `varchar` rather than `char`.

Unwanted empty spaces after CHAR or VARCHAR coercion

In Athena engine version 3, when `char(X)` and `varchar` data are coerced to a single type when forming an array or single column, `char(65535)` is the target type, and each field contains many unwanted trailing spaces.

Cause: Athena engine version 3 coerces `varchar` and `char(X)` to `char(65535)` and then right pads the data with spaces.

Suggested Solution: Cast each field explicitly to `varchar`.

Timestamp changes

Date timestamp overflow throws error

Error message: Millis overflow: XXX

Cause: Because ISO 8601 dates were not checked for overflow in previous engine versions, some dates produced a negative timestamp. Athena engine version 3 checks for this overflow and throws an exception.

Suggested Solution: Make sure the timestamp is within range.

Political time zones with TIME not supported

Error message: INVALID LITERAL

Cause: Queries like `SELECT TIME '13:21:32.424 America/Los_Angeles'`.

Suggested solution: Avoid using political time zones with `TIME`.

Precision mismatch in Timestamp columns causes serialization error

Error message: SERIALIZATION_ERROR: Could not serialize column '*COLUMNZ*' of type 'timestamp(3)' at position *X*:*Y*

COLUMNZ is the output name of the column that causes the issue. The numbers *X*:*Y* indicate the position of the column in the output.

Cause: Athena engine version 3 checks to make sure that the precision of timestamps in the data is the same as the precision specified for the column data type in the table specification. Currently, this precision is always 3. If the data has a precision greater than this, queries fail with the error noted.

Suggested solution: Check your data to make sure that your timestamps have millisecond precision.

Incorrect timestamp precision in UNLOAD and CTAS queries for Iceberg tables

Error message: Incorrect timestamp precision for timestamp(6); the configured precision is MILLISECONDS

Cause: Athena engine version 3 checks to make sure that the precision of timestamps in the data is the same as the precision specified for the column data type in the table specification. Currently, this precision is always 3. If the data has a precision greater than this (for example, microseconds instead of milliseconds), queries can fail with the error noted.

Solution: To work around this issue, first `CAST` the timestamp precision to 6, as in the following CTAS example that creates an Iceberg table. Note that the precision must be specified as 6 instead of 3 to avoid the error Timestamp precision (3) not supported for Iceberg.

```
CREATE TABLE my_iceberg_ctas
```

```
WITH (table_type = 'ICEBERG', location = 's3://amzn-s3-demo-bucket/table_ctas/',
format = 'PARQUET')
AS SELECT id, CAST(dt AS timestamp(6)) AS "dt"
FROM my_iceberg
```

Then, because Athena does not support timestamp 6, cast the value again to timestamp (for example, in a view). The following example creates a view from the `my_iceberg_ctas` table.

```
CREATE OR REPLACE VIEW my_iceberg_ctas_view AS
SELECT cast(dt AS timestamp) AS dt
FROM my_iceberg_ctas
```

Reading the Long type as Timestamp or vice versa in ORC files now causes a malformed ORC file error

Error message: Error opening Hive split ‘FILE (SPLIT POSITION)’ Malformed ORC file. Cannot read SQL type timestamp from ORC stream .long_type of type LONG

Cause: Athena engine version 3 now rejects implicit coercion from the `Long` data type to `Timestamp` or from `Timestamp` to `Long`. Previously, `Long` values were implicitly converted into timestamp as if they were epoch milliseconds.

Suggested solution: Use the `from_unixtime` function to explicitly cast the column, or use the `from_unixtime` function to create an additional column for future queries.

Time and interval year to month not supported

Error message: TYPE MISMATCH

Cause: Athena engine version 3 does not support time and interval year to month (for example, `SELECT TIME '01:00' + INTERVAL '3' MONTH`).

Timestamp overflow for int96 Parquet format

Error message: Invalid timeOfDayNanos

Cause: A timestamp overflow for the `int96` Parquet format.

Suggested solution: Identify the specific files that have the issue. Then generate the data file again with an up-to-date, well known Parquet library, or use Athena CTAS. If the issue persists, contact Athena support and let us know how the data files are generated.

Space required between date and time values when casting from string to timestamp

Error message: INVALID_CAST_ARGUMENT: Value cannot be cast to timestamp.

Cause: Athena engine version 3 no longer accepts a hyphen as a valid separator between date and time values in the input string to `cast`. For example, the following query does not work in Athena engine version 3:

```
SELECT CAST('2021-06-06-23:38:46' AS timestamp) AS this_time
```

Suggested solution: In Athena engine version 3, replace the hyphen between the date and the time with a space, as in the following example.

```
SELECT CAST('2021-06-06 23:38:46' AS timestamp) AS this_time
```

to_iso8601() timestamp return value change

Error message: None

Cause: In previous engine versions, the `to_iso8601` function returns a timestamp with time zone even if the value passed to the function does not include the time zone. In Athena engine version 3, the `to_iso8601` function returns a timestamp with time zone only when the argument passed includes the time zone.

For example, the following query passes the current date to the `to_iso8601` function twice: first as a timestamp with time zone, and then as a timestamp.

```
SELECT TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP WITH TIME ZONE)),
 TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP))
```

The following output shows the result of the query in Athena engine version 3.

In previous engine versions:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000Z

Athena engine version 3:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000

Suggested solution: To replicate the previous behaviour, you can pass the timestamp value to the `with_timezone` function before passing it to `to_iso8601`, as in the following example:

```
SELECT to_iso8601(with_timezone(TIMESTAMP '2023-01-01 00:00:00.000', 'UTC'))
```

Result

#	_col0
1	2023-01-01T00:00:00.000Z

at_timezone() first parameter must specify a date

Issue: In Athena engine version 3, the `at_timezone` function cannot take a `time_with_timezone` value as the first parameter.

Cause: Without date information, it cannot be determined whether the value passed is daylight time or standard time. For example, `at_timezone('12:00:00 UTC', 'America/Los_Angeles')` is ambiguous since there is no way to determine whether the value passed is Pacific Daylight Time (PDT) or Pacific Standard Time (PST).

Limitations

Athena engine version 3 has the following limitations.

- **Query performance** – Many queries run faster on Athena engine version 3, but some query plans can differ from previous engine versions. As a result, some queries can differ in latency or cost.
- **Trino and Presto connectors** – Neither Trino nor Presto connectors are supported. Use Amazon Athena Federated Query to connect data sources. For more information, see Use Amazon Athena Federated Query.
- **Fault-tolerant execution** – Trino fault-tolerant execution (Trino Tardigrade) is not supported.

- **Function parameter limit** – Functions cannot take more than 127 parameters. For more information, see Too many arguments for function call.

The following limits were introduced in Athena engine version 2 to ensure that queries do not fail due to resource limitations. These limits are not configurable by users.

- **Number of result elements** – The number of result elements `n` is restricted to 10,000 or less for the following functions: `min(col, n)`, `max(col, n)`, `min_by(col1, col2, n)`, and `max_by(col1, col2, n)`.
- **GROUPING SETS** – The maximum number of slices in a grouping set is 2048.
- **Maximum text file line length** – The default maximum line length for text files is 200 MB.
- **Sequence function maximum result size** – The maximum result size of a sequence function is 50000 entries. For example, `SELECT sequence(0,45000,1)` succeeds, but `SELECT sequence(0,55000,1)` fails with the error message The result of the sequence function must not have more than 50000 entries. This limit applies to all input types for sequence functions, including timestamps.

SQL reference for Athena

Amazon Athena supports a subset of Data Definition Language (DDL) and Data Manipulation Language (DML) statements, functions, operators, and data types. With some exceptions, Athena DDL is based on HiveQL DDL and Athena DML is based on Trino. For information about Athena engine versions, see Athena engine versioning.

Topics

Data types in Amazon Athena

When you run `CREATE TABLE`, you specify column names and the data type that each column can contain. The tables that you create are stored in the AWS Glue Data Catalog.

To facilitate interoperability with other query engines, Athena uses [Apache Hive](#) data type names for DDL statements like `CREATE TABLE`. For DML queries like `SELECT`, `CTAS`, and `INSERT INTO`, Athena uses [Trino](#) data type names. The following table shows the data types supported in Athena. Where DDL and DML types differ in terms of name, availability, or syntax, they are shown in separate columns.

DDL	DML	Description
BOOLEAN		Values are `true` and `false`.
TINYINT		An 8-bit signed integer in two's complement format, with a minimum value of -2^7 and a maximum value of 2^7-1.
SMALLINT		A 16-bit signed integer in two's complement format, with a minimum value of -2^{15} and a maximum value of $2^{15}-1$.
INT, INTEGER		A 32-bit signed value in two's complement format, with a minimum value of -2^{31} and a maximum value of $2^{31}-1$.
BIGINT		A 64-bit signed integer in two's complement format, with a minimum value of -2^{63} and a maximum value of $2^{63}-1$.
FLOAT	REAL	A 32-bit signed single-precision floating point number. The range is 1.40129846432481707e-45 to 3.4028234 6638528860e+38, positive or negative. Follows the IEEE Standard for Floating-Point Arithmetic (IEEE 754).
DOUBLE		A 64-bit signed double-precision floating point number. The range is 4.94065645841246544e-324d to 1.79769313486231570e+308d, positive or negative. Follows the IEEE Standard for Floating-Point Arithmetic (IEEE 754).
DECIMAL(*precision* , *scale*)		*precision* is the total number of digits. *scale* (optional) is the number of digits in fractional part with a default of 0. For example, use these type definitio

DDL	DML	Description
		ns: `decimal(11,5)` , `decimal(15)` . The maximum value for *precision* is 38, and the maximum value for *scale* is 38.
CHAR, CHAR(*length*)		Fixed length character data, with a specified length between 1 and 255, such as char(10). If *length* is specified, strings are truncated at the specified length when read. If the underlying data string is longer, the underlying data string remains unchanged. For more information, see CHAR Hive data type.
STRING	VARCHAR	Variable length character data.
VARCHAR(*length*)		Variable length character data with a maximum read length. Strings are truncated at the specified length when read. If the underlying data string is longer, the underlying data string remains unchanged.
BINARY	VARBINARY	Variable length binary data.
Not available	TIME	A time of day with millisecond precision.
Not available	TIME(*precision*)	A time of day with a specific precision. `TIME(3)` is equivalent to `TIME`.
Not available	TIME WITH TIME ZONE	A time of day in a time zone. Time zones should be specified as offsets from UTC.
DATE		A calendar date with year, month, and day.
TIMESTAMP	TIMESTAMP , TIMESTAMP WITHOUT TIME ZONE	A calendar date and time of day with millisecond precision.

DDL	DML	Description
Not available	TIMESTAMP (*precision*), TIMESTAMP (*precision*) WITHOUT TIME ZONE	A calendar date and time of day with a specific precision. `TIMESTAMP(3)` is equivalent to `TIMESTAMP`.
Not available	TIMESTAMP WITH TIME ZONE	A calendar date and time of day in a time zone. Time zones can be specified as offsets from UTC, as IANA time zone names, or using UTC, UT, Z, or GMT.
Not available	TIMESTAMP (*precision*) WITH TIME ZONE	A calendar date and time of day with a specific precision, in a time zone.
Not available	INTERVAL YEAR TO MONTH	An interval of one or more whole months
Not available	INTERVAL DAY TO SECOND	An interval of one or more seconds, minutes, hours, or days
ARRAY<*element_type*>	ARRAY[*element_type*]	An array of values. All values must be of the same data type.
MAP<*key_type*, *value_type*>	MAP(*key_type*, *value_type*)	A map where values can be looked up by key. All keys must be of the same data type, and all values must be of the same data type. For example, `map<string, integer>`.

DDL	DML	Description
STRUCT<*field_name_1* :*field_type_1* , *field_name_2* :*field_type_2* , ...>	ROW(*field_name_1* *field_type_1* , *field_name_2* *field_type_2* , ...)	A data structure with named fields and their values.
Not available	JSON	JSON value type, which can be a JSON object, a JSON array, a JSON number, a JSON string, `true`, `false` or `null`.
Not available	UUID	A UUID (Universally Unique IDentifier).
Not available	IPADDRESS	An IPv4 or IPv6 address.
Not available	HyperLogLog P4HyperLogLog SetDigest QDigest TDigest	These data types support approximate function internals. For more information about each type, visit the link to the corresponding entry in the Trino documentation.

Topics

- Data type examples
- Considerations for data types
- Work with timestamp data

Data type examples

The following table shows example literals for DML data types.

Data type	Examples
BOOLEAN	`true` `false`
TINYINT	`TINYINT '123'`
SMALLINT	`SMALLINT '123'`
INT, INTEGER	`123456790`
BIGINT	`BIGINT '1234567890'` `2147483648`
REAL	`'123456.78'`
DOUBLE	`1.234`
DECIMAL(*precision* , *scale*)	`DECIMAL '123.456'`
CHAR, CHAR(*length*)	`CHAR 'hello world'`, `CHAR 'hello ''world''!'`
VARCHAR, VARCHAR(*length*)	`VARCHAR 'hello world'`, `VARCHAR 'hello ''world''!'`
VARBINARY	`X'00 01 02'`
TIME, TIME(*precision*)	`TIME '10:11:12'` , `TIME '10:11:12.345'`
TIME WITH TIME ZONE	`TIME '10:11:12.345 -06:00'`
DATE	`DATE '2024-03-25'`
TIMESTAMP, TIMESTAMP WITHOUT TIME ZONE, TIMESTAMP(*precision*), TIMESTAMP(*precision*) WITHOUT TIME ZONE	`TIMESTAMP '2024-03-25 11:12:13'` , `TIMESTAMP '2024-03-25 11:12:13.456'`

Data type	Examples
TIMESTAMP WITH TIME ZONE, TIMESTAMP (*precision*) WITH TIME ZONE	`TIMESTAMP '2024-03-25 11:12:13.456 Europe/Berlin'`
INTERVAL YEAR TO MONTH	`INTERVAL '3' MONTH`
INTERVAL DAY TO SECOND	`INTERVAL '2' DAY`
ARRAY[*element_type*]	`ARRAY['one', 'two', 'three']`

MAP(*key_type*, *value_type*)	`MAP(ARRAY['one', 'two', 'three'], ARRAY[1, 2, 3])`
	
	Note that maps are created from an array of keys and an array of values. The following example creates a table that maps strings to integers.
	
	`CREATE TABLE map_table(col1 map<string, integer>) LOCATION '...';`
	`INSERT INTO map_table values(MAP(ARRAY['foo', 'bar'], ARRAY[1, 2]));`

ROW(*field_name_1 field_type_1*, *field_name_2 field_type_2*, …)	`ROW('one', 'two', 'three')`
	
	Note that rows created this way have no column names. To add column names, you can use CAST, as in the following example:
	
	`CAST(ROW(1, 2, 3) AS ROW(one INT, two INT, three INT))`

JSON	`JSON '{"one":1, "two": 2, "three": 3}'`
UUID	`UUID '12345678-90ab-cdef-1234-567890abcdef'`

IPADDRESS	`IPADDRESS '10.0.0.1'`
	
	`IPADDRESS '2001:db8::1'`

Considerations for data types

Size limits

For data types that do not specify a size limit, keep in mind that there is a practical limit of 32MB for all of the data in a single row. For more information, see Row or column size limitation in Considerations and limitations for SQL queries in Amazon Athena.

CHAR and VARCHAR

A CHAR(*n*) value always has a count of *n* characters. For example, if you cast 'abc' to CHAR(7), 4 trailing spaces are added.

Comparisons of CHAR values include leading and trailing spaces.

If a length is specified for CHAR or VARCHAR, strings are truncated at the specified length when read. If the underlying data string is longer, the underlying data string remains unchanged.

To escape a single quote in a CHAR or VARCHAR, use an additional single quote.

To cast a non-string data type to a string in a DML query, cast to the VARCHAR data type.

To use the `substr` function to return a substring of specified length from a CHAR data type, you must first cast the CHAR value as a VARCHAR. In the following example, `col1` uses the CHAR data type.

```
substr(CAST(col1 AS VARCHAR), 1, 4)
```

DECIMAL

To specify decimal values as literals in SELECT queries, such as when selecting rows with a specific decimal value, you can specify the DECIMAL type and list the decimal value as a literal in single quotes in your query, as in the following examples.

```
SELECT * FROM my_table
WHERE decimal_value = DECIMAL '0.12'
```

```
SELECT DECIMAL '44.6' + DECIMAL '77.2'
```

Work with timestamp data

This section describes some considerations for working with timestamp data in Athena.

> **Note**
>
> The treatment of timestamps has changed somewhat between previous engine versions and Athena engine version 3. For information about timestamp-related errors that can occur in Athena engine version 3 and suggested solutions, see Timestamp changes in the Athena engine version 3 reference.

Format for writing timestamp data to Amazon S3 objects

The format in which timestamp data should be written into Amazon S3 objects depends on both the column data type and the SerDe library that you use.

- If you have a table column of type `DATE`, Athena expects the corresponding column or property of the data to be a string in the ISO format `YYYY-MM-DD`, or a built-in date type like those for Parquet or ORC.
- If you have a table column of type `TIME`, Athena expects the corresponding column or property of the data to be a string in the ISO format `HH:MM:SS`, or a built-in time type like those for Parquet or ORC.
- If you have a table column of type `TIMESTAMP`, Athena expects the corresponding column or property of the data to be a string in the format `YYYY-MM-DD HH:MM:SS.SSS` (note the space between the date and time), or a built-in time type like those for Parquet, ORC, or Ion. Note that Athena does not guarantee the behavior for timestamps that are invalid (for example, `0000-00-00 08:00:00.000`).

  > **Note**
  >
  > OpenCSVSerDe timestamps are an exception and must be encoded as millisecond resolution UNIX epochs.

Ensuring that time-partitioned data matches the timestamp field in a record

The producer of the data must make sure partition values align with the data within the partition. For example, if your data has a `timestamp` property and you use Firehose to load the data into Amazon S3, you must use dynamic partitioning because the default partitioning of Firehose is wall-clock-based.

Use string as the data type for partition keys

For performance reasons, it is preferable to use STRING as the data type for partition keys. Even though Athena recognizes partition values in the format YYYY-MM-DD as dates when you use the DATE type, this can lead to poor performance. For this reason, we recommend that you use the STRING data type for partition keys instead.

How to write queries for timestamp fields that are also time-partitioned

How you write queries for timestamp fields that are time-partitioned depends on the type of table that you want to query.

Hive tables

With the Hive tables most commonly used in Athena, the query engine has no knowledge of relationships between columns and partition keys. For this reason, you must always add predicates in your queries for both the column and the partition key.

For example, suppose you have an event_time column and an event_date partition key and want to query events between 23:00 and 03:00. In this case, you must include predicates in your query for both the column and the partition key, as in the following example.

```
WHERE event_time BETWEEN start_time AND end_time
  AND event_date BETWEEN start_time_date AND end_time_date
```

Iceberg tables

With Iceberg tables, you can use computed partition values, which simplifies your queries. For example, suppose your Iceberg table was created with a PARTITIONED BY clause like the following:

```
PARTITIONED BY (event_date month(event_time))
```

In this case, the query engine automatically prunes partitions based on the values of the event_time predicates. Because of this, your query only needs to specify a predicate for event_time, as in the following example.

```
WHERE event_time BETWEEN start_time AND end_time
```

For more information, see Create Iceberg tables.

When using Iceberg's hidden partitioning for a timestamp column, Iceberg might create a partition on a constructed table column derived from a timestamp column and transformed into a date for more effective partitioning. For example, it might create `event_date` from the timestamp column `event_time` and automatically partition on `event_date`. In this case, the partition **type** is a **date**.

For optimal query performance when you use the partition, filter on full day ranges to enable predicate pushdown. For example, the following query wouldn't be pushed down because the range can't be converted to a single date partition, even though it falls within a single day:

```
WHERE event_time >= TIMESTAMP '2024-04-18 00:00:00' AND event_time < TIMESTAMP
 '2024-04-18 12:00:00'
```

Instead, use a full day range to allow predicate pushdown and improve query performance as in following example.

```
WHERE event_time >= TIMESTAMP '2024-04-18 00:00:00' AND event_time < TIMESTAMP
 '2024-04-19 00:00:00'
```

You can also use the `BETWEEN start_time AND end_time` syntax or use the multi-day ranges as long as the timestamps portions are `00:00:00`.

For more information, see the Trino blog post.

DML queries, functions, and operators

The Athena DML query engine generally supports Trino and Presto syntax and adds its own improvements. Athena does not support all Trino or Presto features. For more information, see the topics for specific statements in this section and Considerations and limitations. For information about functions, see Functions in Amazon Athena. For information about Athena engine versions, see Athena engine versioning.

For information about DDL statements, see DDL statements. For a list of unsupported DDL statements, see Unsupported DDL.

Topics

- DELETE
- UPDATE
- MERGE INTO
- OPTIMIZE
- VACUUM
- Using EXPLAIN and EXPLAIN ANALYZE in Athena
- PREPARE
- UNLOAD
- Functions in Amazon Athena
- Use supported time zones

SELECT

Retrieves rows of data from zero or more tables.

> **Note**
>
> This topic provides summary information for reference. Comprehensive information about using SELECT and the SQL language is beyond the scope of this documentation. For information about using SQL that is specific to Athena, see Considerations and limitations for SQL queries in Amazon Athena and Run SQL queries in Amazon Athena. For an example of creating a database, creating a table, and running a SELECT query on the table in Athena, see Get started.

Synopsis

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
```

```
[ LIMIT [ count | ALL ] ]
```

Note

Reserved words in SQL SELECT statements must be enclosed in double quotes. For more information, see Reserved keywords to escape in SQL SELECT statements.

Parameters

[WITH with_query [, ...]]

You can use `WITH` to flatten nested queries, or to simplify subqueries.

Using the `WITH` clause to create recursive queries is supported starting in Athena engine version 3. The maximum recursion depth is 10.

The `WITH` clause precedes the `SELECT` list in a query and defines one or more subqueries for use within the `SELECT` query.

Each subquery defines a temporary table, similar to a view definition, which you can reference in the `FROM` clause. The tables are used only when the query runs.

`with_query` syntax is:

```
subquery_table_name [ ( column_name [, ...] ) ] AS (subquery)
```

Where:

- `subquery_table_name` is a unique name for a temporary table that defines the results of the `WITH` clause subquery. Each `subquery` must have a table name that can be referenced in the `FROM` clause.
- `column_name [, ...]` is an optional list of output column names. The number of column names must be equal to or less than the number of columns defined by `subquery`.
- `subquery` is any query statement.

[ALL | DISTINCT] select_expression

`select_expression` determines the rows to be selected. A `select_expression` can use one of the following formats:

```
expression [ [ AS ] column_alias ] [, ...]
```

```
row_expression.* [ AS ( column_alias [, ...] ) ]
```

```
relation.*
```

```
*
```

- The `expression [ [ AS ] column_alias ]` syntax specifies an output column. The optional `[AS] column_alias` syntax specifies a custom heading name to be used for the column in the output.
- For `row_expression.* [ AS ( column_alias [, ...] ) ]`, `row_expression` is an arbitrary expression of data type `ROW`. The fields of the row define the output columns to be included in the result.
- For `relation.*`, the columns of `relation` are included in the result. This syntax does not permit the use of column aliases.
- The asterisk `*` specifies that all columns be included in the result set.
- In the result set, the order of columns is the same as the order of their specification by the select expression. If a select expression returns multiple columns, the column order follows the order used in the source relation or row type expression.
- When column aliases are specified, the aliases override preexisting column or row field names. If the select expression does not have column names, zero-indexed anonymous column names (`_col0`, `_col1`, `_col2, ...`) are displayed in the output.
- `ALL` is the default. Using `ALL` is treated the same as if it were omitted; all rows for all columns are selected and duplicates are kept.
- Use `DISTINCT` to return only distinct values when a column contains duplicate values.

FROM from_item [, ...]

Indicates the input to the query, where `from_item` can be a view, a join construct, or a subquery as described below.

The `from_item` can be either:

- `table_name [ [ AS ] alias [ (column_alias [, ...]) ] ]`

Where `table_name` is the name of the target table from which to select rows, `alias` is the name to give the output of the `SELECT` statement, and `column_alias` defines the columns for the `alias` specified.

-OR-

- `join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]`

  Where `join_type` is one of:

  - `[ INNER ] JOIN`
  - `LEFT [ OUTER ] JOIN`
  - `RIGHT [ OUTER ] JOIN`
  - `FULL [ OUTER ] JOIN`
  - `CROSS JOIN`
  - `ON join_condition | USING (join_column [, ...])` Where using `join_condition` allows you to specify column names for join keys in multiple tables, and using `join_column` requires `join_column` to exist in both tables.

[WHERE condition]

Filters results according to the `condition` you specify, where `condition` generally has the following syntax.

```
column_name operator value [[[AND | OR] column_name operator value] ...]
```

The *`operator`* can be one of the comparators =, >, <, >=, <=, <>, !=.

The following subquery expressions can also be used in the `WHERE` clause.

- `[NOT] BETWEEN` *`integer_A`* `AND` *`integer_B`* – Specifies a range between two integers, as in the following example. If the column data type is `varchar`, the column must be cast to integer first.

  ```
  SELECT DISTINCT processid FROM "webdata"."impressions"
  WHERE cast(processid as int) BETWEEN 1500 and 1800
  ORDER BY processid
  ```

- `[NOT] LIKE` *`value`* – Searches for the pattern specified. Use the percent sign (%) as a wildcard character, as in the following example.

```
SELECT * FROM "webdata"."impressions"
WHERE referrer LIKE '%.org'
```

- `[NOT] IN (value[, value[, ...])` – Specifies a list of possible values for a column, as in the following example.

```
SELECT * FROM "webdata"."impressions"
WHERE referrer IN ('example.com','example.net','example.org')
```

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]]

Divides the output of the `SELECT` statement into rows with matching values.

`ALL` and `DISTINCT` determine whether duplicate grouping sets each produce distinct output rows. If omitted, `ALL` is assumed.

`grouping_expressions` allow you to perform complex grouping operations. You can use complex grouping operations to perform analysis that requires aggregation on multiple sets of columns in a single query.

The `grouping_expressions` element can be any function, such as `SUM`, `AVG`, or `COUNT`, performed on input columns.

`GROUP BY` expressions can group output by input column names that don't appear in the output of the `SELECT` statement.

All output expressions must be either aggregate functions or columns present in the `GROUP BY` clause.

You can use a single query to perform analysis that requires aggregating multiple column sets.

Athena supports complex aggregations using `GROUPING SETS`, `CUBE` and `ROLLUP`. `GROUP BY GROUPING SETS` specifies multiple lists of columns to group on. `GROUP BY CUBE` generates all possible grouping sets for a given set of columns. `GROUP BY ROLLUP` generates all possible subtotals for a given set of columns. Complex grouping operations do not support grouping on expressions composed of input columns. Only column names are allowed.

You can often use `UNION ALL` to achieve the same results as these `GROUP BY` operations, but queries that use `GROUP BY` have the advantage of reading the data one time, whereas `UNION ALL` reads the underlying data three times and may produce inconsistent results when the data source is subject to change.

[HAVING condition]

Used with aggregate functions and the `GROUP BY` clause. Controls which groups are selected, eliminating groups that don't satisfy `condition`. This filtering occurs after groups and aggregates are computed.

[{ UNION | INTERSECT | EXCEPT } [ALL | DISTINCT] union_query]]

`UNION`, `INTERSECT`, and `EXCEPT` combine the results of more than one `SELECT` statement into a single query. `ALL` or `DISTINCT` control the uniqueness of the rows included in the final result set.

`UNION` combines the rows resulting from the first query with the rows resulting from the second query. To eliminate duplicates, `UNION` builds a hash table, which consumes memory. For better performance, consider using `UNION ALL` if your query does not require the elimination of duplicates. Multiple `UNION` clauses are processed left to right unless you use parentheses to explicitly define the order of processing.

`INTERSECT` returns only the rows that are present in the results of both the first and the second queries.

`EXCEPT` returns the rows from the results of the first query, excluding the rows found by the second query.

`ALL` causes all rows to be included, even if the rows are identical.

`DISTINCT` causes only unique rows to be included in the combined result set.

[ORDER BY expression [ASC | DESC] [NULLS FIRST | NULLS LAST] [, ...]]

Sorts a result set by one or more output `expression`.

When the clause contains multiple expressions, the result set is sorted according to the first `expression`. Then the second `expression` is applied to rows that have matching values from the first expression, and so on.

Each `expression` may specify output columns from `SELECT` or an ordinal number for an output column by position, starting at one.

`ORDER BY` is evaluated as the last step after any `GROUP BY` or `HAVING` clause. `ASC` and `DESC` determine whether results are sorted in ascending or descending order. The default sorting order is ascending (`ASC`). The default null ordering is `NULLS LAST`, regardless of ascending or descending sort order.

[OFFSET count [ROW | ROWS]]

Use the `OFFSET` clause to discard a number of leading rows from the result set. If the `ORDER BY` clause is present, the `OFFSET` clause is evaluated over a sorted result set, and the set remains sorted after the skipped rows are discarded. If the query has no `ORDER BY` clause, it is arbitrary which rows are discarded. If the count specified by `OFFSET` equals or exceeds the size of the result set, the final result is empty.

LIMIT [count | ALL]

Restricts the number of rows in the result set to `count`. `LIMIT ALL` is the same as omitting the `LIMIT` clause. If the query has no `ORDER BY` clause, the results are arbitrary.

TABLESAMPLE [BERNOULLI | SYSTEM] (percentage)

Optional operator to select rows from a table based on a sampling method.

`BERNOULLI` selects each row to be in the table sample with a probability of `percentage`. All physical blocks of the table are scanned, and certain rows are skipped based on a comparison between the sample `percentage` and a random value calculated at runtime.

With `SYSTEM`, the table is divided into logical segments of data, and the table is sampled at this granularity.

Either all rows from a particular segment are selected, or the segment is skipped based on a comparison between the sample `percentage` and a random value calculated at runtime. `SYSTEM` sampling is dependent on the connector. This method does not guarantee independent sampling probabilities.

[UNNEST (array_or_map) [WITH ORDINALITY]]

Expands an array or map into a relation. Arrays are expanded into a single column. Maps are expanded into two columns (*key*, *value*).

You can use `UNNEST` with multiple arguments, which are expanded into multiple columns with as many rows as the highest cardinality argument.

Other columns are padded with nulls.

The `WITH ORDINALITY` clause adds an ordinality column to the end.

`UNNEST` is usually used with a `JOIN` and can reference columns from relations on the left side of the `JOIN`.

Getting the file locations for source data in Amazon S3

To see the Amazon S3 file location for the data in a table row, you can use `"$path"` in a `SELECT` query, as in the following example:

```
SELECT "$path" FROM "my_database"."my_table" WHERE year=2019;
```

This returns a result like the following:

```
s3://amzn-s3-demo-bucket/datasets_mytable/year=2019/data_file1.json
```

To return a sorted, unique list of the S3 filename paths for the data in a table, you can use `SELECT DISTINCT` and `ORDER BY`, as in the following example.

```
SELECT DISTINCT "$path" AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

To return only the filenames without the path, you can pass `"$path"` as a parameter to an `regexp_extract` function, as in the following example.

```
SELECT DISTINCT regexp_extract("$path", '[^/]+$') AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

To return the data from a specific file, specify the file in the `WHERE` clause, as in the following example.

```
SELECT *,"$path" FROM my_database.my_table WHERE "$path" = 's3://amzn-s3-demo-bucket/
my_table/my_partition/file-01.csv'
```

For more information and examples, see the Knowledge Center article [How can I see the Amazon S3 source file for a row in an Athena table?](#).

> **Note**
>
> In Athena, the Hive or Iceberg hidden metadata columns `$bucket`, `$file_modified_time`, `$file_size`, and `$partition` are not supported for views.

Escaping single quotes

To escape a single quote, precede it with another single quote, as in the following example. Do not confuse this with a double quote.

```
Select 'O''Reilly'
```

Results

`O'Reilly`

Additional resources

For more information about using `SELECT` statements in Athena, see the following resources.

For information about this	See this
Running queries in Athena	Run SQL queries in Amazon Athena
Using `SELECT` to create a table	Create a table from query results (CTAS)
Inserting data from a `SELECT` query into another table	INSERT INTO
Using built-in functions in `SELECT` statements	Functions in Amazon Athena
Using user defined functions in `SELECT` statements	Query with user defined functions
Querying Data Catalog metadata	Query the AWS Glue Data Catalog

INSERT INTO

Inserts new rows into a destination table based on a `SELECT` query statement that runs on a source table, or based on a set of `VALUES` provided as part of the statement. When the source table is based on underlying data in one format, such as CSV or JSON, and the destination table is based on another format, such as Parquet or ORC, you can use `INSERT INTO` queries to transform selected data into the destination table's format.

Considerations and limitations

Consider the following when using `INSERT` queries with Athena.

- When running an `INSERT` query on a table with underlying data that is encrypted in Amazon S3, the output files that the `INSERT` query writes are not encrypted by default. We recommend that you encrypt `INSERT` query results if you are inserting into tables with encrypted data.

  For more information about encrypting query results using the console, see Encrypt Athena query results stored in Amazon S3. To enable encryption using the AWS CLI or Athena API, use the `EncryptionConfiguration` properties of the StartQueryExecution action to specify Amazon S3 encryption options according to your requirements.
- For `INSERT INTO` statements, the expected bucket owner setting does not apply to the destination table location in Amazon S3. The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. For more information, see Specify a query result location using the Athena console.
- For ACID compliant `INSERT INTO` statements, see the `INSERT INTO` section of Update Iceberg table data.

Supported formats and SerDes

You can run an `INSERT` query on tables created from data with the following formats and SerDes.

Data format	SerDe
Avro	org.apache.hadoop.hive.serde2.avro.AvroSerDe
Ion	com.amazon.ionhiveserde.IonHiveSerDe
JSON	org.apache.hive.hcatalog.data.JsonSerDe
ORC	org.apache.hadoop.hive.ql.io.orc.OrcSerde
Parquet	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
Text file	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

Data format	SerDe
	Note TSV and custom-delimited files are supported.
CSV	org.apache.hadoop.hive.serde2.OpenCSVSerde **Note** Writes are only supported for string types. From Athena, you cannot write to any tables that contain non-string types in Glue schema. For more information, see CSV SerDe.

Bucketed tables not supported

`INSERT INTO` is not supported on bucketed tables. For more information, see Use partitioning and bucketing.

Federated queries not supported

`INSERT INTO` is not supported for federated queries. Attempting to do so may result in the error message This operation is currently not supported for external catalogs. For information about federated queries, see Use Amazon Athena Federated Query.

Partitioning

Consider the points in this section when using partitioning with `INSERT INTO` or `CREATE TABLE AS SELECT` queries.

Limits

The `INSERT INTO` statement supports writing a maximum of 100 partitions to the destination table. If you run the `SELECT` clause on a table with more than 100 partitions, the query fails unless the `SELECT` query is limited to 100 partitions or fewer.

For information about working around this limitation, see Use CTAS and INSERT INTO to work around the 100 partition limit.

Column ordering

`INSERT INTO` or `CREATE TABLE AS SELECT` statements expect the partitioned column to be the last column in the list of projected columns in a `SELECT` statement.

If the source table is non-partitioned, or partitioned on different columns compared to the destination table, queries like `INSERT INTO destination_table SELECT * FROM source_table` consider the values in the last column of the source table to be values for a partition column in the destination table. Keep this in mind when trying to create a partitioned table from a non-partitioned table.

Resources

For more information about using `INSERT INTO` with partitioning, see the following resources.

- For inserting partitioned data into a partitioned table, see Use CTAS and INSERT INTO to work around the 100 partition limit.
- For inserting unpartitioned data into a partitioned table, see Use CTAS and INSERT INTO for ETL and data analysis.

Files written to Amazon S3

Athena writes files to source data locations in Amazon S3 as a result of the `INSERT` command. Each `INSERT` operation creates a new file, rather than appending to an existing file. The file locations depend on the structure of the table and the `SELECT` query, if present. Athena generates a data manifest file for each `INSERT` query. The manifest tracks the files that the query wrote. It is saved to the Athena query result location in Amazon S3. For more information, see Identify query output files.

Avoid highly transactional updates

When you use `INSERT INTO` to add rows to a table in Amazon S3, Athena does not rewrite or modify existing files. Instead, it writes the rows as one or more new files. Because tables with many small files result in lower query performance, and write and read operations such as `PutObject` and `GetObject` result in higher costs from Amazon S3, consider the following options when using `INSERT INTO`:

- Run `INSERT INTO` operations less frequently on larger batches of rows.
- For large data ingestion volumes, consider using a service like Amazon Data Firehose.

- Avoid using INSERT INTO altogether. Instead, accumulate rows into larger files and upload them directly to Amazon S3 where they can be queried by Athena.

Locating orphaned files

If a CTAS or INSERT INTO statement fails, orphaned data can be left in the data location and might be read in subsequent queries. To locate orphaned files for inspection or deletion, you can use the data manifest file that Athena provides to track the list of files to be written. For more information, see Identify query output files and DataManifestLocation.

INSERT INTO...SELECT

Specifies the query to run on one table, source_table, which determines rows to insert into a second table, destination_table. If the SELECT query specifies columns in the source_table, the columns must precisely match those in the destination_table.

For more information about SELECT queries, see SELECT.

Synopsis

```
INSERT INTO destination_table
SELECT select_query
FROM source_table_or_view
```

Examples

Select all rows in the vancouver_pageviews table and insert them into the canada_pageviews table:

```
INSERT INTO canada_pageviews
SELECT *
FROM vancouver_pageviews;
```

Select only those rows in the vancouver_pageviews table where the date column has a value between 2019-07-01 and 2019-07-31, and then insert them into canada_july_pageviews:

```
INSERT INTO canada_july_pageviews
SELECT *
FROM vancouver_pageviews
WHERE date
```

```
    BETWEEN date '2019-07-01'
        AND '2019-07-31';
```

Select the values in the `city` and `state` columns in the `cities_world` table only from those rows with a value of `usa` in the `country` column and insert them into the `city` and `state` columns in the `cities_usa` table:

```
INSERT INTO cities_usa (city,state)
SELECT city,state
FROM cities_world
    WHERE country='usa'
```

INSERT INTO...VALUES

Inserts rows into an existing table by specifying columns and values. Specified columns and associated data types must precisely match the columns and data types in the destination table.

> **Important**
>
> We do not recommend inserting rows using `VALUES` because Athena generates files for each `INSERT` operation. This can cause many small files to be created and degrade the table's query performance. To identify files that an `INSERT` query creates, examine the data manifest file. For more information, see Work with query results and recent queries.

Synopsis

```
INSERT INTO destination_table [(col1,col2,...)]
VALUES (col1value,col2value,...)[,
       (col1value,col2value,...)][,
       ...]
```

Examples

In the following examples, the cities table has three columns: `id`, `city`, `state`, `state_motto`. The `id` column is type `INT` and all other columns are type `VARCHAR`.

Insert a single row into the `cities` table, with all column values specified:

```
INSERT INTO cities
```

```
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice')
```

Insert two rows into the `cities` table:

```
INSERT INTO cities
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice'),
       (3,'Boise','ID','Esto perpetua')
```

VALUES

Creates a literal inline table. The table can be anonymous, or you can use the `AS` clause to specify a table name, column names, or both.

Synopsis

```
VALUES row [, ...]
```

Parameters

row

The `row` parameter can be a single expression or `( column_expression [, ...] )`.

Examples

Return a table with one column and three rows:

```
VALUES 1, 2, 3
```

Return a table with two columns and three rows:

```
VALUES
    (1, 'a'),
    (2, 'b'),
    (3, 'c')
```

Return a table with the columns `id` and `name`:

```
SELECT * FROM (
    VALUES
```

```
        (1, 'a'),
        (2, 'b'),
        (3, 'c')
) AS t (id, name)
```

Create a table called `customers` with the columns `id` and `name`:

```
CREATE TABLE customers AS
SELECT * FROM (
    VALUES
        (1, 'a'),
        (2, 'b'),
        (3, 'c')
) AS t (id, name)
```

See also

INSERT INTO...VALUES

DELETE

Deletes rows in an Apache Iceberg table. `DELETE` is transactional and is supported only for Apache Iceberg tables.

Synopsis

To delete the rows from an Iceberg table, use the following syntax.

```
DELETE FROM [db_name.]table_name [WHERE predicate]
```

For more information and examples, see the `DELETE` section of Update Iceberg table data.

UPDATE

Updates rows in an Apache Iceberg table. `UPDATE` is transactional and is supported only for Apache Iceberg tables. The statement works only on existing rows and cannot be used to insert or append a row.

Synopsis

To update the rows in an Iceberg table, use the following syntax.

```
UPDATE [db_name.]table_name SET xx=yy[,...] [WHERE predicate]
```

For more information and examples, see the UPDATE section of [Update Iceberg table data](#).

MERGE INTO

Conditionally updates, deletes, or inserts rows into an Apache Iceberg table. A single statement can combine update, delete, and insert actions.

> **Note**
>
> `MERGE INTO` is transactional and is supported only for Apache Iceberg tables in Athena engine version 3.

Synopsis

To conditionally update, delete, or insert rows from an Iceberg table, use the following syntax.

```
MERGE INTO target_table [ [ AS ]  target_alias ]
USING { source_table | query } [ [ AS ] source_alias ]
ON search_condition
when_clause [...]
```

The *`when_clause`* is one of the following:

```
WHEN MATCHED [ AND condition ]
    THEN DELETE
```

```
WHEN MATCHED [ AND condition ]
    THEN UPDATE SET ( column = expression [, ...] )
```

```
WHEN NOT MATCHED [ AND condition ]
    THEN INSERT (column_name[, column_name ...]) VALUES (expression, ...)
```

`MERGE` supports an arbitrary number of `WHEN` clauses with different `MATCHED` conditions. The condition clauses execute the `DELETE`, `UPDATE` or `INSERT` operation in the first `WHEN` clause selected by the `MATCHED` state and the match condition.

For each source row, the WHEN clauses are processed in order. Only the first matching WHEN clause is executed. Subsequent clauses are ignored. A user error is raised when a single target table row matches more than one source row.

If a source row is not matched by any WHEN clause and there is no WHEN NOT MATCHED clause, the source row is ignored.

In WHEN clauses that have UPDATE operations, the column value expressions can refer to any field of the target or the source. In the NOT MATCHED case, the INSERT expressions can refer to any field of the source.

Example

The following example merges rows from the second table into the first table if the rows don't exist in the first table. Note that the columns listed in the VALUES clause must be prefixed by the source table alias. The target columns listed in the INSERT clause must *not* be so prefixed.

```
MERGE INTO iceberg_table_sample as ice1
USING iceberg2_table_sample as ice2
ON ice1.col1 = ice2.col1
WHEN NOT MATCHED
THEN INSERT (col1)
      VALUES (ice2.col1)
```

For more MERGE INTO examples, see [Update Iceberg table data].

OPTIMIZE

Optimizes rows in an Apache Iceberg table by rewriting data files into a more optimized layout based on their size and number of associated delete files.

> **Note**
> OPTIMIZE is transactional and is supported only for Apache Iceberg tables.

Syntax

The following syntax summary shows how to optimize data layout for an Iceberg table.

```
OPTIMIZE [db_name.]table_name REWRITE DATA USING BIN_PACK
  [WHERE predicate]
```

> **Note**
>
> Only partition columns are allowed in the WHERE clause *predicate*. Specifying a non-partition column will cause the query to fail.

The compaction action is charged by the amount of data scanned during the rewrite process. The REWRITE DATA action uses predicates to select for files that contain matching rows. If any row in the file matches the predicate, the file is selected for optimization. Thus, to control the number of files affected by the compaction operation, you can specify a WHERE clause.

Configuring compaction properties

To control the size of the files to be selected for compaction and the resulting file size after compaction, you can use table property parameters. You can use the ALTER TABLE SET TBLPROPERTIES command to configure the related table properties.

Additional resources

Optimize Iceberg tables

VACUUM

The VACUUM statement performs table maintenance on Apache Iceberg tables by performing snapshot expiration and orphan file removal.

> **Note**
>
> VACUUM is transactional and is supported only for Apache Iceberg tables in Athena engine version 3.

The VACUUM statement optimizes Iceberg tables by reducing storage consumption. For more information about using VACUUM, see Optimize Iceberg tables. Note that, because the VACUUM statement makes API calls to Amazon S3, charges apply for the associated requests to Amazon S3.

> **Warning**
>
> If you run a snapshot expiration operation, you can no longer time travel to expired snapshots.

Synopsis

To remove data files no longer needed for an Iceberg table, use the following syntax.

```
VACUUM [database_name.]target_table
```

- `VACUUM` expects the Iceberg data to be in an Amazon S3 folder rather than an Amazon S3 bucket. For example, if your Iceberg data is at `s3://amzn-s3-demo-bucket/` instead of `s3://amzn-s3-demo-bucket/myicebergfolder/`, the `VACUUM` statement fails with the error message GENERIC_INTERNAL_ERROR: Path missing in file system location: `s3://amzn-s3-demo-bucket`.
- For `VACUUM` to be able to delete data files, your query execution role must have `s3:DeleteObject` permissions on the bucket where your Iceberg tables, metadata, snapshots, and data files are located. If the permission is not present, the `VACUUM` query will succeed, but the files will not be deleted.
- To run `VACUUM` on a table with a name that begins with an underscore (for example, `_mytable`), enclose the table name in backticks, as in the following example. If you prefix the table name with a database name, do not enclose the database name in backticks. Note that double quotes will not work in place of backticks.

  This behavior is particular to `VACUUM`. The `CREATE` and `INSERT INTO` statements do not require backticks for table names that begin with underscores.

  ```
  VACUUM `_mytable`
  VACUUM my_database.`_mytable`
  ```

Operations performed

`VACUUM` performs the following operations:

- Removes snapshots that are older than the amount of time that is specified by the `vacuum_max_snapshot_age_seconds` table property. By default, this property is set to 432000 seconds (5 days).
- Removes snapshots that are not within the period to be retained that are in excess of the number specified by the `vacuum_min_snapshots_to_keep` table property. The default is 1.

  You can specify these table properties in your `CREATE TABLE` statement. After the table has been created, you can use the ALTER TABLE SET TBLPROPERTIES statement to update them.
- Removes any metadata and data files that are unreachable as a result of the snapshot removal. You can configure the number of old metadata files to be retained by setting the `vacuum_max_metadata_files_to_keep` table property. The default value is 100.
- Removes orphan files that are older than the time specified in the `vacuum_max_snapshot_age_seconds` table property. Orphan files are files in the table's data directory that are not part of the table state.

For more information about creating and managing Apache Iceberg tables in Athena, see Create Iceberg tables and Manage Iceberg tables.

Using EXPLAIN and EXPLAIN ANALYZE in Athena

The `EXPLAIN` statement shows the logical or distributed execution plan of a specified SQL statement, or validates the SQL statement. You can output the results in text format or in a data format for rendering into a graph.

> **Note**
>
> You can view graphical representations of logical and distributed plans for your queries in the Athena console without using the `EXPLAIN` syntax. For more information, see View execution plans for SQL queries.

The `EXPLAIN ANALYZE` statement shows both the distributed execution plan of a specified SQL statement and the computational cost of each operation in a SQL query. You can output the results in text or JSON format.

Considerations and limitations

The `EXPLAIN` and `EXPLAIN ANALYZE` statements in Athena have the following limitations.

- Because EXPLAIN queries do not scan any data, Athena does not charge for them. However, because EXPLAIN queries make calls to AWS Glue to retrieve table metadata, you may incur charges from Glue if the calls go above the free tier limit for glue.
- Because EXPLAIN ANALYZE queries are executed, they do scan data, and Athena charges for the amount of data scanned.
- Row or cell filtering information defined in Lake Formation and query stats information are not shown in the output of EXPLAIN and EXPLAIN ANALYZE.

EXPLAIN syntax

```
EXPLAIN [ ( option [, ...]) ] statement
```

option can be one of the following:

```
FORMAT { TEXT | GRAPHVIZ | JSON }
TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

If the FORMAT option is not specified, the output defaults to TEXT format. The IO type provides information about the tables and schemas that the query reads.

EXPLAIN ANALYZE syntax

In addition to the output included in EXPLAIN, EXPLAIN ANALYZE output also includes runtime statistics for the specified query such as CPU usage, the number of rows input, and the number of rows output.

```
EXPLAIN ANALYZE [ ( option [, ...]) ] statement
```

option can be one of the following:

```
FORMAT { TEXT | JSON }
```

If the FORMAT option is not specified, the output defaults to TEXT format. Because all queries for EXPLAIN ANALYZE are DISTRIBUTED, the TYPE option is not available for EXPLAIN ANALYZE.

statement can be one of the following:

```
SELECT
```

```
CREATE TABLE AS SELECT
INSERT
UNLOAD
```

EXPLAIN examples

The following examples for `EXPLAIN` progress from the more straightforward to the more complex.

Example 1: Use the EXPLAIN statement to show a query plan in text format

In the following example, `EXPLAIN` shows the execution plan for a `SELECT` query on Elastic Load Balancing logs. The format defaults to text output.

```
EXPLAIN
SELECT
   request_timestamp,
   elb_name,
   request_ip
FROM sampledb.elb_logs;
```

Results

```
- Output[request_timestamp, elb_name, request_ip] => [[request_timestamp, elb_name,
 request_ip]]
    - RemoteExchange[GATHER] => [[request_timestamp, elb_name, request_ip]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=sampledb,
 tableName=elb_logs,
analyzePartitionValues=Optional.empty}] => [[request_timestamp, elb_name, request_ip]]
                LAYOUT: sampledb.elb_logs
                request_ip := request_ip:string:2:REGULAR
                request_timestamp := request_timestamp:string:0:REGULAR
                elb_name := elb_name:string:1:REGULAR
```

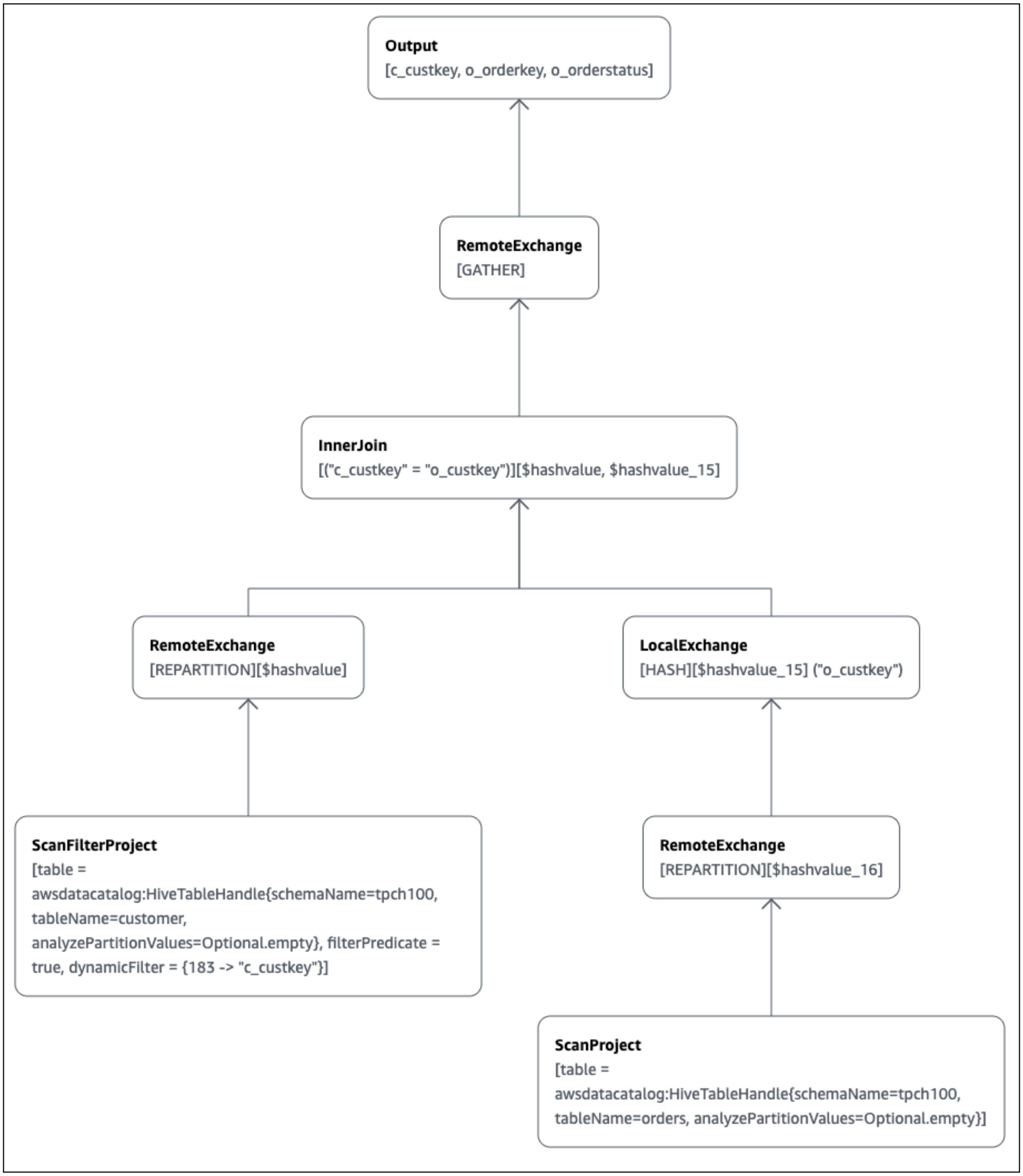
Example 2: Use EXPLAIN to graph a query plan

You can use the Athena console to graph a query plan for you. Enter a `SELECT` statement like the following into the Athena query editor, and then choose **EXPLAIN**.

```
SELECT
      c.c_custkey,
```

```
    o.o_orderkey,
    o.o_orderstatus
 FROM tpch100.customer c
 JOIN tpch100.orders o
     ON c.c_custkey = o.o_custkey
```

The **Explain** page of the Athena query editor opens and shows you a distributed plan and a logical plan for the query. The following graph shows the logical plan for the example.



Important

Currently, some partition filters may not be visible in the nested operator tree graph even though Athena does apply them to your query. To verify the effect of such filters, run `EXPLAIN` or `EXPLAIN ANALYZE` on your query and view the results.

For more information about using the query plan graphing features in the Athena console, see [View execution plans for SQL queries](#).

Example 3: Use the EXPLAIN statement to verify partition pruning

When you use a filtering predicate on a partitioned key to query a partitioned table, the query engine applies the predicate to the partitioned key to reduce the amount of data read.

The following example uses an `EXPLAIN` query to verify partition pruning for a `SELECT` query on a partitioned table. First, a `CREATE TABLE` statement creates the `tpch100.orders_partitioned` table. The table is partitioned on column `o_orderdate`.

```
CREATE TABLE `tpch100.orders_partitioned`(
  `o_orderkey` int,
  `o_custkey` int,
  `o_orderstatus` string,
  `o_totalprice` double,
  `o_orderpriority` string,
  `o_clerk` string,
  `o_shippriority` int,
  `o_comment` string)
PARTITIONED BY (
  `o_orderdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://amzn-s3-demo-bucket/<your_directory_path>/'
```

The `tpch100.orders_partitioned` table has several partitions on `o_orderdate`, as shown by the `SHOW PARTITIONS` command.

```
SHOW PARTITIONS tpch100.orders_partitioned;

o_orderdate=1994
o_orderdate=2015
o_orderdate=1998
o_orderdate=1995
o_orderdate=1993
o_orderdate=1997
o_orderdate=1992
o_orderdate=1996
```

The following `EXPLAIN` query verifies partition pruning on the specified `SELECT` statement.

```
EXPLAIN
SELECT
   o_orderkey,
   o_custkey,
   o_orderdate
FROM tpch100.orders_partitioned
WHERE o_orderdate = '1995'
```

Results

```
Query Plan
- Output[o_orderkey, o_custkey, o_orderdate] => [[o_orderkey, o_custkey, o_orderdate]]
    - RemoteExchange[GATHER] => [[o_orderkey, o_custkey, o_orderdate]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=tpch100,
 tableName=orders_partitioned,
analyzePartitionValues=Optional.empty}] => [[o_orderkey, o_custkey, o_orderdate]]
                LAYOUT: tpch100.orders_partitioned
                o_orderdate := o_orderdate:string:-1:PARTITION_KEY
                    :: [[1995]]
                o_custkey := o_custkey:int:1:REGULAR
                o_orderkey := o_orderkey:int:0:REGULAR
```

The bold text in the result shows that the predicate `o_orderdate = '1995'` was applied on the `PARTITION_KEY`.

Example 4: Use an EXPLAIN query to check the join order and join type

The following EXPLAIN query checks the SELECT statement's join order and join type. Use a query like this to examine query memory usage so that you can reduce the chances of getting an EXCEEDED_LOCAL_MEMORY_LIMIT error.

```
EXPLAIN (TYPE DISTRIBUTED)
   SELECT
      c.c_custkey,
      o.o_orderkey,
      o.o_orderstatus
   FROM tpch100.customer c
   JOIN tpch100.orders o
       ON c.c_custkey = o.o_custkey
   WHERE c.c_custkey = 123
```

Results

```
Query Plan
Fragment 0 [SINGLE]
    Output layout: [c_custkey, o_orderkey, o_orderstatus]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - Output[c_custkey, o_orderkey, o_orderstatus] => [[c_custkey, o_orderkey,
 o_orderstatus]]
        - RemoteSource[1] => [[c_custkey, o_orderstatus, o_orderkey]]

Fragment 1 [SOURCE]
    Output layout: [c_custkey, o_orderstatus, o_orderkey]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - CrossJoin => [[c_custkey, o_orderstatus, o_orderkey]]
            Distribution: REPLICATED
        - ScanFilter[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("c_custkey" = 123)] => [[c_custkey]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
        - LocalExchange[SINGLE] () => [[o_orderstatus, o_orderkey]]
            - RemoteSource[2] => [[o_orderstatus, o_orderkey]]

Fragment 2 [SOURCE]
    Output layout: [o_orderstatus, o_orderkey]
```

```
    Output partitioning: BROADCAST []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - ScanFilterProject[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=orders, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("o_custkey" = 123)] => [[o_orderstatus, o_orderkey]]
            LAYOUT: tpch100.orders
            o_orderstatus := o_orderstatus:string:2:REGULAR
            o_custkey := o_custkey:int:1:REGULAR
            o_orderkey := o_orderkey:int:0:REGULAR
```

The example query was optimized into a cross join for better performance. The results show that `tpch100.orders` will be distributed as the `BROADCAST` distribution type. This implies that the `tpch100.orders` table will be distributed to all nodes that perform the join operation. The `BROADCAST` distribution type will require that the all of the filtered results of the `tpch100.orders` table fit into the memory of each node that performs the join operation.

However, the `tpch100.customer` table is smaller than `tpch100.orders`. Because `tpch100.customer` requires less memory, you can rewrite the query to `BROADCAST` `tpch100.customer` instead of `tpch100.orders`. This reduces the chance of the query receiving the `EXCEEDED_LOCAL_MEMORY_LIMIT` error. This strategy assumes the following points:

- The `tpch100.customer.c_custkey` is unique in the `tpch100.customer` table.
- There is a one-to-many mapping relationship between `tpch100.customer` and `tpch100.orders`.

The following example shows the rewritten query.

```
SELECT
    c.c_custkey,
    o.o_orderkey,
    o.o_orderstatus
FROM tpch100.orders o
JOIN tpch100.customer c -- the filtered results of tpch100.customer are distributed to
 all nodes.
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey = 123
```

Example 5: Use an EXPLAIN query to remove predicates that have no effect

You can use an EXPLAIN query to check the effectiveness of filtering predicates. You can use the results to remove predicates that have no effect, as in the following example.

```
EXPLAIN
   SELECT
      c.c_name
   FROM tpch100.customer c
   WHERE c.c_custkey = CAST(RANDOM() * 1000 AS INT)
   AND c.c_custkey BETWEEN 1000 AND 2000
   AND c.c_custkey = 1500
```

Results

```
Query Plan
- Output[c_name] => [[c_name]]
    - RemoteExchange[GATHER] => [[c_name]]
        - ScanFilterProject[table =
awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty},
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" =
CAST(("random"() * 1E3) AS int)))] => [[c_name]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
                c_name := c_name:string:1:REGULAR
```

The filterPredicate in the results shows that the optimizer merged the original three predicates into two predicates and changed their order of application.

```
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" = CAST(("random"() * 1E3) AS
 int)))
```

Because the results show that the predicate AND c.c_custkey BETWEEN 1000 AND 2000 has no effect, you can remove this predicate without changing the query results.

For information about the terms used in the results of EXPLAIN queries, see Understand Athena EXPLAIN statement results.

EXPLAIN ANALYZE examples

The following examples show example EXPLAIN ANALYZE queries and outputs.

Example 1: Use EXPLAIN ANALYZE to show a query plan and computational cost in text format

In the following example, EXPLAIN ANALYZE shows the execution plan and computational costs for a SELECT query on CloudFront logs. The format defaults to text output.

```
EXPLAIN ANALYZE SELECT FROM cloudfront_logs LIMIT 10
```

Results

```
 Fragment 1
     CPU: 24.60ms, Input: 10 rows (1.48kB); per task: std.dev.: 0.00, Output: 10 rows
 (1.48kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer,\
       os, browser, browserversion]
Limit[10] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
             CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
             Input avg.: 10.00 rows, Input std.dev.: 0.00%
LocalExchange[SINGLE] () => [[date, time, location, bytes, requestip, method, host,
 uri, status, referrer, os,\
 browser, browserversion]]
                 CPU: 0.00ns (0.00%), Output: 10 rows (1.48kB)
                 Input avg.: 0.63 rows, Input std.dev.: 387.30%
RemoteSource[2] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
                     CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
                     Input avg.: 0.63 rows, Input std.dev.: 387.30%

 Fragment 2
     CPU: 3.83s, Input: 998 rows (147.21kB); per task: std.dev.: 0.00, Output: 20 rows
 (2.95kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
       browser, browserversion]
LimitPartial[10] => [[date, time, location, bytes, requestip, method, host, uri,
 status, referrer, os,\
  browser, browserversion]]
             CPU: 5.00ms (0.13%), Output: 20 rows (2.95kB)
             Input avg.: 166.33 rows, Input std.dev.: 141.42%
```

```
TableScan[awsdatacatalog:HiveTableHandle{schemaName=default, tableName=cloudfront_logs,
\
  analyzePartitionValues=Optional.empty},
grouped = false] => [[date, time, location, bytes, requestip, method, host, uri, st
                 CPU: 3.82s (99.82%), Output: 998 rows (147.21kB)
                 Input avg.: 166.33 rows, Input std.dev.: 141.42%
                 LAYOUT: default.cloudfront_logs
                 date := date:date:0:REGULAR
                 referrer := referrer:string:9:REGULAR
                 os := os:string:10:REGULAR
                 method := method:string:5:REGULAR
                 bytes := bytes:int:3:REGULAR
                 browser := browser:string:11:REGULAR
                 host := host:string:6:REGULAR
                 requestip := requestip:string:4:REGULAR
                 location := location:string:2:REGULAR
                 time := time:string:1:REGULAR
                 uri := uri:string:7:REGULAR
                 browserversion := browserversion:string:12:REGULAR
                 status := status:int:8:REGULAR
```

Example 2: Use EXPLAIN ANALYZE to show a query plan in JSON format

The following example shows the execution plan and computational costs for a SELECT query on CloudFront logs. The example specifies JSON as the output format.

```
EXPLAIN ANALYZE (FORMAT JSON) SELECT * FROM cloudfront_logs LIMIT 10
```

Results

```
{
    "fragments": [{
        "id": "1",

        "stageStats": {
            "totalCpuTime": "3.31ms",
            "inputRows": "10 rows",
            "inputDataSize": "1514B",
            "stdDevInputRows": "0.00",
            "outputRows": "10 rows",
            "outputDataSize": "1514B"
        },
        "outputLayout": "date, time, location, bytes, requestip, method, host,\
```

```
            uri, status, referrer, os, browser, browserversion",

         "logicalPlan": {
             "1": [{
                 "name": "Limit",
                 "identifier": "[10]",
                 "outputs": ["date", "time", "location", "bytes", "requestip", "method",
 "host",\
                   "uri", "status", "referrer", "os", "browser", "browserversion"],
                 "details": "",
                 "distributedNodeStats": {
                     "nodeCpuTime": "0.00ns",
                     "nodeOutputRows": 10,
                     "nodeOutputDataSize": "1514B",
                     "operatorInputRowsStats": [{
                         "nodeInputRows": 10.0,
                         "nodeInputRowsStdDev": 0.0
                     }]
                 },
                 "children": [{
                     "name": "LocalExchange",
                     "identifier": "[SINGLE] ()",
                     "outputs": ["date", "time", "location", "bytes", "requestip",
 "method", "host",\
                       "uri", "status", "referrer", "os", "browser", "browserversion"],
                     "details": "",
                     "distributedNodeStats": {
                         "nodeCpuTime": "0.00ns",
                         "nodeOutputRows": 10,
                         "nodeOutputDataSize": "1514B",
                         "operatorInputRowsStats": [{
                             "nodeInputRows": 0.625,
                             "nodeInputRowsStdDev": 387.2983346207417
                         }]
                     },
                     "children": [{
                         "name": "RemoteSource",
                         "identifier": "[2]",
                         "outputs": ["date", "time", "location", "bytes", "requestip",
 "method", "host",\
                           "uri", "status", "referrer", "os", "browser",
 "browserversion"],
                         "details": "",
                         "distributedNodeStats": {
```

```
                            "nodeCpuTime": "0.00ns",
                            "nodeOutputRows": 10,
                            "nodeOutputDataSize": "1514B",
                            "operatorInputRowsStats": [{
                                "nodeInputRows": 0.625,
                                "nodeInputRowsStdDev": 387.2983346207417
                            }]
                        },
                        "children": []
                    }]
                }]
            }]
        }
    }, {
        "id": "2",

        "stageStats": {
            "totalCpuTime": "1.62s",
            "inputRows": "500 rows",
            "inputDataSize": "75564B",
            "stdDevInputRows": "0.00",
            "outputRows": "10 rows",
            "outputDataSize": "1514B"
        },
        "outputLayout": "date, time, location, bytes, requestip, method, host, uri, 
status,\
          referrer, os, browser, browserversion",

        "logicalPlan": {
            "1": [{
                "name": "LimitPartial",
                "identifier": "[10]",
                "outputs": ["date", "time", "location", "bytes", "requestip", "method", 
"host", "uri",\
                  "status", "referrer", "os", "browser", "browserversion"],
                "details": "",
                "distributedNodeStats": {
                    "nodeCpuTime": "0.00ns",
                    "nodeOutputRows": 10,
                    "nodeOutputDataSize": "1514B",
                    "operatorInputRowsStats": [{
                        "nodeInputRows": 83.33333333333333,
                        "nodeInputRowsStdDev": 223.60679774997897
                    }]
```

```
                },
                "children": [{
                    "name": "TableScan",
                    "identifier": "[awsdatacatalog:HiveTableHandle{schemaName=default,\
                       tableName=cloudfront_logs,
 analyzePartitionValues=Optional.empty},\
                       grouped = false]",
                    "outputs": ["date", "time", "location", "bytes", "requestip",
 "method", "host", "uri",\
                       "status", "referrer", "os", "browser", "browserversion"],
                    "details": "LAYOUT: default.cloudfront_logs\ndate :=
 date:date:0:REGULAR\nreferrer :=\
                       referrer: string:9:REGULAR\nos := os:string:10:REGULAR
\nmethod := method:string:5:\
                       REGULAR\nbytes := bytes:int:3:REGULAR\nbrowser :=
 browser:string:11:REGULAR\nhost :=\
                       host:string:6:REGULAR\nrequestip := requestip:string:4:REGULAR
\nlocation :=\
                       location:string:2:REGULAR\ntime := time:string:1: REGULAR
\nuri := uri:string:7:\
                       REGULAR\nbrowserversion := browserversion:string:12:REGULAR
\nstatus :=\
                       status:int:8:REGULAR\n",
                    "distributedNodeStats": {
                        "nodeCpuTime": "1.62s",
                        "nodeOutputRows": 500,
                        "nodeOutputDataSize": "75564B",
                        "operatorInputRowsStats": [{
                            "nodeInputRows": 83.33333333333333,
                            "nodeInputRowsStdDev": 223.60679774997897
                        }]
                    },
                    "children": []
                }]
            }]
        }
    }]
}
```

Additional resources

For additional information, see the following resources.

- Understand Athena EXPLAIN statement results

- View execution plans for SQL queries
- View statistics and execution details for completed queries
- Trino `EXPLAIN` documentation
- Trino `EXPLAIN ANALYZE` documentation
- Optimize Federated Query Performance using EXPLAIN and EXPLAIN ANALYZE in Amazon Athena in the *AWS Big Data Blog*.

Understand Athena EXPLAIN statement results

This topic provides a brief guide to the operational terms used in Athena `EXPLAIN` statement results.

EXPLAIN statement output types

`EXPLAIN` statement outputs can be one of two types:

- **Logical plan** – Shows the logical plan that the SQL engine uses to execute a statement. The syntax for this option is `EXPLAIN` or `EXPLAIN (TYPE LOGICAL)`.
- **Distributed plan** – Shows an execution plan in a distributed environment. The output shows fragments, which are processing stages. Each plan fragment is processed by one or more nodes. Data can be exchanged between the nodes that process the fragments. The syntax for this option is `EXPLAIN (TYPE DISTRIBUTED)`.

  In the output for a distributed plan, fragments (processing stages) are indicated by `Fragment` *`number`* [*`fragment_type`*], where *`number`* is a zero-based integer and *`fragment_type`* specifies how the fragment is executed by the nodes. Fragment types, which provide insight into the layout of the data exchange, are described in the following table.

  Distributed plan fragment types

Fragment type	Description
`SINGLE`	The fragment is executed on a single node.
`HASH`	The fragment is executed on a fixed number of nodes. The input data is distributed using a hash function.

Fragment type	Description
`ROUND_ROBIN`	The fragment is executed on a fixed number of nodes. The input data is distributed in a round-robin fashion.
`BROADCAST`	The fragment is executed on a fixed number of nodes. The input data is broadcast to all nodes.
`SOURCE`	The fragment is executed on nodes where input splits are accessed.

Exchange

Exchange-related terms describe how data is exchanged between worker nodes. Transfers can be either local or remote.

LocalExchange [*exchange_type*]

Transfers data locally within worker nodes for different stages of a query. The value for *exchange_type* can be one of the logical or distributed exchange types as described later in this section.

RemoteExchange [*exchange_type*]

Transfers data between worker nodes for different stages of a query. The value for *exchange_type* can be one of the logical or distributed exchange types as described later in this section.

Logical Exchange types

The following exchange types describe actions taken during the exchange phase of a logical plan.

- **`GATHER`** – A single worker node gathers output from all other worker nodes. For example, the last stage of a select query gathers results from all nodes and writes the results to Amazon S3.
- **`REPARTITION`** – Sends the row data to a specific worker based on the partitioning scheme required to apply to the next operator.
- **`REPLICATE`** – Copies the row data to all workers.

Distributed Exchange types

The following exchange types indicate the layout of the data when they are exchanged between nodes in a distributed plan.

- **`HASH`** – The exchange distributes data to multiple destinations using a hash function.
- **`SINGLE`** – The exchange distributes data to a single destination.

Scanning

The following terms describe how data is scanned during a query.

TableScan

Scans a table's source data from Amazon S3 or an Apache Hive connector and applies partition pruning generated from the filter predicate.

ScanFilter

Scans a table's source data from Amazon S3 or a Hive connector and applies partition pruning generated from the filter predicate and from additional filter predicates not applied through partition pruning.

ScanFilterProject

First, scans a table's source data from Amazon S3 or a Hive connector and applies partition pruning generated from the filter predicate and from additional filter predicates not applied through partition pruning. Then, modifies the memory layout of the output data into a new projection to improve performance of later stages.

Join

Joins data between two tables. Joins can be categorized by join type and by distribution type.

Join types

Join types define the way in which the join operation occurs.

CrossJoin – Produces the Cartesian product of the two tables joined.

InnerJoin – Selects records that have matching values in both tables.

LeftJoin – Selects all records from the left table and the matching records from the right table. If no match occurs, the result on the right side is NULL.

RightJoin – Selects all records from the right table, and the matching records from the left table. If no match occurs, the result on the left side is NULL.

FullJoin – Selects all records where there is a match in the left or right table records. The joined table contains all records from both the tables and fills in NULLs for missing matches on either side.

> **Note**
>
> For performance reasons, the query engine can rewrite a join query into a different join type to produce the same results. For example, an inner join query with predicate on one table can be rewritten into a `CrossJoin`. This pushes the predicate down to the scanning phase of the table so that fewer data are scanned.

Join distribution types

Distribution types define how data is exchanged between worker nodes when the join operation is performed.

Partitioned – Both the left and right table are hash-partitioned across all worker nodes. Partitioned distribution consumes less memory in each node. Partitioned distribution can be much slower than replicated joins. Partitioned joins are suitable when you join two large tables.

Replicated – One table is hash-partitioned across all worker nodes and the other table is replicated to all worker nodes to perform the join operation. Replicated distribution can be much faster than partitioned joins, but it consumes more memory in each worker node. If the replicated table is too large, the worker node can experience an out-of-memory error. Replicated joins are suitable when one of the joined tables is small.

PREPARE

Creates a SQL statement with the name `statement_name` to be run at a later time. The statement can include parameters represented by question marks. To supply values for the parameters and run the prepared statement, use EXECUTE.

Synopsis

```
PREPARE statement_name FROM statement
```

The following table describes the parameters.

Parameter	Description
`statement_name`	The name of the statement to be prepared. The name must be unique within the workgroup.
`statement`	A `SELECT`, `CTAS`, or `INSERT INTO` query.

Note

The maximum number of prepared statements in a workgroup is 1000.

Examples

The following example prepares a select query without parameters.

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

The following example prepares a select query that includes parameters. The values for `productid` and `quantity` will be supplied by the `USING` clause of an `EXECUTE` statement:

```
PREPARE my_select2 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

The following example prepares an insert query.

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

Additional resources

[Use prepared statements]

[EXECUTE]

[DEALLOCATE PREPARE]

[INSERT INTO]

EXECUTE

Runs a prepared statement with the name `statement_name`. Parameter values for the question marks in the prepared statement are defined in the `USING` clause in a comma separated list. To create a prepared statement, use [PREPARE].

Synopsis

```
EXECUTE statement_name [ USING parameter1[, parameter2, ... ] ]
```

Examples

The following example prepares and executes a query with no parameters.

```
PREPARE my_select1 FROM
SELECT name FROM nation
EXECUTE my_select1
```

The following example prepares and executes a query with a single parameter.

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
EXECUTE my_select2 USING 2012
```

This is equivalent to:

```
SELECT * FROM "my_database"."my_table" WHERE year = 2012
```

The following example prepares and executes a query with two parameters.

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

```
EXECUTE my_select3 USING 346078, 12
```

Additional resources

Use prepared statements

PREPARE

INSERT INTO

DEALLOCATE PREPARE

Removes the prepared statement with the specified name from the prepared statements in the current workgroup.

Synopsis

```
DEALLOCATE PREPARE statement_name
```

Examples

The following example removes the `my_select1` prepared statement from the current workgroup.

```
DEALLOCATE PREPARE my_select1
```

Additional resources

Use prepared statements

PREPARE

UNLOAD

Writes query results from a `SELECT` statement to the specified data format. Supported formats for `UNLOAD` include Apache Parquet, ORC, Apache Avro, and JSON. CSV is the only output format supported by the Athena `SELECT` command, but you can use the `UNLOAD` command, which supports a variety of output formats, to enclose your `SELECT` query and rewrite its output to one of the formats that `UNLOAD` supports.

Although you can use the `CREATE TABLE AS` (CTAS) statement to output data in formats other than CSV, CTAS statements require the creation of a table in Athena. The `UNLOAD` statement is useful when you want to output the results of a `SELECT` query in a non-CSV format but do not want the associated table. For example, a downstream application might require the results of a SELECT query to be in JSON format, and Parquet or ORC might provide a performance advantage over CSV if you intend to use the results of the SELECT query for additional analysis.

Considerations and limitations

When you use the UNLOAD statement in Athena, keep in mind the following points:

- **No global ordering of files** – UNLOAD results are written to multiple files in parallel. If the SELECT query in the UNLOAD statement specifies a sort order, each file's contents are in sorted order, but the files are not sorted relative to each other.
- **Orphaned data not deleted** – In the case of a failure, Athena does not attempt to delete orphaned data. This behavior is the same as that for CTAS and INSERT INTO statements.
- **Maximum partitions** – The maximum number of partitions that can be used with UNLOAD is 100.
- **Metadata and manifest files** – Athena generates a metadata file and data manifest file for each UNLOAD query. The manifest tracks the files that the query wrote. Both files are saved to your Athena query result location in Amazon S3. For more information, see Identify query output files.
- **Encryption** – UNLOAD output files are encrypted according to the encryption configuration used for Amazon S3. To set up encryption configuration to encrypt your UNLOAD result, you can use the EncryptionConfiguration API.
- **Prepared statements** – UNLOAD can be used with prepared statements. For information about prepared statements in Athena, see Use parameterized queries.
- **Service quotas** – UNLOAD uses DML query quotas. For quota information, see Service Quotas.
- **Expected bucket owner** – The expected bucket owner setting does not apply to the destination Amazon S3 location specfied in the UNLOAD query. The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. For more information, see Specify a query result location using the Athena console.

Syntax

The UNLOAD statement uses the following syntax.

```
UNLOAD (SELECT col_name[, ...] FROM old_table)
TO 's3://amzn-s3-demo-bucket/my_folder/'
WITH ( property_name = 'expression' [, ...] )
```

Except when writing to partitions, the `TO` destination must specify a location in Amazon S3 that has no data. Before the `UNLOAD` query writes to the location specified, it verifies that the bucket location is empty. Because `UNLOAD` does not write data to the specified location if the location already has data in it, `UNLOAD` does not overwrite existing data. To reuse a bucket location as a destination for `UNLOAD`, delete the data in the bucket location, and then run the query again.

Note that when `UNLOAD` writes to partitions, this behavior is different. If you run the same `UNLOAD` query multiple times that has the same `SELECT` statement, the same `TO` location and the same partitions, each `UNLOAD` query unloads the data into Amazon S3 at the location and partitions specified.

Parameters

Possible values for *`property_name`* are as follows.

format = '*`file_format`*'

Required. Specifies the file format of the output. Possible values for *`file_format`* are `ORC`, `PARQUET`, `AVRO`, `JSON`, or `TEXTFILE`.

compression = '*`compression_format`*'

Optional. This option is specific to the ORC and Parquet formats. For ORC, the default is `zlib`, and for Parquet, the default is `gzip`. For information about supported compression formats, see Athena compression support.

> **Note**
>
> This option does not apply to the `AVRO` format. Athena uses `gzip` for the `JSON` and `TEXTFILE` formats.

compression_level = *`compression_level`*

Optional. The compression level to use for ZSTD compression. This property applies only to ZSTD compression. For more information, see Use ZSTD compression levels.

field_delimiter = '*`delimiter`*'

Optional. Specifies a single-character field delimiter for files in CSV, TSV, and other text formats. The following example specifies a comma delimiter.

```
WITH (field_delimiter = ',')
```

Currently, multicharacter field delimiters are not supported. If you do not specify a field delimiter, the octal character `\001` (^A) is used.

partitioned_by = ARRAY[*col_name*[,...]]

Optional. An array list of columns by which the output is partitioned.

> **Note**
>
> In your `SELECT` statement, make sure that the names of the partitioned columns are last in your list of columns.

Examples

The following example writes the output of a `SELECT` query to the Amazon S3 location `s3://amzn-s3-demo-bucket/unload_test_1/` using JSON format.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://amzn-s3-demo-bucket/unload_test_1/'
WITH (format = 'JSON')
```

The following example writes the output of a `SELECT` query in Parquet format using Snappy compression.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://amzn-s3-demo-bucket/'
WITH (format = 'PARQUET',compression = 'SNAPPY')
```

The following example writes four columns in text format, with the output partitioned by the last column.

```
UNLOAD (SELECT name1, address1, comment1, key1 FROM table1)
TO 's3://amzn-s3-demo-bucket/ partitioned/'
WITH (format = 'TEXTFILE', partitioned_by = ARRAY['key1'])
```

The following example unloads the query results to the specified location using the Parquet file format, ZSTD compression, and ZSTD compression level 4.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://amzn-s3-demo-bucket/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

Additional resources

- Simplify your ETL and ML pipelines using the Amazon Athena UNLOAD feature in the *AWS Big Data Blog*.

Functions in Amazon Athena

For changes in functions between Athena engine versions, see Athena engine versioning. For a list of the time zones that can be used with the `AT TIME ZONE` operator, see Use supported time zones.

Topics

- Athena engine version 3 functions

Athena engine version 3 functions

Functions in Athena engine version 3 are based on Trino. For information about Trino functions, operators, and expressions, see Functions and operators and the following subsections from the Trino documentation.

- Aggregate
- Array
- Binary
- Bitwise
- Color
- Comparison
- Conditional
- Conversion
- Date and time
- Decimal
- Geospatial

- HyperLogLog
- IP Address
- JSON
- Lambda
- Logical
- Machine learning
- Map
- Math
- Quantile digest
- Regular expression
- Session
- Set Digest
- String
- Table
- Teradata
- T-Digest
- URL
- UUID
- Window

invoker_principal() function

The `invoker_principal` function is unique to Athena engine version 3 and is not found in Trino.

Returns a `VARCHAR` that contains the ARN of the principal (IAM role or Identity Center identity) that ran the query calling the function. For example, if the query invoker uses the permissions of an IAM role to run the query, the function returns the ARN of the IAM role. The role that runs the query must allow the `LakeFormation:GetDataLakePrincipal` action.

Usage

```
SELECT invoker_principal()
```

The following table shows an example result.

#	_col0
1	arn:aws:iam::*111122223333* :role/Admin

Use supported time zones

You can use the `AT TIME ZONE` operator in a `SELECT timestamp` statement to specify the timezone for the timestamp that is returned, as in the following example:

```
SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'America/Los_Angeles' AS la_time;
```

Results

```
la_time

2012-10-30 18:00:00.000 America/Los_Angeles
```

For a list of supported time zones in Athena, expand the List of supported time zones at the end of this topic.

Timezone functions and examples

Following are some additional timezone related functions and examples.

- **at_timezone(*timestamp*, *zone*)** – Returns the value of *timestamp* in the corresponding local time for *zone*.

  Example

  ```
  SELECT at_timezone(timestamp '2021-08-22 00:00 UTC', 'Canada/Newfoundland')
  ```

  Result

  ```
  2021-08-21 21:30:00.000 Canada/Newfoundland
  ```

- **timezone_hour(*timestamp*)** – Returns the hour of the time zone offset from timestamp as a `bigint`.

  Example

```
SELECT timezone_hour(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
Newfoundland')
```

Result

```
-2
```

- **timezone_minute(*timestamp*)** – Returns the minute of the time zone offset from *timestamp* as a `bigint`.

  Example

  ```
  SELECT timezone_minute(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
  Newfoundland')
  ```

  Result

  ```
  -30
  ```

- **with_timezone(*timestamp*, *zone*)** – Returns a timestamp with time zone from the specified *timestamp* and *zone* values.

  Example

  ```
  SELECT with_timezone(timestamp '2021-08-22 04:00', 'Canada/Newfoundland')
  ```

  Result

  ```
  2021-08-22 04:00:00.000 Canada/Newfoundland
  ```

List of supported time zones

The following list contains the time zones that can be used with the `AT TIME ZONE` operator in Athena. For additional timezone related functions and examples, see Timezone functions and examples.

```
Africa/Abidjan
Africa/Accra
```

```
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Juba
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
```

```
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
America/Atka
America/Bahia
America/Bahia_Banderas
America/Barbados
America/Belem
America/Belize
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
```

```
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Creston
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
America/Fort_Nelson
America/Fort_Wayne
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
```

```
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles
America/Louisville
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Ojinaga
America/Panama
America/Pangnirtung
```

```
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Punta_Arenas
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
Antarctica/Casey
Antarctica/Davis
```

```
Antarctica/DumontDUrville
Antarctica/Macquarie
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer
Antarctica/Rothera
Antarctica/South_Pole
Antarctica/Syowa
Antarctica/Troll
Antarctica/Vostok
Arctic/Longyearbyen
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Atyrau
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Barnaul
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Chita
Asia/Choibalsan
Asia/Chongqing
Asia/Chungking
Asia/Colombo
Asia/Dacca
Asia/Damascus
Asia/Dhaka
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Hebron
Asia/Ho_Chi_Minh
```

```
Asia/Hong_Kong
Asia/Hovd
Asia/Irkutsk
Asia/Istanbul
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Kathmandu
Asia/Katmandu
Asia/Khandyga
Asia/Kolkata
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao
Asia/Macau
Asia/Magadan
Asia/Makassar
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novokuznetsk
Asia/Novosibirsk
Asia/Omsk
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
Asia/Rangoon
Asia/Riyadh
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Srednekolymsk
```

```
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Tel_Aviv
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Tomsk
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Urumqi
Asia/Ust-Nera
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yangon
Asia/Yekaterinburg
Asia/Yerevan
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Faroe
Atlantic/Jan_Mayen
Atlantic/Madeira
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
Australia/ACT
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Canberra
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/LHI
Australia/Lindeman
Australia/Lord_Howe
```

```
Australia/Melbourne
Australia/NSW
Australia/North
Australia/Perth
Australia/Queensland
Australia/South
Australia/Sydney
Australia/Tasmania
Australia/Victoria
Australia/West
Australia/Yancowinna
Brazil/Acre
Brazil/DeNoronha
Brazil/East
Brazil/West
CET
CST6CDT
Canada/Atlantic
Canada/Central
Canada/Eastern
Canada/Mountain
Canada/Newfoundland
Canada/Pacific
Canada/Saskatchewan
Canada/Yukon
Chile/Continental
Chile/EasterIsland
Cuba
EET
EST5EDT
Egypt
Eire
Europe/Amsterdam
Europe/Andorra
Europe/Astrakhan
Europe/Athens
Europe/Belfast
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Busingen
```

```
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Guernsey
Europe/Helsinki
Europe/Isle_of_Man
Europe/Istanbul
Europe/Jersey
Europe/Kaliningrad
Europe/Kiev
Europe/Kirov
Europe/Lisbon
Europe/Ljubljana
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Mariehamn
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Nicosia
Europe/Oslo
Europe/Paris
Europe/Podgorica
Europe/Prague
Europe/Riga
Europe/Rome
Europe/Samara
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
Europe/Sofia
Europe/Stockholm
Europe/Tallinn
Europe/Tirane
Europe/Tiraspol
Europe/Ulyanovsk
Europe/Uzhgorod
Europe/Vaduz
Europe/Vatican
Europe/Vienna
```

```
Europe/Vilnius
Europe/Volgograd
Europe/Warsaw
Europe/Zagreb
Europe/Zaporozhye
Europe/Zurich
GB
GB-Eire
Hongkong
Iceland
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion
Iran
Israel
Jamaica
Japan
Kwajalein
Libya
MET
MST7MDT
Mexico/BajaNorte
Mexico/BajaSur
Mexico/General
NZ
NZ-CHAT
Navajo
PRC
PST8PDT
Pacific/Apia
Pacific/Auckland
Pacific/Bougainville
Pacific/Chatham
Pacific/Chuuk
Pacific/Easter
Pacific/Efate
```

```
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein
Pacific/Majuro
Pacific/Marquesas
Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Pohnpei
Pacific/Ponape
Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Samoa
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
Pacific/Yap
Poland
Portugal
ROK
Singapore
Turkey
US/Alaska
US/Aleutian
US/Arizona
```

```
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific
US/Pacific-New
US/Samoa
W-SU
WET
```

DDL statements

Use the supported data definition language (DDL) statements presented here directly in Athena. The Athena query engine is based in part on HiveQL DDL. Athena does not support all DDL statements, and there are some differences between HiveQL DDL and Athena DDL. For more information, see the reference topics in this section and Unsupported DDL.

Topics

- DESCRIBE
- DESCRIBE VIEW
- DROP DATABASE
- DROP TABLE
- DROP VIEW
- MSCK REPAIR TABLE
- SHOW COLUMNS
- SHOW CREATE TABLE
- SHOW CREATE VIEW
- SHOW DATABASES
- SHOW PARTITIONS
- SHOW TABLES
- SHOW TBLPROPERTIES
- SHOW VIEWS

Unsupported DDL

The following DDL statements are not supported by Athena SQL. For DDL statements supported for Iceberg tables in Athena, see Evolve Iceberg table schema and Perform other DDL operations on Iceberg tables.

- ALTER INDEX
- ALTER TABLE *table_name* ARCHIVE PARTITION
- ALTER TABLE *table_name* CLUSTERED BY
- ALTER TABLE *table_name* DROP COLUMN (supported for Iceberg tables)
- ALTER TABLE *table_name* EXCHANGE PARTITION
- ALTER TABLE *table_name* NOT CLUSTERED
- ALTER TABLE *table_name* NOT SKEWED
- ALTER TABLE *table_name* NOT SORTED
- ALTER TABLE *table_name* NOT STORED AS DIRECTORIES
- ALTER TABLE *table_name* partitionSpec CHANGE COLUMNS
- ALTER TABLE *table_name* partitionSpec COMPACT

- ALTER TABLE *table_name* partitionSpec CONCATENATE
- ALTER TABLE *table_name* partitionSpec SET FILEFORMAT
- ALTER TABLE *table_name* RENAME TO (supported for Iceberg tables)
- ALTER TABLE *table_name* SET SERDEPROPERTIES
- ALTER TABLE *table_name* SET SKEWED LOCATION
- ALTER TABLE *table_name* SKEWED BY
- ALTER TABLE *table_name* TOUCH
- ALTER TABLE *table_name* UNARCHIVE PARTITION
- COMMIT
- CREATE INDEX
- CREATE ROLE
- CREATE TABLE *table_name* LIKE *existing_table_name*
- CREATE TEMPORARY MACRO
- DELETE FROM
- DESCRIBE DATABASE
- DFS
- DROP INDEX
- DROP ROLE
- DROP TEMPORARY MACRO
- EXPORT TABLE
- GRANT ROLE
- IMPORT TABLE
- LOCK DATABASE
- LOCK TABLE
- REVOKE ROLE
- ROLLBACK
- SHOW COMPACTIONS
- SHOW CURRENT ROLES
- SHOW GRANT
- SHOW INDEXES

- SHOW LOCKS
- SHOW PRINCIPALS
- SHOW ROLE GRANT
- SHOW ROLES
- SHOW STATS
- SHOW TRANSACTIONS
- START TRANSACTION
- UNLOCK DATABASE
- UNLOCK TABLE

ALTER DATABASE SET DBPROPERTIES

Creates one or more properties for a database. The use of `DATABASE` and `SCHEMA` are interchangeable; they mean the same thing.

Synopsis

```
ALTER {DATABASE|SCHEMA} database_name
  SET DBPROPERTIES ('property_name'='property_value' [, ...] )
```

Parameters

SET DBPROPERTIES ('property_name'='property_value' [, ...]

Specifies a property or properties for the database named `property_name` and establishes the value for each of the properties respectively as `property_value`. If `property_name` already exists, the old value is overwritten with `property_value`.

Examples

```
ALTER DATABASE jd_datasets
  SET DBPROPERTIES ('creator'='John Doe', 'department'='applied mathematics');
```

```
ALTER SCHEMA jd_datasets
  SET DBPROPERTIES ('creator'='Jane Doe');
```

ALTER TABLE ADD COLUMNS

Adds one or more columns to an existing table. When the optional PARTITION syntax is used, updates partition metadata.

Synopsis

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  ADD COLUMNS (col_name data_type)
```

Parameters

PARTITION (partition_col_name = partition_col_value [,...])

Creates a partition with the column name/value combinations that you specify. Enclose `partition_col_value` in quotation marks only if the data type of the column is a string.

ADD COLUMNS (col_name data_type [,col_name data_type,...])

Adds columns after existing columns but before partition columns.

Examples

```
ALTER TABLE events ADD COLUMNS (eventowner string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (event string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (eventdescription
 string)
```

Notes

- To see a new table column in the Athena Query Editor navigation pane after you run `ALTER TABLE ADD COLUMNS`, manually refresh the table list in the editor, and then expand the table again.
- `ALTER TABLE ADD COLUMNS` does not work for columns with the `date` datatype. To workaround this issue, use the `timestamp` datatype instead.

ALTER TABLE ADD PARTITION

Creates one or more partition columns for the table. Each partition consists of one or more distinct column name/value combinations. A separate data directory is created for each specified combination, which can improve query performance in some circumstances. Partitioned columns don't exist within the table data itself, so if you use a column name that has the same name as a column in the table itself, you get an error. For more information, see Partition your data.

In Athena, a table and its partitions must use the same data formats but their schemas may differ. For more information, see Update tables with partitions.

For information about the resource-level permissions required in IAM policies (including `glue:CreatePartition`), see AWS Glue API permissions: Actions and resources reference and Configure access to databases and tables in the AWS Glue Data Catalog. For troubleshooting information about permissions when using Athena, see the Permissions section of the Troubleshoot issues in Athena topic.

Synopsis

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION
  (partition_col1_name = partition_col1_value
  [,partition_col2_name = partition_col2_value]
  [,...])
  [LOCATION 'location1']
  [PARTITION
  (partition_colA_name = partition_colA_value
  [,partition_colB_name = partition_colB_value
  [,...])]
  [LOCATION 'location2']
  [,...]
```

Parameters

When you add a partition, you specify one or more column name/value pairs for the partition and the Amazon S3 path where the data files for that partition reside.

[IF NOT EXISTS]

Causes the error to be suppressed if a partition with the same definition already exists.

PARTITION (partition_col_name = partition_col_value [,...])

Creates a partition with the column name/value combinations that you specify. Enclose `partition_col_value` in string characters only if the data type of the column is a string.

[LOCATION 'location']

Specifies the directory in which to store the partition defined by the preceding statement. The `LOCATION` clause is optional when the data uses Hive-style partitioning (`pk1=v1/pk2=v2/pk3=v3`). With Hive-style partitioning, the full Amazon S3 URI is constructed automatically from the table's location, the partition key names, and the partition key values. For more information, see Partition your data.

Considerations

Amazon Athena does not impose a specific limit on the number of partitions you can add in a single `ALTER TABLE ADD PARTITION` DDL statement. However, if you need to add a significant number of partitions, consider breaking the operation into smaller batches to avoid potential performance issues. The following example uses successive commands to add partitions individually and uses `IF NOT EXISTS` to avoid adding duplicates.

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-01')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-02')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-03')
```

When working with partitions in Athena, also keep in mind the following points:

- Although Athena supports querying AWS Glue tables that have 10 million partitions, Athena cannot read more than 1 million partitions in a single scan.
- To optimize your queries and reduce the number of partitions scanned, consider strategies like partition pruning or using partition indexes.

For additional considerations regarding working with partitions in Athena, see Partition your data.

Examples

The following example adds a single partition to a table for Hive-style partitioned data.

```
ALTER TABLE orders ADD
```

```
  PARTITION (dt = '2016-05-14', country = 'IN');
```

The following example adds multiple partitions to a table for Hive-style partitioned data.

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN')
  PARTITION (dt = '2016-06-01', country = 'IN');
```

When the table is not for Hive-style partitioned data, the `LOCATION` clause is required and should be the full Amazon S3 URI for the prefix that contains the partition's data.

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN') LOCATION 's3://amzn-s3-demo-bucket/
path/to/INDIA_31_May_2016/'
  PARTITION (dt = '2016-06-01', country = 'IN') LOCATION 's3://amzn-s3-demo-bucket/
path/to/INDIA_01_June_2016/';
```

To ignore errors when the partition already exists, use the `IF NOT EXISTS` clause, as in the following example.

```
ALTER TABLE orders ADD IF NOT EXISTS
  PARTITION (dt = '2016-05-14', country = 'IN');
```

Zero byte `_$folder$` files

If you run an `ALTER TABLE ADD PARTITION` statement and mistakenly specify a partition that already exists and an incorrect Amazon S3 location, zero byte placeholder files of the format *`partition_value`*`_$folder$` are created in Amazon S3. You must remove these files manually.

To prevent this from happening, use the `ADD IF NOT EXISTS` syntax in your `ALTER TABLE ADD PARTITION` statement, as in the following example.

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION […]
```

ALTER TABLE CHANGE COLUMN

Changes the name, type, order, or comment for a column in a table.

Synopsis

```
ALTER TABLE [db_name.]table_name
```

```
CHANGE [COLUMN] col_old_name col_new_name column_type
[COMMENT col_comment] [FIRST|AFTER column_name]
```

Examples

The following example changes the column name `area` to `zip`, makes the data type integer, and places the renamed column after the `id` column.

```
ALTER TABLE example_table CHANGE COLUMN area zip int AFTER id
```

The following example adds a comment to the `zip` column in the metadata for `example_table`. To see the comment, use the AWS CLI [`aws athena get-table-metadata`] command or visit the schema for the table in the AWS Glue console.

```
ALTER TABLE example_table CHANGE COLUMN zip zip int COMMENT 'USA zipcode'
```

ALTER TABLE DROP PARTITION

Drops one or more specified partitions for the named table.

Synopsis

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec) [, PARTITION
 (partition_spec)]
```

Parameters

[IF EXISTS]

Suppresses the error message if the partition specified does not exist.

PARTITION (partition_spec)

Each `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value [,...]`.

Examples

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN');
```

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN'), PARTITION (dt = '2014-05-15',
 country = 'IN');
```

Notes

The `ALTER TABLE DROP PARTITION` statement does not provide a single syntax for dropping all partitions at once or support filtering criteria to specify a range of partitions to drop.

As a workaround, you can use the AWS Glue API GetPartitions and BatchDeletePartition actions in scripting. The `GetPartitions` action supports complex filter expressions like those in a SQL `WHERE` expression. After you use `GetPartitions` to create a filtered list of partitions to delete, you can use the `BatchDeletePartition` action to delete the partitions in batches of 25.

ALTER TABLE RENAME PARTITION

Renames a partition value.

> **Note**
>
> ALTER TABLE RENAME PARTITION does not rename partition columns. To change a partition column name, you can use the AWS Glue console. For more information, see Renaming a partition column in AWS Glue later in this document.

Synopsis

For the table named `table_name`, renames the partition value specified by `partition_spec` to the value specified by `new_partition_spec`.

```
ALTER TABLE table_name PARTITION (partition_spec) RENAME TO PARTITION
 (new_partition_spec)
```

Parameters

PARTITION (partition_spec)

Each `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value [,...]`.

Examples

```
ALTER TABLE orders
PARTITION (dt = '2014-05-14', country = 'IN') RENAME TO PARTITION (dt = '2014-05-15',
 country = 'IN');
```

Renaming a partition column in AWS Glue

Use the following procedure to rename partition column names in the AWS Glue console.

To rename a table partition column in the AWS Glue console

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the navigation pane, choose **Tables**.
3. On the **Tables** page, use the **Filter tables** search box to find the table that you want to change.
4. In the **Name** column, choose the link of the table that you want to change.
5. On the details page for the table, in the **Schema** section, do one of the following:
   - To make the name change in JSON format, choose **Edit schema as JSON**.
   - To change the name directly, choose **Edit schema**. This procedure chooses **Edit schema**.
6. Select the check box for the partitioned column that you want to rename, and then choose **Edit**.
7. In the **Edit schema entry** dialog box, for **Name**, enter the new name for the partition column.
8. Choose **Save as new table version**. This action updates the partition column name and preserves the schema evolution history without creating a separate physical copy of your data.
9. To compare table versions, on the details page for the table, choose **Actions**, and then choose **Compare versions**.

Additional resources

For more information about partitioning, see Partition your data.

ALTER TABLE REPLACE COLUMNS

Removes all existing columns from a table created with the LazySimpleSerDe and replaces them with the set of columns specified. When the optional `PARTITION` syntax is used, updates partition metadata. You can also use ALTER TABLE REPLACE COLUMNS to drop columns by specifying only the columns that you want to keep.

Synopsis

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  REPLACE COLUMNS (col_name data_type [, col_name data_type, ...])
```

Parameters

PARTITION (partition_col_name = partition_col_value [,...])

Specifies a partition with the column name/value combinations that you specify. Enclose `partition_col_value` in quotation marks only if the data type of the column is a string.

REPLACE COLUMNS (col_name data_type [,col_name data_type,…])

Replaces existing columns with the column names and datatypes specified.

Notes

- To see the change in table columns in the Athena Query Editor navigation pane after you run `ALTER TABLE REPLACE COLUMNS`, you might have to manually refresh the table list in the editor, and then expand the table again.
- `ALTER TABLE REPLACE COLUMNS` does not work for columns with the `date` datatype. To workaround this issue, use the `timestamp` datatype in the table instead.
- Note that even if you are replacing just a single column, the syntax must be `ALTER TABLE` *`table-name`* `REPLACE COLUMNS`, with *columns* in the plural. You must specify not only the column that you want to replace, but the columns that you want to keep – if not, the columns that you do not specify will be dropped. This syntax and behavior derives from Apache Hive DDL. For reference, see Add/Replace columns in the Apache documentation.

Example

In the following example, the table `names_cities`, which was created using the LazySimpleSerDe, has three columns named `col1`, `col2`, and `col3`. All columns are of type `string`. To show the columns in the table, the following command uses the SHOW COLUMNS statement.

```
SHOW COLUMNS IN names_cities
```

Result of the query:

```
col1
col2
col3
```

The following `ALTER TABLE REPLACE COLUMNS` command replaces the column names with `first_name`, `last_name`, and `city`. The underlying source data is not affected.

```
ALTER TABLE names_cities
REPLACE COLUMNS (first_name string, last_name string, city string)
```

To test the result, `SHOW COLUMNS` is run again.

```
SHOW COLUMNS IN names_cities
```

Result of the query:

```
first_name
last_name
city
```

Another way to show the new column names is to preview the table in the Athena Query Editor or run your own `SELECT` query.

ALTER TABLE SET LOCATION

Changes the location for the table named `table_name`, and optionally a partition with `partition_spec`.

Synopsis

```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new location'
```

Parameters

PARTITION (partition_spec)

Specifies the partition with parameters `partition_spec` whose location you want to change. The `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value`.

SET LOCATION 'new location'

Specifies the new location, which must be an Amazon S3 location. For information about syntax, see Table Location in Amazon S3.

Examples

```
ALTER TABLE customers PARTITION (zip='98040', state='WA') SET LOCATION 's3://amzn-s3-
demo-bucket/custdata/';
```

ALTER TABLE SET TBLPROPERTIES

Adds custom or predefined metadata properties to a table and sets their assigned values. To see the properties in a table, use the SHOW TBLPROPERTIES command.

Apache Hive Managed tables are not supported, so setting `'EXTERNAL'='FALSE'` has no effect.

Synopsis

```
ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value' [ , ... ])
```

Parameters

SET TBLPROPERTIES ('property_name' = 'property_value' [, ...])

Specifies the metadata properties to add as `property_name` and the value for each as `property value`. If `property_name` already exists, its value is set to the newly specified `property_value`.

The following predefined table properties have special uses.

Predefined property	Description
`classification`	Indicates the data type for AWS Glue. Possible values are `csv`, `parquet`, `orc`, `avro`, or `json`. Tables created for Athena in the CloudTrail console add `cloudtrail` as a value for the `classification` property. For more information, see the TBLPROPERTIES section of CREATE TABLE.
`has_encrypted_data`	Indicates whether the dataset specified by `LOCATION` is encrypted. For more information, see the TBLPROPERTIES section of CREATE TABLE and Create tables based on encrypted datasets in Amazon S3.
`orc.compress`	Specifies a compression format for data in ORC format. For more information, see ORC SerDe.
`parquet.compression`	Specifies a compression format for data in Parquet format. For more information, see Parquet SerDe.
`write.compression`	Specifies a compression format for data in the text file or JSON formats. For the Parquet and ORC formats, use the `parquet.compression` and `orc.compress` properties respectively.
`compression_level`	Specifies a compression level to use. This property applies only to ZSTD compression. Possible values are from 1 to 22. The default value is 3. For more information, see Use ZSTD compression levels.
`projection.*`	Custom properties used in partition projection that allow Athena to know what partition patterns to expect when it runs a query on a table. For more information, see Use partition projection with Amazon Athena.
`skip.header.line.count`	Ignores headers in data when you define a table. For more information, see Ignoring headers.

Predefined property	Description
`storage.location.template`	Specifies a custom Amazon S3 path template for projected partitions. For more information, see Set up partition projection.

Examples

The following example adds a comment note to table properties.

```
ALTER TABLE orders
SET TBLPROPERTIES ('notes'="Please don't drop this table.");
```

The following example modifies the table `existing_table` to use Parquet file format with ZSTD compression and ZSTD compression level 4.

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = 4)
```

ALTER VIEW DIALECT

Adds or drops an engine dialect from a AWS Glue Data Catalog view. Applies to AWS Glue Data Catalog views only. Requires `Lake Formation` admin or definer permissions.

For more information about AWS Glue Data Catalog views, see Use Data Catalog views in Athena.

Syntax

```
ALTER VIEW name [ FORCE ] [ ADD|UPDATE ] DIALECT AS query
```

```
ALTER VIEW name [ DROP ] DIALECT
```

FORCE

The `FORCE` keyword causes conflicting engine dialect information in a view to be overwritten with the new definition. The `FORCE` keyword is useful when an update to a Data Catalog view results in conflicting view definitions across existing engine dialects. Suppose a Data Catalog view has both the Athena and Amazon Redshift dialects and the update results in a conflict with Amazon Redshift in the view definition. In this case, you can use the `FORCE` keyword to allow the update to complete and mark the Amazon Redshift dialect as stale. When engines marked as stale query the view, the query fails. The engines throw an exception to disallow stale results. To correct this, update the stale dialects in the view.

ADD

Adds a new engine dialect to the Data Catalog view. The engine specified must not already exist in the Data Catalog view.

UPDATE

Updates an engine dialect that already exists in the Data Catalog view.

DROP

Drops an existing engine dialect from a Data Catalog view. After you drop an engine from a Data Catalog view, the Data Catalog view cannot be queried by the engine that was dropped. Other engine dialects in the view can still query the view.

DIALECT AS

Introduces an engine-specific SQL query.

Examples

```
ALTER VIEW orders_by_date FORCE ADD DIALECT
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date FORCE UPDATE DIALECT
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date DROP DIALECT
```

CREATE DATABASE

Creates a database. The use of DATABASE and SCHEMA is interchangeable. They mean the same thing.

> **Note**
>
> For an example of creating a database, creating a table, and running a SELECT query on the table in Athena, see Get started.

Synopsis

```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] database_name
  [COMMENT 'database_comment']
  [LOCATION 'S3_loc']
  [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

For restrictions on database names in Athena, see Name databases, tables, and columns.

Parameters

[IF NOT EXISTS]

Causes the error to be suppressed if a database named `database_name` already exists.

[COMMENT database_comment]

Establishes the metadata value for the built-in metadata property named `comment` and the value you provide for `database_comment`. In AWS Glue, the COMMENT contents are written to the `Description` field of the database properties.

[LOCATION S3_loc]

Specifies the location where database files and metastore will exist as `S3_loc`. The location must be an Amazon S3 location.

[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]

Allows you to specify custom metadata properties for the database definition.

Examples

```
CREATE DATABASE clickstreams;
```

```
CREATE DATABASE IF NOT EXISTS clickstreams
  COMMENT 'Site Foo clickstream data aggregates'
  LOCATION 's3://amzn-s3-demo-bucket/clickstreams/'
  WITH DBPROPERTIES ('creator'='Jane D.', 'Dept.'='Marketing analytics');
```

Viewing database properties

To view the database properties for a database that you create in AWSDataCatalog using `CREATE DATABASE`, you can use the AWS CLI command [`aws glue get-database`], as in the following example:

```
aws glue get-database --name <your-database-name>
```

In JSON output, the result looks like the following:

```
{
    "Database": {
        "Name": "<your-database-name>",
        "Description": "<your-database-comment>",
        "LocationUri": "s3://amzn-s3-demo-bucket",
        "Parameters": {
            "<your-database-property-name>": "<your-database-property-value>"
        },
        "CreateTime": 1603383451.0,
        "CreateTableDefaultPermissions": [
            {
                "Principal": {
                    "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                },
                "Permissions": [
                    "ALL"
                ]
            }
        ]
    }
}
```

For more information about the AWS CLI, see the AWS Command Line Interface User Guide.

CREATE TABLE

Creates a table with the name and the parameters that you specify.

> **Note**
>
> This page contains summary reference information. For more information about creating tables in Athena and an example `CREATE TABLE` statement, see Create tables in Athena. For an example of creating a database, creating a table, and running a `SELECT` query on the table in Athena, see Get started.

Synopsis

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
 [db_name.]table_name [(col_name data_type [COMMENT col_comment] [, ...] )]
 [COMMENT table_comment]
 [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
 [CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]
 [ROW FORMAT row_format]
 [STORED AS file_format]
 [WITH SERDEPROPERTIES (...)]
 [LOCATION 's3://amzn-s3-demo-bucket/[folder]/']
 [TBLPROPERTIES ( ['has_encrypted_data'='true | false',]
 ['classification'='aws_glue_classification',] property_name=property_value [, ...] ) ]
```

Parameters

EXTERNAL

Specifies that the table is based on an underlying data file that exists in Amazon S3, in the `LOCATION` that you specify. Except when creating Iceberg tables, always use the `EXTERNAL` keyword. If you use `CREATE TABLE` without the `EXTERNAL` keyword for non-Iceberg tables, Athena issues an error. When you create an external table, the data referenced must comply with the default format or the format that you specify with the `ROW FORMAT`, `STORED AS`, and `WITH SERDEPROPERTIES` clauses.

[IF NOT EXISTS]

This parameter checks if a table with the same name already exists. If it does, the parameter returns `TRUE`, and Amazon Athena cancels the `CREATE TABLE` action. Because cancellation occurs before Athena calls the data catalog, it doesn't emit a AWS CloudTrail event.

[db_name.]table_name

Specifies a name for the table to be created. The optional `db_name` parameter specifies the database where the table exists. If omitted, the current database is assumed. If the table name includes numbers, enclose `table_name` in quotation marks, for example `"table123"`. If `table_name` begins with an underscore, use backticks, for example, `` `_mytable` ``. Special characters (other than underscore) are not supported.

Athena table names are case-insensitive; however, if you work with Apache Spark, Spark requires lowercase table names. For restrictions on table names in Athena, see [Name databases, tables, and columns]().

[(col_name data_type [COMMENT col_comment] [, ...])]

Specifies the name for each column to be created, along with the column's data type. Column names do not allow special characters other than underscore `(_)`. If `col_name` begins with an underscore, enclose the column name in backticks, for example `` `_mycolumn` ``. For restrictions on column names in Athena, see [Name databases, tables, and columns]().

The `data_type` value can be any of the following:

- `boolean` – Values are `true` and `false`.
- `tinyint` – A 8-bit signed integer in two's complement format, with a minimum value of -2^7 and a maximum value of 2^7-1.
- `smallint` – A 16-bit signed integer in two's complement format, with a minimum value of -2^15 and a maximum value of 2^15-1.
- `int` – In Data Definition Language (DDL) queries like `CREATE TABLE`, use the `int` keyword to represent an integer. In other queries, use the keyword `integer`, where `integer` is represented as a 32-bit signed value in two's complement format, with a minimum value of-2^31 and a maximum value of 2^31-1. In the JDBC driver, `integer` is returned, to ensure compatibility with business analytics applications.
- `bigint` – A 64-bit signed integer in two's complement format, with a minimum value of -2^63 and a maximum value of 2^63-1.

- `double` – A 64-bit signed double-precision floating point number. The range is 4.94065645841246544e-324d to 1.79769313486231570e+308d, positive or negative. `double` follows the IEEE Standard for Floating-Point Arithmetic (IEEE 754).
- `float` – A 32-bit signed single-precision floating point number. The range is 1.40129846432481707e-45 to 3.40282346638528860e+38, positive or negative. `float` follows the IEEE Standard for Floating-Point Arithmetic (IEEE 754). Equivalent to the `real` in Presto. In Athena, use `float` in DDL statements like `CREATE TABLE` and `real` in SQL functions like `SELECT CAST`. The AWS Glue crawler returns values in `float`, and Athena translates `real` and `float` types internally (see the June 5, 2018 release notes).
- `decimal [ (precision, scale) ]`, where *precision* is the total number of digits, and *scale* (optional) is the number of digits in fractional part, the default is 0. For example, use these type definitions: `decimal(11,5)`, `decimal(15)`. The maximum value for *precision* is 38, and the maximum value for *scale* is 38.

  To specify decimal values as literals, such as when selecting rows with a specific decimal value in a query DDL expression, specify the `decimal` type definition, and list the decimal value as a literal (in single quotes) in your query, as in this example: `decimal_value = decimal '0.12'`.
- `char` – Fixed length character data, with a specified length between 1 and 255, such as `char(10)`. For more information, see CHAR Hive data type.
- `varchar` – Variable length character data, with a specified length between 1 and 65535, such as `varchar(10)`. For more information, see VARCHAR Hive data type.
- `string` – A string literal enclosed in single or double quotes.

  Note

  Non-string data types cannot be cast to `string` in Athena; cast them to `varchar` instead.
- `binary` – (for data in Parquet)
- `date` – A date in ISO format, such as *YYYY*-*MM*-*DD*. For example, `date '2008-09-15'`. An exception is the OpenCSVSerDe, which uses the number of days elapsed since January 1, 1970. For more information, see Open CSV SerDe for processing CSV.
- `timestamp` – Date and time instant in a `java.sql.Timestamp` compatible format up to a maximum resolution of milliseconds, such as *yyyy*-*MM*-*dd* *HH*:*mm*:*ss*[.f...]. For example,

`timestamp '2008-09-15 03:04:05.324'`. An exception is the OpenCSVSerDe, which uses `TIMESTAMP` data in the UNIX numeric format (for example, `1579059880000`). For more information, see Open CSV SerDe for processing CSV.

- `array` < data_type >
- `map` < primitive_type, data_type >
- `struct` < col_name : data_type [comment col_comment] [, ...] >

[COMMENT table_comment]

Creates the `comment` table property and populates it with the `table_comment` you specify.

[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]

Creates a partitioned table with one or more partition columns that have the `col_name`, `data_type` and `col_comment` specified. A table can have one or more partitions, which consist of a distinct column name and value combination. A separate data directory is created for each specified combination, which can improve query performance in some circumstances. Partitioned columns don't exist within the table data itself. If you use a value for `col_name` that is the same as a table column, you get an error. For more information, see Partitioning Data.

> **Note**
>
> After you create a table with partitions, run a subsequent query that consists of the MSCK REPAIR TABLE clause to refresh partition metadata, for example, `MSCK REPAIR TABLE cloudfront_logs;`. For partitions that are not Hive compatible, use ALTER TABLE ADD PARTITION to load the partitions so that you can query the data.

[CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]

Divides, with or without partitioning, the data in the specified `col_name` columns into data subsets called *buckets*. The `num_buckets` parameter specifies the number of buckets to create. Bucketing can improve the performance of some queries on large data sets.

[ROW FORMAT row_format]

Specifies the row format of the table and its underlying source data if applicable. For `row_format`, you can specify one or more delimiters with the `DELIMITED` clause or, alternatively, use the SERDE clause as described below. If ROW FORMAT is omitted or ROW FORMAT DELIMITED is specified, a native SerDe is used.

- [DELIMITED FIELDS TERMINATED BY char [ESCAPED BY char]]
- [DELIMITED COLLECTION ITEMS TERMINATED BY char]
- [MAP KEYS TERMINATED BY char]
- [LINES TERMINATED BY char]
- [NULL DEFINED AS char]

  Available only with Hive 0.13 and when the STORED AS file format is TEXTFILE.

--OR--

- SERDE 'serde_name' [WITH SERDEPROPERTIES ("property_name" = "property_value", "property_name" = "property_value" [, ...])]

  The serde_name indicates the SerDe to use. The WITH SERDEPROPERTIES clause allows you to provide one or more custom properties allowed by the SerDe.

[STORED AS file_format]

Specifies the file format for table data. If omitted, TEXTFILE is the default. Options for file_format are:

- SEQUENCEFILE
- TEXTFILE
- RCFILE
- ORC
- PARQUET
- AVRO
- ION
- INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

[LOCATION 's3://amzn-s3-demo-bucket/[folder]/']

Specifies the location of the underlying data in Amazon S3 from which the table is created. The location path must be a bucket name or a bucket name and one or more folders. If you are using partitions, specify the root of the partitioned data. For more information about table location, see Specify a table location in Amazon S3. For information about data format and permissions, see Amazon S3 considerations.

Use a trailing slash for your folder or bucket. Do not use file names or glob characters.

Use:

`s3://amzn-s3-demo-bucket/`

`s3://amzn-s3-demo-bucket/folder/`

`s3://amzn-s3-demo-bucket/folder/anotherfolder/`

Don't use:

`s3://amzn-s3-demo-bucket`

`s3://amzn-s3-demo-bucket/*`

`s3://amzn-s3-demo-bucket/mydatafile.dat`

[TBLPROPERTIES (['has_encrypted_data'='true | false',] ['classification'='classification_value',] property_name=property_value [, ...])]

Specifies custom metadata key-value pairs for the table definition in addition to predefined table properties, such as `"comment"`.

has_encrypted_data – Athena has a built-in property, `has_encrypted_data`. Set this property to `true` to indicate that the underlying dataset specified by `LOCATION` is encrypted. If omitted and if the workgroup's settings do not override client-side settings, `false` is assumed. If omitted or set to `false` when underlying data is encrypted, the query results in an error. For more information, see Encryption at rest.

classification – Tables created for Athena in the CloudTrail console add `cloudtrail` as a value for the `classification` property. To run ETL jobs, AWS Glue requires that you create a table with the `classification` property to indicate the data type for AWS Glue as `csv`, `parquet`, `orc`, `avro`, or `json`. For example, `'classification'='csv'`. ETL jobs will fail if you do not specify this property. You can subsequently specify it using the AWS Glue console, API, or CLI. For more information, see Create tables for ETL jobs and Authoring Jobs in AWS Glue in the *AWS Glue Developer Guide*.

compression_level – The `compression_level` property specifies the compression level to use. This property applies only to ZSTD compression. Possible values are from 1 to 22. The default value is 3. For more information, see Use ZSTD compression levels.

For more information about other table properties, see ALTER TABLE SET TBLPROPERTIES.

Examples

The following example CREATE TABLE statement creates a table based on tab-separated planet data stored in Amazon S3.

```
CREATE EXTERNAL TABLE planet_data (
  planet_name string,
  order_from_sun int,
  au_to_sun float,
  mass float,
  gravity_earth float,
  orbit_years float,
  day_length float
  )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://amzn-s3-demo-bucket/tsv/'
```

Note the following points:

- The ROW FORMAT DELIMITED clause indicates that the data is delimited by a specific character.
- The FIELDS TERMINATED BY '\t' clause specifies that the fields in the TSV data are separated by the tab character ('\t').
- The STORED AS TEXTFILE clause indicates that the data is stored as plain text files in Amazon S3.

To query the data, you could use a simple SELECT statement like the following:

```
SELECT * FROM planet_data
```

To use the example to create your own TSV table in Athena, replace the table and column names with the names and data types of your own table and columns, and update the LOCATION clause to point to the Amazon S3 path where your TSV files are stored.

For more information about creating tables, see Create tables in Athena.

CREATE TABLE AS

Creates a new table populated with the results of a SELECT query. To create an empty table, use CREATE TABLE. `CREATE TABLE AS` combines a `CREATE TABLE` DDL statement with a `SELECT` DML statement and therefore technically contains both DDL and DML. Note that although `CREATE TABLE AS` is grouped here with other DDL statements, CTAS queries in Athena are treated as DML for Service Quotas purposes. For information about Service Quotas in Athena, see Service Quotas.

> **Note**
>
> For CTAS statements, the expected bucket owner setting does not apply to the destination table location in Amazon S3. The expected bucket owner setting applies only to the Amazon S3 output location that you specify for Athena query results. For more information, see Specify a query result location using the Athena console.

For additional information about `CREATE TABLE AS` that is beyond the scope of this reference topic, see Create a table from query results (CTAS).

Topics

- Synopsis
- CTAS table properties
- Examples

Synopsis

```
CREATE TABLE table_name
[ WITH ( property_name = expression [, ...] ) ]
AS query
[ WITH [ NO ] DATA ]
```

Where:

WITH (property_name = expression [, ...])

A list of optional CTAS table properties, some of which are specific to the data storage format. See CTAS table properties.

query

A SELECT query that is used to create a new table.

> **Important**
>
> If you plan to create a query with partitions, specify the names of partitioned columns last in the list of columns in the `SELECT` statement.

[WITH [NO] DATA]

If `WITH NO DATA` is used, a new empty table with the same schema as the original table is created.

> **Note**
>
> To include column headers in your query result output, you can use a simple `SELECT` query instead of a CTAS query. You can retrieve the results from your query results location or download the results directly using the Athena console. For more information, see Work with query results and recent queries.

CTAS table properties

Each CTAS table in Athena has a list of optional CTAS table properties that you specify using `WITH (property_name = expression [, ...] )`. For information about using these parameters, see Examples of CTAS queries.

`WITH (property_name = expression [, ...], )`

`table_type = ['HIVE', 'ICEBERG']`

Optional. The default is `HIVE`. Specifies the table type of the resulting table

Example:

```
WITH (table_type ='ICEBERG')
```

external_location = [location]

> **Note**
>
> Because Iceberg tables are not external, this property does not apply to Iceberg tables. To define the root location of an Iceberg table in a CTAS statement, use the `location` property described later in this section.

Optional. The location where Athena saves your CTAS query in Amazon S3.

Example:

```
WITH (external_location ='s3://amzn-s3-demo-bucket/tables/parquet_table/')
```

Athena does not use the same path for query results twice. If you specify the location manually, make sure that the Amazon S3 location that you specify has no data. Athena never attempts to delete your data. If you want to use the same location again, manually delete the data, or your CTAS query will fail.

If you run a CTAS query that specifies an `external_location` in a workgroup that enforces a query results location, the query fails with an error message. To see the query results location specified for the workgroup, see the workgroup's details.

If your workgroup overrides the client-side setting for query results location, Athena creates your table in the following location:

```
s3://amzn-s3-demo-bucket/tables/query-id/
```

If you do not use the `external_location` property to specify a location and your workgroup does not override client-side settings, Athena uses your client-side setting for the query results location to create your table in the following location:

```
s3://amzn-s3-demo-bucket/Unsaved-or-query-name/year/month/date/tables/query-id/
```

is_external = [boolean]

Optional. Indicates if the table is an external table. The default is true. For Iceberg tables, this must be set to false.

Example:

```
WITH (is_external = false)
```

`location = [location]`

Required for Iceberg tables. Specifies the root location for the Iceberg table to be created from the query results.

Example:

```
WITH (location ='s3://amzn-s3-demo-bucket/tables/iceberg_table/')
```

`field_delimiter = [delimiter]`

Optional and specific to text-based data storage formats. The single-character field delimiter for files in CSV, TSV, and text files. For example, `WITH (field_delimiter = ',')`. Currently, multicharacter field delimiters are not supported for CTAS queries. If you don't specify a field delimiter, `\001` is used by default.

`format = [storage_format]`

The storage format for the CTAS query results, such as `ORC`, `PARQUET`, `AVRO`, `JSON`, `ION`, or `TEXTFILE`. For Iceberg tables, the allowed formats are `ORC`, `PARQUET`, and `AVRO`. If omitted, `PARQUET` is used by default. The name of this parameter, `format`, must be listed in lowercase, or your CTAS query will fail.

Example:

```
WITH (format = 'PARQUET')
```

`bucketed_by = ARRAY[ column_name[,…], bucket_count = [int] ]`

> **Note**
>
> This property does not apply to Iceberg tables. For Iceberg tables, use partitioning with bucket transform.

An array list of buckets to bucket data. If omitted, Athena does not bucket your data in this query.

bucket_count = [int]

> **Note**
>
> This property does not apply to Iceberg tables. For Iceberg tables, use partitioning with bucket transform.

The number of buckets for bucketing your data. If omitted, Athena does not bucket your data. Example:

```
CREATE TABLE bucketed_table WITH (
  bucketed_by = ARRAY[column_name],
  bucket_count = 30, format = 'PARQUET',
  external_location ='s3://amzn-s3-demo-bucket/tables/parquet_table/'
) AS
SELECT
  *
FROM
  table_name
```

partitioned_by = ARRAY[col_name[,…]]

> **Note**
>
> This property does not apply to Iceberg tables. To use partition transforms for Iceberg tables, use the `partitioning` property described later in this section.

Optional. An array list of columns by which the CTAS table will be partitioned. Verify that the names of partitioned columns are listed last in the list of columns in the `SELECT` statement.

partitioning = ARRAY[partition_transform, ...]

Optional. Specifies the partitioning of the Iceberg table to be created. Iceberg supports a wide variety of partition transforms and partition evolution. Partition transforms are summarized in the following table.

Transform	Description
`year(ts)`	Creates a partition for each year. The partition value is the integer difference in years between `ts` and January 1, 1970.
`month(ts)`	Creates a partition for each month of each year. The partition value is the integer difference in months between `ts` and January 1, 1970.
`day(ts)`	Creates a partition for each day of each year. The partition value is the integer difference in days between `ts` and January 1, 1970.
`hour(ts)`	Creates a partition for each hour of each day. The partition value is a timestamp with the minutes and seconds set to zero.
`bucket(x, nbuckets)`	Hashes the data into the specified number of buckets. The partition value is an integer hash of `x`, with a value between 0 and `nbuckets - 1`, inclusive.
`truncate(s, nchars)`	Makes the partition value the first `nchars` characters of `s`.

Example:

```
WITH (partitioning = ARRAY['month(order_date)',
                           'bucket(account_number, 10)',
                           'country']))
```

`optimize_rewrite_min_data_file_size_bytes = [long]`

Optional. Data optimization specific configuration. Files smaller than the specified value are included for optimization. The default is 0.75 times the value of `write_target_data_file_size_bytes`. This property applies only to Iceberg tables. For more information, see Optimize Iceberg tables.

Example:

```
WITH (optimize_rewrite_min_data_file_size_bytes = 402653184)
```

optimize_rewrite_max_data_file_size_bytes = [long]

Optional. Data optimization specific configuration. Files larger than the specified value are included for optimization. The default is 1.8 times the value of `write_target_data_file_size_bytes`. This property applies only to Iceberg tables. For more information, see Optimize Iceberg tables.

Example:

```
WITH (optimize_rewrite_max_data_file_size_bytes = 966367641)
```

optimize_rewrite_data_file_threshold = [int]

Optional. Data optimization specific configuration. If there are fewer data files that require optimization than the given threshold, the files are not rewritten. This allows the accumulation of more data files to produce files closer to the target size and skip unnecessary computation for cost savings. The default is 5. This property applies only to Iceberg tables. For more information, see Optimize Iceberg tables.

Example:

```
WITH (optimize_rewrite_data_file_threshold = 5)
```

optimize_rewrite_delete_file_threshold = [int]

Optional. Data optimization specific configuration. If there are fewer delete files associated with a data file than the threshold, the data file is not rewritten. This allows the accumulation of more delete files for each data file for cost savings. The default is 2. This property applies only to Iceberg tables. For more information, see Optimize Iceberg tables.

Example:

```
WITH (optimize_rewrite_delete_file_threshold = 2)
```

vacuum_min_snapshots_to_keep = [int]

Optional. Vacuum specific configuration. The minimum number of most recent snapshots to retain. The default is 1. This property applies only to Iceberg tables. For more information, see VACUUM.

> **Note**
>
> The `vacuum_min_snapshots_to_keep` property requires Athena engine version 3.

Example:

```
WITH (vacuum_min_snapshots_to_keep = 1)
```

`vacuum_max_snapshot_age_seconds = [long]`

Optional. Vacuum specific configuration. A period in seconds that represents the age of the snapshots to retain. The default is 432000 (5 days). This property applies only to Iceberg tables. For more information, see [VACUUM](#).

> **Note**
>
> The `vacuum_max_snapshot_age_seconds` property requires Athena engine version 3.

Example:

```
WITH (vacuum_max_snapshot_age_seconds = 432000)
```

`write_compression = [compression_format]`

The compression type to use for any storage format that allows compression to be specified. The `compression_format` value specifies the compression to be used when the data is written to the table. You can specify compression for the `TEXTFILE`, `JSON`, `PARQUET`, and `ORC` file formats.

For example, if the `format` property specifies `PARQUET` as the storage format, the value for `write_compression` specifies the compression format for Parquet. In this case, specifying a value for `write_compression` is equivalent to specifying a value for `parquet_compression`.

Similarly, if the `format` property specifies `ORC` as the storage format, the value for `write_compression` specifies the compression format for ORC. In this case, specifying a value for `write_compression` is equivalent to specifying a value for `orc_compression`.

Multiple compression format table properties cannot be specified in the same CTAS query. For example, you cannot specify both `write_compression` and `parquet_compression` in the same query. The same applies for `write_compression` and `orc_compression`. For information about the compression types that are supported for each file format, see Use compression in Athena.

`orc_compression = [compression_format]`

The compression type to use for the `ORC` file format when `ORC` data is written to the table. For example, `WITH (orc_compression = 'ZLIB')`. Chunks within the `ORC` file (except the `ORC` Postscript) are compressed using the compression that you specify. If omitted, ZLIB compression is used by default for `ORC`.

> **Note**
>
> For consistency, we recommend that you use the `write_compression` property instead of `orc_compression`. Use the `format` property to specify the storage format as `ORC`, and then use the `write_compression` property to specify the compression format that `ORC` will use.

`parquet_compression = [compression_format]`

The compression type to use for the Parquet file format when Parquet data is written to the table. For example, `WITH (parquet_compression = 'SNAPPY')`. This compression is applied to column chunks within the Parquet files. If omitted, GZIP compression is used by default for Parquet.

> **Note**
>
> For consistency, we recommend that you use the `write_compression` property instead of `parquet_compression`. Use the `format` property to specify the storage format as `PARQUET`, and then use the `write_compression` property to specify the compression format that `PARQUET` will use.

compression_level = [compression_level]

The compression level to use. This property applies only to ZSTD compression. Possible values are from 1 to 22. The default value is 3. For more information, see Use ZSTD compression levels.

Examples

For examples of CTAS queries, consult the following resources.

- Examples of CTAS queries
- Use CTAS and INSERT INTO for ETL and data analysis
- Use CTAS statements with Amazon Athena to reduce cost and improve performance
- Use CTAS and INSERT INTO to work around the 100 partition limit

CREATE VIEW and CREATE DIALECT VIEW

A view is a logical table that can be referenced by future queries. Views do not contain any data and do not write data. Instead, the query specified by the view runs each time you reference the view by another query.

- `CREATE VIEW` creates an Athena view from a specified `SELECT` query. Athena views work within Athena. For more information about Athena views, see Work with views.
- `CREATE PROTECTED MULTI DIALECT VIEW` creates a AWS Glue Data Catalog view in the AWS Glue Data Catalog. AWS Glue Data Catalog views provide a single common view across AWS services like Amazon Athena and Amazon Redshift. For more information about AWS Glue Data Catalog views, see Use Data Catalog views in Athena.

CREATE VIEW

Creates a view for use within Athena.

Synopsis

```
CREATE [ OR REPLACE ] VIEW view_name AS query
```

The optional `OR REPLACE` clause lets you update the existing view by replacing it. For more information, see Create views.

Examples

To create a view `test` from the table `orders`, use a query similar to the following:

```
CREATE VIEW test AS
SELECT
orderkey,
orderstatus,
totalprice / 2 AS half
FROM orders;
```

To create a view `orders_by_date` from the table `orders`, use the following query:

```
CREATE VIEW orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate;
```

To update an existing view, use an example similar to the following:

```
CREATE OR REPLACE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 4 AS quarter
FROM orders;
```

For more information about using Athena views, see [Work with views](#).

CREATE PROTECTED MULTI DIALECT VIEW

Creates a AWS Glue Data Catalog view in the AWS Glue Data Catalog. A Data Catalog view is a single view schema that works across Athena and other SQL engines such as Amazon Redshift and Amazon EMR.

Syntax

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
SECURITY DEFINER
[ SHOW VIEW JSON ]
AS query
```

OR REPLACE

(Optional) Updates the existing view by replacing it. A Data Catalog view cannot be replaced if SQL dialects from other engines are present in the view. If the calling engine has the only SQL dialect present in the view, the view can be replaced.

PROTECTED

Required keyword. Specifies that the view is protected against data leaks. Data Catalog views can only be created as a `PROTECTED` view.

MULTI DIALECT

Specifies that the view supports the SQL dialects of different query engines and can therefore be read by those engines.

SECURITY DEFINER

Specifies that definer semantics are in force for this view. Definer semantics mean that the effective read permissions on the underlying tables belong to the principal or role that defined the view rather than the principal that performs the actual read.

SHOW VIEW JSON

(Optional) Returns the JSON for the Data Catalog view specification without actually creating a view. This "dry-run" option is useful when you want to validate the SQL for the view and return the table metadata that AWS Glue will use.

Example

The following example creates the `orders_by_date` Data Catalog view based on a query on the `orders` table.

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

For more information about using AWS Glue Data Catalog views, see [Use Data Catalog views in Athena](#).

DESCRIBE

Shows one or more columns, including partition columns, for the specified table. This command is useful for examining the attributes of complex columns.

Synopsis

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_spec]
 [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

> **Important**
>
> The syntax for this statement is `DESCRIBE table_name`, not `DESCRIBE TABLE table_name`. Using the latter syntax results in the error message FAILED: SemanticException [Error 10001]: Table not found table.

Parameters

[EXTENDED | FORMATTED]

Determines the format of the output. Omitting these parameters shows column names and their corresponding data types, including partition columns, in tabular format. Specifying `FORMATTED` not only shows column names and data types in tabular format, but also detailed table and storage information. `EXTENDED` shows column and data type information in tabular format, and detailed metadata for the table in Thrift serialized form. This format is less readable and is useful primarily for debugging.

[PARTITION partition_spec]

If included, lists the metadata for the partition specified by `partition_spec`, where `partition_spec` is in the format `(partition_column = partition_col_value, partition_column = partition_col_value, ...)`.

[col_name ([.field_name] | [.'$elem$'] | [.'key'] | [.'$value$'])*]

Specifies the column and attributes to examine. You can specify `.field_name` for an element of a struct, `'$elem$'` for array element, `'$key$'` for a map key, and `'$value$'` for map value. You can specify this recursively to further explore the complex column.

Examples

```
DESCRIBE orders
```

```
DESCRIBE FORMATTED mydatabase.mytable PARTITION (part_col = 100) columnA;
```

The following query and output shows column and data type information from an `impressions` table based on Amazon EMR sample data.

```
DESCRIBE impressions
```

```
requestbegintime          string                                          from
 deserializer
adid                      string                                          from
 deserializer
impressionid              string                                          from
 deserializer
referrer                  string                                          from
 deserializer
useragent                 string                                          from
 deserializer
usercookie                string                                          from
 deserializer
ip                        string                                          from
 deserializer
number                    string                                          from
 deserializer
processid                 string                                          from
 deserializer
browsercookie             string                                          from
 deserializer
requestendtime            string                                          from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                          from
 deserializer
hostname                  string                                          from
 deserializer
sessionid                 string                                          from
 deserializer
```

```
dt                          string

# Partition Information
# col_name                  data_type                   comment

dt                          string
```

The following example query and output show the result for the same table when the `FORMATTED` option is used.

```
DESCRIBE FORMATTED impressions
```

```
requestbegintime            string                                          from
 deserializer
adid                        string                                          from
 deserializer
impressionid                string                                          from
 deserializer
referrer                    string                                          from
 deserializer
useragent                   string                                          from
 deserializer
usercookie                  string                                          from
 deserializer
ip                          string                                          from
 deserializer
number                      string                                          from
 deserializer
processid                   string                                          from
 deserializer
browsercokie                string                                          from
 deserializer
requestendtime              string                                          from
 deserializer
timers                      struct<modellookup:string,requesttime:string>   from
 deserializer
threadid                    string                                          from
 deserializer
hostname                    string                                          from
 deserializer
sessionid                   string                                          from
 deserializer
dt                          string
```

```
# Partition Information
# col_name                data_type                 comment

dt                        string

# Detailed Table Information
Database:                 sampledb
Owner:                    hadoop
CreateTime:               Thu Apr 23 02:55:21 UTC 2020
LastAccessTime:           UNKNOWN
Protect Mode:             None
Retention:                0
Location:                 s3://us-east-1.elasticmapreduce/samples/hive-ads/tables/
impressions
Table Type:               EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                  TRUE
        transient_lastDdlTime     1587610521

# Storage Information
SerDe Library:                         org.openx.data.jsonserde.JsonSerDe
InputFormat:                           org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
 org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat
Compressed:                            No
Num Buckets:                           -1
Bucket Columns:                        []
Sort Columns:                          []
Storage Desc Params:
        paths                                  requestbegintime, adid, impressionid,
 referrer, useragent, usercookie, ip
        serialization.format                   1
```

The following example query and output show the result for the same table when the EXTENDED option is used. The detailed table information is output on a single line, but is formatted here for readability.

```
DESCRIBE EXTENDED impressions
```

```
requestbegintime          string                                        from
 deserializer
```

```
adid                        string                                          from
 deserializer
impressionid                string                                          from
 deserializer
referrer                    string                                          from
 deserializer
useragent                   string                                          from
 deserializer
usercookie                  string                                          from
 deserializer
ip                          string                                          from
 deserializer
number                      string                                          from
 deserializer
processid                   string                                          from
 deserializer
browsercokie                string                                          from
 deserializer
requestendtime              string                                          from
 deserializer
timers                      struct<modellookup:string,requesttime:string>   from
 deserializer
threadid                    string                                          from
 deserializer
hostname                    string                                          from
 deserializer
sessionid                   string                                          from
 deserializer
dt                          string

# Partition Information
# col_name                  data_type                 comment

dt                          string

Detailed Table Information       Table(tableName:impressions, dbName:sampledb,
 owner:hadoop, createTime:1587610521,
lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:
[FieldSchema(name:requestbegintime, type:string, comment:null),
FieldSchema(name:adid, type:string, comment:null), FieldSchema(name:impressionid,
 type:string, comment:null),
FieldSchema(name:referrer, type:string, comment:null), FieldSchema(name:useragent,
 type:string, comment:null),
```

```
FieldSchema(name:usercookie, type:string, comment:null), FieldSchema(name:ip,
 type:string, comment:null),
FieldSchema(name:number, type:string, comment:null), FieldSchema(name:processid,
 type:string, comment:null),
FieldSchema(name:browsercokie, type:string, comment:null),
 FieldSchema(name:requestendtime, type:string, comment:null),
FieldSchema(name:timers, type:struct<modellookup:string,requesttime:string>,
 comment:null), FieldSchema(name:threadid,
type:string, comment:null), FieldSchema(name:hostname, type:string, comment:null),
 FieldSchema(name:sessionid,
type:string, comment:null)], location:s3://us-east-1.elasticmapreduce/samples/hive-ads/
tables/impressions,
inputFormat:org.apache.hadoop.mapred.TextInputFormat,
outputFormat:org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat, compressed:false,
 numBuckets:-1,
serdeInfo:SerDeInfo(name:null, serializationLib:org.openx.data.jsonserde.JsonSerDe,
 parameters:{serialization.format=1,
paths=requestbegintime, adid, impressionid, referrer, useragent, usercookie, ip}),
 bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[],
 skewedColValueLocationMaps:{}),
storedAsSubDirectories:false), partitionKeys:[FieldSchema(name:dt, type:string,
 comment:null)],
parameters:{EXTERNAL=TRUE, transient_lastDdlTime=1587610521}, viewOriginalText:null,
 viewExpandedText:null,
tableType:EXTERNAL_TABLE)
```

DESCRIBE VIEW

Shows the list of columns for the specified Athena or AWS Glue Data Catalog view. Useful for examining the attributes of a complex view.

For Data Catalog views, the output of the statement is controlled by Lake Formation access control and shows only the columns that the caller has access to.

Synopsis

```
DESCRIBE [db_name.]view_name
```

Example

```
DESCRIBE orders
```

See also SHOW COLUMNS, SHOW CREATE VIEW, SHOW VIEWS, and DROP VIEW.

DROP DATABASE

Removes the named database from the catalog. If the database contains tables, you must either drop the tables before running `DROP DATABASE` or use the `CASCADE` clause. The use of `DATABASE` and `SCHEMA` are interchangeable. They mean the same thing.

Synopsis

```
DROP {DATABASE | SCHEMA} [IF EXISTS] database_name [RESTRICT | CASCADE]
```

Parameters

[IF EXISTS]

Causes the error to be suppressed if `database_name` doesn't exist.

[RESTRICT|CASCADE]

Determines how tables within `database_name` are regarded during the `DROP` operation. If you specify `RESTRICT`, the database is not dropped if it contains tables. This is the default behavior. Specifying `CASCADE` causes the database and all its tables to be dropped.

Examples

```
DROP DATABASE clickstreams;
```

```
DROP SCHEMA IF EXISTS clickstreams CASCADE;
```

Note

When you try to drop a database whose name has special characters (for example, `my-database`), you may receive an error message. To resolve this issue, try enclosing the database name in back tick (`) characters. For information about naming databases in Athena, see Name databases, tables, and columns.

DROP TABLE

Removes the metadata table definition for the table named `table_name`. When you drop an external table, the underlying data remains intact.

Synopsis

```
DROP TABLE [IF EXISTS] table_name
```

Parameters

[IF EXISTS]

Causes the error to be suppressed if `table_name` doesn't exist.

Examples

```
DROP TABLE fulfilled_orders
```

```
DROP TABLE IF EXISTS fulfilled_orders
```

When using the Athena console query editor to drop a table that has special characters other than the underscore (_), use backticks, as in the following example.

```
DROP TABLE `my-athena-database-01.my-athena-table`
```

When using the JDBC connector to drop a table that has special characters, backtick characters are not required.

```
DROP TABLE my-athena-database-01.my-athena-table
```

DROP VIEW

Drops (deletes) an existing Athena or AWS Glue Data Catalog view. The optional `IF EXISTS` clause causes the error to be suppressed if the view does not exist.

For Data Catalog views, drops the view only if Athena view syntax (dialect) is present in the Data Catalog view. For example, if a user calls `DROP VIEW` from Athena, the view is dropped only if the Athena dialect exists in the view. Otherwise, the operation fails. Dropping Data Catalog views requires Lake Formation admin or view definer permissions.

For more information, see Work with views and Use Data Catalog views in Athena.

Synopsis

```
DROP VIEW [ IF EXISTS ] view_name
```

Examples

```
DROP VIEW orders_by_date
```

```
DROP VIEW IF EXISTS orders_by_date
```

See also CREATE VIEW and CREATE DIALECT VIEW, SHOW COLUMNS, SHOW CREATE VIEW, SHOW VIEWS, and DESCRIBE VIEW.

MSCK REPAIR TABLE

Use the `MSCK REPAIR TABLE` command to update the metadata in the catalog after you add Hive compatible partitions.

The `MSCK REPAIR TABLE` command scans a file system such as Amazon S3 for Hive compatible partitions that were added to the file system after the table was created. `MSCK REPAIR TABLE` compares the partitions in the table metadata and the partitions in S3. If new partitions are present in the S3 location that you specified when you created the table, it adds those partitions to the metadata and to the Athena table.

When you add physical partitions, the metadata in the catalog becomes inconsistent with the layout of the data in the file system, and information about the new partitions needs to be added to the catalog. To update the metadata, run `MSCK REPAIR TABLE` so that you can query the data in the new partitions from Athena.

> **Note**
>
> `MSCK REPAIR TABLE` only adds partitions to metadata; it does not remove them. To remove partitions from metadata after the partitions have been manually deleted in

Amazon S3, run the command ALTER TABLE *table-name* DROP PARTITION. For more information see ALTER TABLE DROP PARTITION.

Considerations and limitations

When using MSCK REPAIR TABLE, keep in mind the following points:

- It is possible it will take some time to add all partitions. If this operation times out, it will be in an incomplete state where only a few partitions are added to the catalog. You should run MSCK REPAIR TABLE on the same table until all partitions are added. For more information, see Partition your data.
- For partitions that are not compatible with Hive, use ALTER TABLE ADD PARTITION to load the partitions so that you can query their data.
- Partition locations to be used with Athena must use the s3 protocol (for example, s3://amzn-s3-demo-bucket/*folder*/). In Athena, locations that use other protocols (for example, s3a://*bucket*/*folder*/) will result in query failures when MSCK REPAIR TABLE queries are run on the containing tables.
- Because MSCK REPAIR TABLE scans both a folder and its subfolders to find a matching partition scheme, be sure to keep data for separate tables in separate folder hierarchies. For example, suppose you have data for table 1 in s3://amzn-s3-demo-bucket1 and data for table 2 in s3://amzn-s3-demo-bucket1/table-2-data. If both tables are partitioned by string, MSCK REPAIR TABLE will add the partitions for table 2 to table 1. To avoid this, use separate folder structures like s3://amzn-s3-demo-bucket1 and s3://amzn-s3-demo-bucket2 instead. Note that this behavior is consistent with Amazon EMR and Apache Hive.
- Due to a known issue, MSCK REPAIR TABLE fails silently when partition values contain a colon (:) character (for example, when the partition value is a timestamp). As a workaround, use ALTER TABLE ADD PARTITION.
- MSCK REPAIR TABLE does not add partition column names that begin with an underscore (_). To work around this limitation, use ALTER TABLE ADD PARTITION.

Synopsis

```
MSCK REPAIR TABLE table_name
```

Examples

```
MSCK REPAIR TABLE orders;
```

Troubleshooting

After you run `MSCK REPAIR TABLE`, if Athena does not add the partitions to the table in the AWS Glue Data Catalog, check the following:

- **AWS Glue access** – Make sure that the AWS Identity and Access Management (IAM) role has a policy that allows the `glue:BatchCreatePartition` action. For more information, see Allow glue:BatchCreatePartition in the IAM policy later in this document.
- **Amazon S3 access** – Make sure that the role has a policy with sufficient permissions to access Amazon S3, including the `s3:DescribeJob` action. For an example of which Amazon S3 actions to allow, see the example bucket policy in Configure cross-account access in Athena to Amazon S3 buckets.
- **Amazon S3 object key casing** – Make sure that the Amazon S3 path is in lower case instead of camel case (for example, `userid` instead of `userId`), or use `ALTER TABLE ADD PARTITION` to specify the object key names. For more information, see Change or redefine the Amazon S3 path later in this document.
- **Query timeouts** – `MSCK REPAIR TABLE` is best used when creating a table for the first time or when there is uncertainty about parity between data and partition metadata. If you use `MSCK REPAIR TABLE` to add new partitions frequently (for example, on a daily basis) and are experiencing query timeouts, consider using ALTER TABLE ADD PARTITION.
- **Partitions missing from file system** – If you delete a partition manually in Amazon S3 and then run `MSCK REPAIR TABLE`, you may receive the error message Partitions missing from filesystem. This occurs because `MSCK REPAIR TABLE` doesn't remove stale partitions from table metadata. To remove the deleted partitions from table metadata, run ALTER TABLE DROP PARTITION instead. Note that SHOW PARTITIONS similarly lists only the partitions in metadata, not the partitions in the file system.
- **"NullPointerException name is null" error**

  If you use the AWS Glue CreateTable API operation or the AWS CloudFormation `AWS::Glue::Table` template to create a table for use in Athena without specifying the `TableType` property and then run a DDL query like `SHOW CREATE TABLE` or `MSCK REPAIR TABLE`, you can receive the error message FAILED: NullPointerException Name is null.

To resolve the error, specify a value for the TableInput `TableType` attribute as part of the AWS Glue `CreateTable` API call or AWS CloudFormation template. Possible values for `TableType` include `EXTERNAL_TABLE` or `VIRTUAL_VIEW`.

This requirement applies only when you create a table using the AWS Glue `CreateTable` API operation or the `AWS::Glue::Table` template. If you create a table for Athena by using a DDL statement or an AWS Glue crawler, the `TableType` property is defined for you automatically.

The following sections provide some additional detail.

Allow glue:BatchCreatePartition in the IAM policy

Review the IAM policies attached to the role that you're using to run `MSCK REPAIR TABLE`. When you use the AWS Glue Data Catalog with Athena, the IAM policy must allow the `glue:BatchCreatePartition` action. For an example of an IAM policy that allows the `glue:BatchCreatePartition` action, see AWS managed policy: AmazonAthenaFullAccess.

Change or redefine the Amazon S3 path

If one or more object keys in the Amazon S3 path are in camel case instead of lower case, `MSCK REPAIR TABLE` might not add the partitions to the AWS Glue Data Catalog. For example, if your Amazon S3 path includes the object key name `userId`, the following partitions might not be added to the AWS Glue Data Catalog:

```
s3://amzn-s3-demo-bucket/path/userId=1/

s3://amzn-s3-demo-bucket/path/userId=2/

s3://amzn-s3-demo-bucket/path/userId=3/
```

To resolve this issue, do one of the following:

- Use lower case instead of camel case when you create your Amazon S3 object keys:

  ```
  s3://amzn-s3-demo-bucket/path/userid=1/

  s3://amzn-s3-demo-bucket/path/userid=2/

  s3://amzn-s3-demo-bucket/path/userid=3/
  ```

- Use ALTER TABLE ADD PARTITION to redefine the location, as in the following example:

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
PARTITION (userId=1)
LOCATION 's3://amzn-s3-demo-bucket/path/userId=1/'
PARTITION (userId=2)
LOCATION 's3://amzn-s3-demo-bucket/path/userId=2/'
PARTITION (userId=3)
LOCATION 's3://amzn-s3-demo-bucket/path/userId=3/'
```

Note that although Amazon S3 object key names can use upper case, Amazon S3 bucket names themselves must always be in lower case. For more information, see Object key naming guidelines and Bucket naming rules in the *Amazon S3 User Guide*.

SHOW COLUMNS

Shows only the column names for a single specified table, Athena view, or Data Catalog view. To obtain more detailed information for Athena views, query the AWS Glue Data Catalog instead. For information and examples, see the following sections of the Query the AWS Glue Data Catalog topic:

- To view column metadata such as data type, see List or search columns for a specified table or view.
- To view all columns for all tables in a specific database in `AwsDataCatalog`, see List or search columns for a specified table or view.
- To view all columns for all tables in all databases in `AwsDataCatalog`, see List all columns for all tables.
- To view the columns that specific tables in a database have in common, see List the columns that specific tables have in common.

For Data Catalog views, the output of the statement is controlled by Lake Formation access control and shows only the columns that the caller has access to.

Synopsis

```
SHOW COLUMNS {FROM|IN} database_name.table_or_view_name
```

```
SHOW COLUMNS {FROM|IN} table_or_view_name [{FROM|IN} database_name]
```

The FROM and IN keywords can be used interchangeably. If *table_or_view_name* or *database_name* has special characters like hyphens, surround the name with back quotes (for example, `` `my-database`.`my-table` ``). Do not surround the *table_or_view_name* or *database_name* with single or double quotes. Currently, the use of LIKE and pattern matching expressions is not supported.

Examples

The following equivalent examples show the columns from the orders table in the customers database. The first two examples assume that customers is the current database.

```
SHOW COLUMNS FROM orders
```

```
SHOW COLUMNS IN orders
```

```
SHOW COLUMNS FROM customers.orders
```

```
SHOW COLUMNS IN customers.orders
```

```
SHOW COLUMNS FROM orders FROM customers
```

```
SHOW COLUMNS IN orders IN customers
```

SHOW CREATE TABLE

Analyzes an existing table named table_name to generate the query that created it.

Synopsis

```
SHOW CREATE TABLE [db_name.]table_name
```

Parameters

TABLE [db_name.]table_name

The db_name parameter is optional. If omitted, the context defaults to the current database.

> **Note**
>
> The table name is required.

Examples

```
SHOW CREATE TABLE orderclickstoday;
```

```
SHOW CREATE TABLE `salesdata.orderclickstoday`;
```

Troubleshooting

If you use the AWS Glue CreateTable API operation or the AWS CloudFormation `AWS::Glue::Table` template to create a table for use in Athena without specifying the `TableType` property and then run a DDL query like `SHOW CREATE TABLE` or `MSCK REPAIR TABLE`, you can receive the error message FAILED: NullPointerException Name is null.

To resolve the error, specify a value for the TableInput `TableType` attribute as part of the AWS Glue `CreateTable` API call or AWS CloudFormation template. Possible values for `TableType` include `EXTERNAL_TABLE` or `VIRTUAL_VIEW`.

This requirement applies only when you create a table using the AWS Glue `CreateTable` API operation or the `AWS::Glue::Table` template. If you create a table for Athena by using a DDL statement or an AWS Glue crawler, the `TableType` property is defined for you automatically.

SHOW CREATE VIEW

Shows the SQL statement that created the specified Athena or Data Catalog view. The SQL returned shows the create view syntax used in Athena. Calling `SHOW CREATE VIEW` on Data Catalog views requires Lake Formation admin or view definer permissions.

Synopsis

```
SHOW CREATE VIEW view_name
```

Examples

```
SHOW CREATE VIEW orders_by_date
```

See also CREATE VIEW and CREATE DIALECT VIEW and DROP VIEW.

SHOW DATABASES

Lists all databases defined in the metastore. You can use `DATABASES` or `SCHEMAS`. They mean the same thing.

The programmatic equivalent of `SHOW DATABASES` is the ListDatabases Athena API action. The equivalent method in AWS SDK for Python (Boto3) is list_databases.

Synopsis

```
SHOW {DATABASES | SCHEMAS} [LIKE 'regular_expression']
```

Parameters

[LIKE '*regular_expression*']

Filters the list of databases to those that match the *regular_expression* that you specify. For wildcard character matching, you can use the combination `.*`, which matches any character zero to unlimited times.

Examples

```
SHOW SCHEMAS;
```

```
SHOW DATABASES LIKE '.*analytics';
```

SHOW PARTITIONS

Lists all the partitions in an Athena table in unsorted order.

Synopsis

```
SHOW PARTITIONS table_name
```

- To show the partitions in a table and list them in a specific order, see the List partitions for a specific table section on the Query the AWS Glue Data Catalog page.

- To view the contents of a partition, see the Query the data section on the Partition your data page.
- SHOW PARTITIONS does not list partitions that are projected by Athena but not registered in the AWS Glue catalog. For information about partition projection, see Use partition projection with Amazon Athena.
- SHOW PARTITIONS lists the partitions in metadata, not the partitions in the actual file system. To update the metadata after you delete partitions manually in Amazon S3, run ALTER TABLE DROP PARTITION.

Examples

The following example query shows the partitions for the `flight_delays_csv` table, which shows flight table data from the US Department of Transportation. For more information about the example `flight_delays_csv` table, see Lazy Simple SerDe for CSV, TSV, and custom-delimited files. The table is partitioned by year.

```
SHOW PARTITIONS flight_delays_csv
```

Results

```
year=2007
year=2015
year=1999
year=1993
year=1991
year=2003
year=1996
year=2014
year=2004
year=2011
...
```

The following example query shows the partitions for the `impressions` table, which contains sample web browsing data. For more information about the example `impressions` table, see Partition your data. The table is partitioned by the `dt` (datetime) column.

```
SHOW PARTITIONS impressions
```

Results

```
dt=2009-04-12-16-00
dt=2009-04-13-18-15
dt=2009-04-14-00-20
dt=2009-04-12-13-00
dt=2009-04-13-02-15
dt=2009-04-14-12-05
dt=2009-04-14-06-15
dt=2009-04-12-21-15
dt=2009-04-13-22-15
...
```

Listing partitions in sorted order

To order the partitions in the results list, use the following `SELECT` syntax instead of `SHOW PARTITIONS`.

```
SELECT * FROM database_name."table_name$partitions" ORDER BY column_name
```

The following query shows the list of partitions for the `flight_delays_csv` example, but in sorted order.

```
SELECT * FROM "flight_delays_csv$partitions" ORDER BY year
```

Results

```
year
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
```

```
...
```

For more information, see the List partitions for a specific table section on the Query the AWS Glue Data Catalog page.

SHOW TABLES

Lists all the base tables and views in a database.

> **Note**
>
> The StatementType parameter for `SHOW TABLES` in GetQueryExecution API operation is categorized as `UTILITY`, not `DDL`.

Synopsis

```
SHOW TABLES [IN database_name] ['regular_expression']
```

Parameters

[IN database_name]

Specifies the `database_name` from which tables will be listed. If omitted, the database from the current context is assumed.

> **Note**
>
> `SHOW TABLES` may fail if `database_name` uses an unsupported character such as a hyphen. As a workaround, try enclosing the database name in backticks.

['regular_expression']

Filters the list of tables to those that match the `regular_expression` you specify. To indicate any character in `AWSDataCatalog` tables, you can use the `*` or `.*` wildcard expression. For Apache Hive databases, use the `.*` wildcard expression. To indicate a choice between characters, use the `|` character.

Examples

Example – Show all of the tables in the database `sampledb`

```
SHOW TABLES IN sampledb
```

```
Results
```

```
alb_logs
cloudfront_logs
elb_logs
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example – Show the names of all tables in `sampledb` that include the word "flights"

```
SHOW TABLES IN sampledb '*flights*'
```

```
Results
```

```
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example – Show the names of all tables in `sampledb` that end in the word "logs"

```
SHOW TABLES IN sampledb '*logs'
```

```
Results
```

```
alb_logs
cloudfront_logs
elb_logs
```

SHOW TBLPROPERTIES

Lists table properties for the named table.

Synopsis

```
SHOW TBLPROPERTIES table_name [('property_name')]
```

Parameters

[('property_name')]

If included, only the value of the property named `property_name` is listed.

Examples

```
SHOW TBLPROPERTIES orders;
```

```
SHOW TBLPROPERTIES orders('comment');
```

SHOW VIEWS

Lists the Athena or Data Catalog views in a list of `STRING` type values. Each value in the list is the name of a view in the specified database, or in the current database if you omit the database name. Use the optional `LIKE` clause with a regular expression to restrict the list of view names. For Data Catalog views, lists only the views that use Athena SQL syntax. Other Data Catalog views are filtered out.

Synopsis

```
SHOW VIEWS [IN database_name] [LIKE 'regular_expression']
```

Parameters

[IN database_name]

Specifies the `database_name` from which views will be listed. If omitted, the database from the current context is assumed.

[LIKE 'regular_expression']

Filters the list of views to those that match the `regular_expression` you specify. Only the wild card character `*`, which indicates any character, or `|`, which indicates a choice between characters, can be used.

Examples

```
SHOW VIEWS
```

```
SHOW VIEWS IN marketing_analytics LIKE 'orders*'
```

See also SHOW COLUMNS, SHOW CREATE VIEW, DESCRIBE VIEW, and DROP VIEW.

Considerations and limitations for SQL queries in Amazon Athena

When running queries in Athena, keep in mind the following considerations and limitations:

- **Stored procedures** – Stored procedures are not supported.
- **Maximum number of partitions** – The maximum number of partitions you can create with `CREATE TABLE AS SELECT` (CTAS) statements is 100. For information, see CREATE TABLE AS. For a workaround, see Use CTAS and INSERT INTO to work around the 100 partition limit.
- **Unsupported statements** – Unsupported statements include the following. For a complete list of unsupported DDL statements in Athena, see Unsupported DDL.
  - `CREATE TABLE LIKE` is not supported.
  - `DESCRIBE INPUT` and `DESCRIBE OUTPUT` are not supported.
  - The `MERGE` statement is supported only for transactional table formats. For more information, see MERGE INTO.
  - `UPDATE` statements are not supported.
  - `DELETE FROM` is not supported.
- **Trino and Presto connectors** – Neither Trino nor Presto connectors are supported. Use Amazon Athena Federated Query to connect data sources. For more information, see Use Amazon Athena Federated Query.
- **Timeouts on tables with many partitions** – Athena may time out when querying a table that has many thousands of partitions. This can happen when the table has many partitions that are not of type `string`. When you use type `string`, Athena prunes partitions at the metastore level. However, when you use other data types, Athena prunes partitions on the server side. The more partitions you have, the longer this process takes and the more likely your queries are to time out. To resolve this issue, set your partition type to `string` so that Athena prunes partitions at the metastore level. This reduces overhead and prevents queries from timing out.

- **S3 Glacier** support – For information about querying restored Amazon S3 Glacier objects, see Query restored Amazon S3 Glacier objects.
- **Files treated as hidden** – Athena treats source files that start with an underscore (_) or a dot (`.`) as hidden. To work around this limitation, rename the files.
- **Row or column size limitation** – The size of a single row or its columns cannot exceed 32 MB. This limit can be exceeded when, for example, a row contains a single column of 35 MB. This is a hard limit of the service and cannot be changed.
- **Max line length in a text file** – The size of single line in a text file has an upper limit of 200 MB. Exceeding this limit can produce the error message TextLineLengthLimitExceededException: Too many bytes before newline. To work around this limitation, make sure that you don't have a single line in a text file exceeding 200 MB.
- **LIMIT clause maximum** – The maximum number of rows that can be specified for the `LIMIT` clause is

  9223372036854775807. When using `ORDER BY`, the maximum number of supported rows for the LIMIT clause is 2147483647. Exceeding this limit results in the error message NOT_SUPPORTED: ORDER BY LIMIT > 2147483647 is not supported.
- **information_schema** – Querying `information_schema` is most performant if you have a small to moderate amount of AWS Glue metadata. If you have a large amount of metadata, errors can occur. For information about querying the `information_schema` database for AWS Glue metadata, see Query the AWS Glue Data Catalog.
- **Array initializations** – Due to a limitation in Java, it is not possible to initialize an array in Athena that has more than 254 arguments.
- **Hidden metadata columns** – The Hive or Iceberg hidden metadata columns `$bucket`, `$file_modified_time`, `$file_size`, and `$partition` are not supported for views. For information about using the `$path` metadata column in Athena, see Getting the file locations for source data in Amazon S3 .

For information about maximum query string length, quotas for query timeouts, and quotas for the active number of DML queries, see Service Quotas.

Troubleshoot issues in Athena

The Athena team has gathered the following troubleshooting information from customer issues. Although not comprehensive, it includes advice regarding some common performance, timeout, and out of memory issues.

Topics

CREATE TABLE AS SELECT (CTAS)

Duplicated data occurs with concurrent CTAS statements

Athena does not maintain concurrent validation for CTAS. Make sure that there is no duplicate CTAS statement for the same location at the same time. Even if a CTAS or INSERT INTO statement fails, orphaned data can be left in the data location specified in the statement.

HIVE_TOO_MANY_OPEN_PARTITIONS

When you use a CTAS statement to create a table with more than 100 partitions, you may receive the error HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets. To work around this limitation, you can use a CTAS statement and a series of `INSERT INTO` statements that create or insert up to 100 partitions each. For more information, see Use CTAS and INSERT INTO to work around the 100 partition limit.

Data file issues

Athena cannot read hidden files

Athena treats sources files that start with an underscore (_) or a dot (.) as hidden. To work around this limitation, rename the files.

Athena reads files that I excluded from the AWS Glue crawler

Athena does not recognize exclude patterns that you specify an AWS Glue crawler. For example, if you have an Amazon S3 bucket that contains both `.csv` and `.json` files and you exclude the `.json` files from the crawler, Athena queries both groups of files. To avoid this, place the files that you want to exclude in a different location.

HIVE_BAD_DATA: Error parsing field value

This error can occur in the following scenarios:

- The data type defined in the table doesn't match the source data, or a single field contains different types of data. For suggested resolutions, see My Amazon Athena query fails with the error "HIVE_BAD_DATA: Error parsing field value for field x: For input string: "12312845691"" in the AWS Knowledge Center.
- Null values are present in an integer field. One workaround is to create the column with the null values as `string` and then use `CAST` to convert the field in a query, supplying a default value of `0` for nulls. For more information, see When I query CSV data in Athena, I get the error "HIVE_BAD_DATA: Error parsing field value '' for field x: For input string: """ in the AWS Knowledge Center.

HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://amzn-s3-demo-bucket

This error can occur when you query an Amazon S3 bucket prefix that has a large number of objects. For more information, see How do I resolve the "HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://amzn-s3-demo-bucket/: Slow down" error in Athena? in the AWS Knowledge Center.

HIVE_CURSOR_ERROR: com.amazonaws.services.s3.model.AmazonS3Exception: The specified key does not exist

This error usually occurs when a file is removed when a query is running. Either rerun the query, or check your workflow to see if another job or process is modifying the files when the query is running.

HIVE_CURSOR_ERROR: Unexpected end of input stream

This message indicates the file is either corrupted or empty. Check the integrity of the file and rerun the query.

HIVE_FILESYSTEM_ERROR: Incorrect fileSize *1234567* for file

This message can occur when a file has changed between query planning and query execution. It usually occurs when a file on Amazon S3 is replaced in-place (for example, a PUT is performed on a key where an object already exists). Athena does not support deleting or replacing the contents of a file when a query is running. To avoid this error, schedule jobs that overwrite or delete files at times when queries do not run, or only write data to new files or partitions.

HIVE_UNKNOWN_ERROR: Unable to create input format

This error can be a result of issues like the following:

- The AWS Glue crawler wasn't able to classify the data format
- Certain AWS Glue table definition properties are empty
- Athena doesn't support the data format of the files in Amazon S3

For more information, see How do I resolve the error "unable to create input format" in Athena? in the AWS Knowledge Center or watch the Knowledge Center video.

The S3 location provided to save your query results is invalid.

Make sure that you have specified a valid S3 location for your query results. For more information, see Specify a query result location in the Work with query results and recent queries topic.

Linux Foundation Delta Lake tables

Delta Lake table schema is out of sync

When you query a Delta Lake table that has a schema in AWS Glue that is outdated, you can receive the following error message:

```
INVALID_GLUE_SCHEMA: Delta Lake table schema in Glue does not match the most recent
 schema of the
Delta Lake transaction log. Please ensure that you have the correct schema defined in
 Glue.
```

The schema can become outdated if it is modified in AWS Glue after it has been added to Athena. To update the schema, perform one of the following steps:

- In AWS Glue, run the AWS Glue crawler.
- In Athena, drop the table and create it again.
- Add missing columns manually, either by using the ALTER TABLE ADD COLUMNS statement in Athena, or by editing the table schema in AWS Glue.

Federated queries

Timeout while calling ListTableMetadata

A call to the ListTableMetadata API can timeout if there are lot of tables in the data source, if the data source is slow, or if the network is slow. To troubleshoot this issue, try the following steps.

- **Check the number of tables** – If you have more than 1000 tables, try reducing the number of tables. For the fastest `ListTableMetadata` response, we recommend having fewer than 1000 tables per catalog.
- **Check the Lambda configuration** – Monitoring the Lambda function behavior is critical. When you use federated catalogs, be sure to examine the execution logs of the Lambda function. Based on the results, adjust the memory and timeout values accordingly. To identify any potential

issues with timeouts, revisit your Lambda configuration. For more information, see Configuring function timeout (console) in the *AWS Lambda Developer Guide*.

- **Check federated data source logs** – Examine the logs and error messages from the federated data source to see if there are any issues or errors. The logs can provide valuable insights into the cause of the timeout.
- **Use `StartQueryExecution` to fetch metadata** – If you have more than 1000 tables, it can take longer than expected to retrieve metadata using your federated connector. Because the asynchronous nature of StartQueryExecution ensures that Athena runs the query in the most optimal way, consider using `StartQueryExecution` as an alternative to `ListTableMetadata`. The following AWS CLI examples show how `StartQueryExecution` can be used instead of `ListTableMetadata` to get all the metadata of tables in your data catalog.

  First, run a query that gets all the tables, as in the following example.

  ```
  aws athena start-query-execution --region us-east-1 \
  --query-string "SELECT table_name FROM information_schema.tables LIMIT 50" \
  --work-group "your-work-group-name"
  ```

  Next, retrieve the metadata of an individual table, as in the following example.

  ```
  aws athena start-query-execution --region us-east-1 \
  --query-string "SELECT * FROM information_schema.columns \
  WHERE table_name = 'your-table-name' AND \
  table_catalog = 'your-catalog-name'" \
  --work-group "your-work-group-name"
  ```

  The time taken to get the results depends on the number of tables in your catalog.

For more information about troubleshooting federated queries, see Common_Problems in the awslabs/aws-athena-query-federation section of GitHub, or see the documentation for the individual Athena data source connectors.

JSON related errors

NULL or incorrect data errors when trying to read JSON data

NULL or incorrect data errors when you try read JSON data can be due to a number of causes. To identify lines that are causing errors when you are using the OpenX SerDe, set `ignore.malformed.json` to `true`. Malformed records will return as NULL. For more information, see I get errors when I try to read JSON data in Amazon Athena in the AWS Knowledge Center or watch the Knowledge Center video.

HIVE_BAD_DATA: Error parsing field value for field 0: java.lang.String cannot be cast to org.openx.data.jsonserde.json.JSONObject

The OpenX JSON SerDe throws this error when it fails to parse a column in an Athena query. This can happen if you define a column as a `map` or `struct`, but the underlying data is actually a `string`, `int`, or other primitive type.

HIVE_CURSOR_ERROR: Row is not a valid JSON object - JSONException: Duplicate key

This error occurs when you use Athena to query AWS Config resources that have multiple tags with the same name in different case. The solution is to run `CREATE TABLE` using `WITH SERDEPROPERTIES 'case.insensitive'='false'` and map the names. For information about `case.insensitive` and mapping, see JSON SerDe libraries. For more information, see How do I resolve "HIVE_CURSOR_ERROR: Row is not a valid JSON object - JSONException: Duplicate key" when reading files from AWS Config in Athena? in the AWS Knowledge Center.

HIVE_CURSOR_ERROR messages with pretty-printed JSON

The Hive JSON SerDe and OpenX JSON SerDe libraries expect each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message like HIVE_CURSOR_ERROR: Row is not a valid JSON Object or HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT when you attempt to query the table after you create it. For more information, see JSON data files in the OpenX SerDe documentation on GitHub.

Multiple JSON records return a SELECT COUNT of 1

If you're using the OpenX JSON SerDe, make sure that the records are separated by a newline character. For more information, see The SELECT COUNT query in Amazon Athena returns only one record even though the input JSON file has multiple records in the AWS Knowledge Center.

Cannot query a table created by a AWS Glue crawler that uses a custom JSON classifier

The Athena engine does not support custom JSON classifiers. To work around this issue, create a new table without the custom classifier. To transform the JSON, you can use CTAS or create a view. For example, if you are working with arrays, you can use the UNNEST option to flatten the JSON. Another option is to use a AWS Glue ETL job that supports the custom classifier, convert the data to parquet in Amazon S3, and then query it in Athena.

MSCK REPAIR TABLE

For information about MSCK REPAIR TABLE related issues, see the Considerations and limitations and Troubleshooting sections of the MSCK REPAIR TABLE page.

Output issues

Unable to verify/create output bucket

This error can occur if the specified query result location doesn't exist or if the proper permissions are not present. For more information, see How do I resolve the "unable to verify/create output bucket" error in Amazon Athena? in the AWS Knowledge Center.

TIMESTAMP result is empty

Athena requires the Java TIMESTAMP format. For more information, see When I query a table in Amazon Athena, the TIMESTAMP result is empty in the AWS Knowledge Center.

Store Athena query output in a format other than CSV

By default, Athena outputs files in CSV format only. To output the results of a `SELECT` query in a different format, you can use the `UNLOAD` statement. For more information, see UNLOAD. You can also use a CTAS query that uses the `format` table property to configure the output format. Unlike `UNLOAD`, the CTAS technique requires the creation of a table. For more information, see How can I store an Athena query output in a format other than CSV, such as a compressed format? in the AWS Knowledge Center.

The S3 location provided to save your query results is invalid

You can receive this error message if your output bucket location is not in the same Region as the Region in which you run your query. To avoid this, specify a query results location in the Region in which you run the query. For steps, see Specify a query result location.

Parquet issues

org.apache.parquet.io.GroupColumnIO cannot be cast to org.apache.parquet.io.PrimitiveColumnIO

This error is caused by a parquet schema mismatch. A column that has a non-primitive type (for example, `array`) has been declared as a primitive type (for example, `string`) in AWS Glue. To troubleshoot this issue, check the data schema in the files and compare it with schema declared in AWS Glue.

Parquet statistics issues

When you read Parquet data, you might receive error messages like the following:

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

To workaround this issue, use the CREATE TABLE or ALTER TABLE SET TBLPROPERTIES statement to set the Parquet SerDe `parquet.ignore.statistics` property to `true`, as in the following examples.

CREATE TABLE example

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES ('parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE example

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

For more information about the Parquet Hive SerDe, see Parquet SerDe.

Partitioning issues

MSCK REPAIR TABLE does not remove stale partitions

If you delete a partition manually in Amazon S3 and then run MSCK REPAIR TABLE, you may receive the error message Partitions missing from filesystem. This occurs because MSCK REPAIR TABLE doesn't remove stale partitions from table metadata. Use ALTER TABLE DROP PARTITION to remove the stale partitions manually. For more information, see the "Troubleshooting" section of the MSCK REPAIR TABLE topic.

MSCK REPAIR TABLE failure

When a large amount of partitions (for example, more than 100,000) are associated with a particular table, `MSCK REPAIR TABLE` can fail due to memory limitations. To work around this limit, use ALTER TABLE ADD PARTITION instead.

MSCK REPAIR TABLE detects partitions but doesn't add them to AWS Glue

This issue can occur if an Amazon S3 path is in camel case instead of lower case or an IAM policy doesn't allow the `glue:BatchCreatePartition` action. For more information, see MSCK REPAIR TABLE detects partitions in Athena but does not add them to the AWS Glue Data Catalog in the AWS Knowledge Center.

Partition projection ranges with the date format of dd-MM-yyyy-HH-mm-ss or yyyy-MM-dd do not work

To work correctly, the date format must be set to `yyyy-MM-dd HH:00:00`. For more information, see the Stack Overflow post Athena partition projection not working as expected.

PARTITION BY doesn't support the BIGINT type

Convert the data type to `string` and retry.

No meaningful partitions available

This error message usually means the partition settings have been corrupted. To resolve this issue, drop the table and create a table with new partitions.

Partition projection does not work in conjunction with range partitions

Check that the time range unit [projection.*<columnName>*.interval.unit] matches the delimiter for the partitions. For example, if partitions are delimited by days, then a range unit of hours will not work.

Partition projection error when range specified by hyphen

Specifying the `range` table property with a hyphen instead of a comma produces an error like INVALID_TABLE_PROPERTY: For input string: "*number*-*number*". Ensure that the range values are separated by a comma, not a hyphen. For more information, see Integer type.

HIVE_UNKNOWN_ERROR: Unable to create input format

One or more of the glue partitions are declared in a different format as each glue partition has their own specific input format independently. Please check how your partitions are defined in AWS Glue.

HIVE_PARTITION_SCHEMA_MISMATCH

If the schema of a partition differs from the schema of the table, a query can fail with the error message HIVE_PARTITION_SCHEMA_MISMATCH.

For each table within the AWS Glue Data Catalog that has partition columns, the schema is stored at the table level and for each individual partition within the table. The schema for partitions are populated by an AWS Glue crawler based on the sample of data that it reads within the partition.

When Athena runs a query, it validates the schema of the table and the schema of any partitions necessary for the query. The validation compares the column data types in order and makes sure that they match for the columns that overlap. This prevents unexpected operations such as adding or removing columns from the middle of a table. If Athena detects that the schema of a partition differs from the schema of the table, Athena may not be able to process the query and fails with `HIVE_PARTITION_SCHEMA_MISMATCH`.

There are a few ways to fix this issue. First, if the data was accidentally added, you can remove the data files that cause the difference in schema, drop the partition, and re-crawl the data. Second, you can drop the individual partition and then run `MSCK REPAIR` within Athena to re-create the partition using the table's schema. This second option works only if you are confident that the schema applied will continue to read the data correctly.

SemanticException table is not partitioned but partition spec exists

This error can occur when no partitions were defined in the CREATE TABLE statement. For more information, see How can I troubleshoot the error "FAILED: SemanticException table is not partitioned but partition spec exists" in Athena? in the AWS Knowledge Center.

Zero byte _$folder$ files

If you run an ALTER TABLE ADD PARTITION statement and mistakenly specify a partition that already exists and an incorrect Amazon S3 location, zero byte placeholder files of the format *partition_value*_$folder$ are created in Amazon S3. You must remove these files manually.

To prevent this from happening, use the ADD IF NOT EXISTS syntax in your ALTER TABLE ADD PARTITION statement, like this:

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITIION […]
```

Zero records returned from partitioned data

This issue can occur for a variety of reasons. For possible causes and resolutions, see I created a table in Amazon Athena with defined partitions, but when I query the table, zero records are returned in the AWS Knowledge Center.

See also HIVE_TOO_MANY_OPEN_PARTITIONS.

Permissions

Access denied error when querying Amazon S3

This can occur when you don't have permission to read the data in the bucket, permission to write to the results bucket, or the Amazon S3 path contains a Region endpoint like us-east-1.amazonaws.com. For more information, see When I run an Athena query, I get an "access denied" error in the AWS Knowledge Center.

Access denied with status code: 403 error when running DDL queries on encrypted data in Amazon S3

When you may receive the error message Access Denied (Service: Amazon S3; Status Code: 403; Error Code: AccessDenied; Request ID: *<request_id>*) if the following conditions are true:

1. You run a DDL query like ALTER TABLE ADD PARTITION or MSCK REPAIR TABLE.

2. You have a bucket that has default encryption configured to use `SSE-S3`.
3. The bucket also has a bucket policy like the following that forces `PutObject` requests to specify the PUT headers `"s3:x-amz-server-side-encryption": "true"` and `"s3:x-amz-server-side-encryption": "AES256"`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption": "true"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "AES256"
                }
            }
        }
    ]
}
```

In a case like this, the recommended solution is to remove the bucket policy like the one above given that the bucket's default encryption is already present.

Access denied with status code: 403 when querying an Amazon S3 bucket in another account

This error can occur when you try to query logs written by another AWS service and the second account is the bucket owner but does not own the objects in the bucket. For more information, see I get the Amazon S3 exception "access denied with status code: 403" in Amazon Athena when I query a bucket in another account in the AWS Knowledge Center.

Use IAM role credentials to connect to the Athena JDBC driver

You can retrieve a role's temporary credentials to authenticate the JDBC connection to Athena. Temporary credentials have a maximum lifespan of 12 hours. For more information, see How can I use my IAM role credentials or switch to another IAM role when connecting to Athena using the JDBC driver? in the AWS Knowledge Center.

Required table storage descriptor is not populated

This can occur when you try to query or view a table that you don't have permissions to. For this, the recommended solution is to grant `DESCRIBE` and `SELECT` permissions on the resources through AWS Lake Formation. If your resource is shared across accounts, where original resource exists in account A and you want to query against a linked resource in account B. You must ensure that your role has `DESCRIBE` permission on the original resource in account A, and `SELECT` permission on the linked resource in account B.

Query syntax issues

FAILED: NullPointerException name is null

If you use the AWS Glue CreateTable API operation or the AWS CloudFormation `AWS::Glue::Table` template to create a table for use in Athena without specifying the `TableType` property and then run a DDL query like `SHOW CREATE TABLE` or `MSCK REPAIR TABLE`, you can receive the error message FAILED: NullPointerException Name is null.

To resolve the error, specify a value for the TableInput `TableType` attribute as part of the AWS Glue `CreateTable` API call or AWS CloudFormation template. Possible values for `TableType` include `EXTERNAL_TABLE` or `VIRTUAL_VIEW`.

This requirement applies only when you create a table using the AWS Glue `CreateTable` API operation or the `AWS::Glue::Table` template. If you create a table for Athena by using a DDL statement or an AWS Glue crawler, the `TableType` property is defined for you automatically.

Function not registered

This error occurs when you try to use a function that Athena doesn't support. For a list of functions that Athena supports, see Functions in Amazon Athena or run the `SHOW FUNCTIONS` statement in the Query Editor. You can also write your own user defined function (UDF). For more information, see How do I resolve the "function not registered" syntax error in Athena? in the AWS Knowledge Center.

GENERIC_INTERNAL_ERROR exceptions

`GENERIC_INTERNAL_ERROR` exceptions can have a variety of causes, including the following:

- **GENERIC_INTERNAL_ERROR: Null** – You might see this exception under either of the following conditions:
  - You have a schema mismatch between the data type of a column in table definition and the actual data type of the dataset.
  - You are running a `CREATE TABLE AS SELECT` (CTAS) query with inaccurate syntax.
- **GENERIC_INTERNAL_ERROR: Parent builder is null** – You might see this exception when you query a table with columns of data type `array`, and you are using the OpenCSVSerDe library. The OpenCSVSerde format doesn't support the `array` data type.
- **GENERIC_INTERNAL_ERROR: Value exceeds MAX_INT** – You might see this exception when the source data column is defined with the data type `INT` and has a numeric value greater than 2,147,483,647.
- **GENERIC_INTERNAL_ERROR: Value exceeds MAX_BYTE** – You might see this exception when the source data column has a numeric value exceeding the allowable size for the data type `BYTE`. The data type `BYTE` is equivalent to `TINYINT`. `TINYINT` is an 8-bit signed integer in two's complement format with a minimum value of -128 and a maximum value of 127.
- **GENERIC_INTERNAL_ERROR: Number of partition values does not match number of filters** – You might see this exception if you have inconsistent partitions on Amazon Simple Storage Service(Amazon S3) data. You might have inconsistent partitions under either of the following conditions:
  - Partitions on Amazon S3 have changed (example: new partitions were added).
  - The number of partition columns in the table do not match those in the partition metadata.

For more detailed information about each of these errors, see How do I resolve the error "GENERIC_INTERNAL_ERROR" when I query a table in Amazon Athena? in the AWS Knowledge Center.

Number of matching groups doesn't match the number of columns

This error occurs when you use the Regex SerDe in a CREATE TABLE statement and the number of regex matching groups doesn't match the number of columns that you specified for the table. For more information, see How do I resolve the RegexSerDe error "number of matching groups doesn't match the number of columns" in amazon Athena? in the AWS Knowledge Center.

queryString failed to satisfy constraint: Member must have length less than or equal to 262144

The maximum query string length in Athena (262,144 bytes) is not an adjustable quota. AWS Support can't increase the quota for you, but you can work around the issue by splitting long queries into smaller ones. For more information, see How can I increase the maximum query string length in Athena? in the AWS Knowledge Center.

SYNTAX_ERROR: Column cannot be resolved

This error can occur when you query a table created by an AWS Glue crawler from a UTF-8 encoded CSV file that has a byte order mark (BOM). AWS Glue doesn't recognize the BOMs and changes them to question marks, which Amazon Athena doesn't recognize. The solution is to remove the question mark in Athena or in AWS Glue.

Too many arguments for function call

In Athena engine version 3, functions cannot take more than 127 arguments. This limitation is by design. If you use a function with more than 127 parameters, an error message like the following occurs:

TOO_MANY_ARGUMENTS: line *nnn*:*nn*: Too many arguments for function call *`function_name`*().

To resolve this issue, use fewer parameters per function call.

Query timeout issues

If you experience timeout errors with your Athena queries, check your CloudTrail logs. Queries can time out due to throttling of AWS Glue or Lake Formation APIs. When these errors occur, the corresponding error messages can indicate a query timeout issue rather than a throttling issue. To troubleshoot the issue, you can check your CloudTrail logs before contacting Support. For more information, see Query AWS CloudTrail logs and Log Amazon Athena API calls with AWS CloudTrail.

For information about query timeout issues with federated queries when you call the `ListTableMetadata` API, see Timeout while calling ListTableMetadata.

Throttling issues

If your queries exceed the limits of dependent services such as Amazon S3, AWS KMS, AWS Glue, or AWS Lambda, the following messages can be expected. To resolve these issues, reduce the number of concurrent calls that originate from the same account.

Service	Error message
AWS Glue	AWSGlueException: Rate exceeded.
AWS KMS	You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
AWS Lambda	Rate exceeded TooManyRequestsException
Amazon S3	AmazonS3Exception: Please reduce your request rate.

For information about ways to prevent Amazon S3 throttling when you use Athena, see Prevent Amazon S3 throttling.

Views

Views created in Apache Hive shell do not work in Athena

Because of their fundamentally different implementations, views created in Apache Hive shell are not compatible with Athena. To resolve this issue, re-create the views in Athena.

View is stale; it must be re-created

You can receive this error if the table that underlies a view has altered or dropped. The resolution is to recreate the view. For more information, see How can I resolve the "view is stale; it must be re-created" error in Athena? in the AWS Knowledge Center.

Workgroups

For information about troubleshooting workgroup issues, see Troubleshoot workgroup errors.

Additional resources

The following pages provide additional information for troubleshooting issues with Amazon Athena.

- Athena error catalog
- Service Quotas
- Considerations and limitations for SQL queries in Amazon Athena
- Unsupported DDL
- Name databases, tables, and columns
- Data types in Amazon Athena
- Choose a SerDe for your data
- Use compression in Athena
- Escape reserved keywords in queries
- Troubleshoot workgroup errors

The following AWS resources can also be of help:

- Athena topics in the AWS knowledge center
- Amazon Athena questions on AWS re:Post
- Athena posts in the AWS big data blog

Troubleshooting often requires iterative query and discovery by an expert or from a community of helpers. If you continue to experience issues after trying the suggestions on this page, contact AWS Support (in the AWS Management Console, click **Support**, **Support Center**) or ask a question on AWS re:Post using the **Amazon Athena** tag.

Athena error catalog

Athena provides standardized error information to help you understand failed queries and take steps after a query failure occurs. The `AthenaError` feature includes an `ErrorCategory` field and an `ErrorType` field. `ErrorCategory` specifies whether the cause of the failed query is due to system error, user error, or other error. `ErrorType` provides more granular information regarding the source of the failure. By combining the two fields, you can get a better understanding of the circumstances surrounding and causes for the specific error that occurred.

Error category

The following table lists the Athena error category values and their meanings.

Error category	Source
1	SYSTEM
2	USER
3	OTHER

Error type reference

The following table lists the Athena error type values and their meanings.

Error type	Description
0	Query exhausted resources at this scale factor
1	Query exhausted resources at this scale factor
2	Query exhausted resources at this scale factor
3	Query exhausted resources at this scale factor
4	Query exhausted resources at this scale factor
5	Query exhausted resources at this scale factor
6	Query exhausted resources at this scale factor
7	Query exhausted resources at this scale factor
8	Query exhausted resources at this scale factor

Error type	Description
100	Internal service error
200	Query engine had an internal error
201	Query engine had an internal error
202	Query engine had an internal error
203	Driver error
204	The metastore had an error
205	Query engine had an internal error
206	Query timed out
207	Query engine had an internal error
208	Query engine had an internal error
209	Failed to cancel query
210	Query timed out
211	Query engine had an internal error
212	Query engine had an internal error
213	Query engine had an internal error
214	Query engine had an internal error
215	Query engine had an internal error
216	Query engine had an internal error
217	Query engine had an internal error
218	Query engine had an internal error

Error type	Description
219	Query engine had an internal error
220	Query engine had an internal error
221	Query engine had an internal error
222	Query engine had an internal error
223	Query engine had an internal error
224	Query engine had an internal error
225	Query engine had an internal error
226	Query engine had an internal error
227	Query engine had an internal error
228	Query engine had an internal error
229	Query engine had an internal error
230	Query engine had an internal error
231	Query engine had an internal error
232	Query engine had an internal error
233	Iceberg error
234	Lake Formation error
235	Query engine had an internal error
236	Query engine had an internal error
237	Serialization error
238	Failed to upload metadata to Amazon S3

Error type	Description
239	General persistence error
240	Failed to submit query
300	Internal service error
301	Internal service error
302	Internal service error
303	Internal service error
400	Internal service error
401	Failed to write query results to Amazon S3
402	Failed to write query results to Amazon S3
1000	User error
1001	Data error
1002	Data error
1003	DDL task failed
1004	Schema error
1005	Serialization error
1006	Syntax error
1007	Data error
1008	Query rejected
1009	Query failed
1010	Internal service error

Error type	Description
1011	Query canceled by user
1012	Query engine had an internal error
1013	Query engine had an internal error
1014	Query canceled by user
1100	Invalid argument provided
1101	Invalid property provided
1102	Query engine had an internal error
1103	Invalid table property provided
1104	Query engine had an internal error
1105	Query engine had an internal error
1106	Invalid function argument provided
1107	Invalid view
1108	Failed to register function
1109	Provided Amazon S3 path not found
1110	Provided table or view does not exist
1200	Query not supported
1201	Provided decoder not supported
1202	Query type not supported
1300	General not found error
1301	General entity not found

Error type	Description
1302	File not found
1303	Provided function or function implementation not found
1304	Query engine had an internal error
1305	Query engine had an internal error
1306	Amazon S3 bucket not found
1307	Selected engine not found
1308	Query engine had an internal error
1400	Throttling error
1401	Query failed due to AWS Glue throttling
1402	Query failed due to too many table versions in AWS Glue
1403	Query failed due to Amazon S3 throttling
1404	Query failed due to Amazon Athena throttling
1405	Query failed due to Amazon Athena throttling
1406	Query failed due to Amazon Athena throttling
1500	Permission error
1501	Amazon S3 permission error
1602	Exceeded reserved capacity limit. Insufficient capacity to execute this query.
1700	Query failed due to a Lake Formation internal exception
1701	Query failed due to an AWS Glue internal exception
9999	Internal service error

Code samples

The examples in this topic use SDK for Java 2.x as a starting point for writing Athena applications.

> **Note**
>
> For information about programming Athena using other language-specific AWS SDKs, see the following resources:
>
> - AWS Command Line Interface (`athena`)
> - AWS SDK for .NET (`Amazon.Athena.Model`)
> - AWS SDK for C++ (`Aws::Athena::AthenaClient`)
> - AWS SDK for Go (`athena`)
> - AWS SDK for JavaScript v3 (`AthenaClient`)
> - AWS SDK for PHP 3.x (`Aws\Athena`)
> - AWS SDK for Python (Boto3) (`Athena.Client`)
> - AWS SDK for Ruby v3 (`Aws::Athena::Client`)

For more information about running the Java code examples in this section, see the Amazon Athena Java readme on the AWS code examples repository on GitHub. For the Java programming reference for Athena, see AthenaClient in the AWS SDK for Java 2.x.

> **Note**
>
> These samples use constants (for example, `ATHENA_SAMPLE_QUERY`) for strings, which are defined in an `ExampleConstants.java` class declaration. Replace these constants with your own strings or defined constants.

Topics

- Constants
- Create a client to access Athena
- Start query execution
- Stop query execution

- List query executions
- Create a named query
- Delete a named query
- List named queries

Constants

The `ExampleConstants.java` class demonstrates how to query a table created by the Get started tutorial in Athena.

```
package aws.example.athena;

public class ExampleConstants {

    public static final int CLIENT_EXECUTION_TIMEOUT = 100000;
    public static final String ATHENA_OUTPUT_BUCKET = "s3://bucketscott2"; // change
 the Amazon S3 bucket name to match
                                                                           // your
 environment
    // Demonstrates how to query a table with a comma-separated value (CSV) table.
    // For information, see
    // https://docs.aws.amazon.com/athena/latest/ug/work-with-data.html
    public static final String ATHENA_SAMPLE_QUERY = "SELECT * FROM scott2;"; // change
 the Query statement to match
                                                                              // your
 environment
    public static final long SLEEP_AMOUNT_IN_MS = 1000;
    public static final String ATHENA_DEFAULT_DATABASE = "mydatabase"; // change the
 database to match your database

}
```

Create a client to access Athena

The `AthenaClientFactory.java` class shows how to create and configure an Amazon Athena client.

```
package aws.example.athena;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.AthenaClientBuilder;

public class AthenaClientFactory {
    private final AthenaClientBuilder builder = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create());

    public AthenaClient createClient() {
        return builder.build();
    }
}
```

Start query execution

The `StartQueryExample` shows how to submit a query to Athena, wait until the results become available, and then process the results.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.GetQueryResultsRequest;
import software.amazon.awssdk.services.athena.model.GetQueryResultsResponse;
import software.amazon.awssdk.services.athena.model.ColumnInfo;
import software.amazon.awssdk.services.athena.model.Row;
import software.amazon.awssdk.services.athena.model.Datum;
import software.amazon.awssdk.services.athena.paginators.GetQueryResultsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StartQueryExample {

    public static void main(String[] args) throws InterruptedException {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String queryExecutionId = submitAthenaQuery(athenaClient);
        waitForQueryToComplete(athenaClient, queryExecutionId);
        processResultRows(athenaClient, queryExecutionId);
        athenaClient.close();
    }

    // Submits a sample query to Amazon Athena and returns the execution ID of the
    // query.
    public static String submitAthenaQuery(AthenaClient athenaClient) {
        try {
            // The QueryExecutionContext allows us to set the database.
            QueryExecutionContext queryExecutionContext =
 QueryExecutionContext.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .build();

            // The result configuration specifies where the results of the query should
 go.
            ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                    .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                    .build();

            StartQueryExecutionRequest startQueryExecutionRequest =
 StartQueryExecutionRequest.builder()
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .queryExecutionContext(queryExecutionContext)
                    .resultConfiguration(resultConfiguration)
                    .build();

            StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                    .startQueryExecution(startQueryExecutionRequest);
            return startQueryExecutionResponse.queryExecutionId();
```

```
        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return "";
    }

    // Wait for an Amazon Athena query to complete, fail or to be cancelled.
    public static void waitForQueryToComplete(AthenaClient athenaClient, String
 queryExecutionId)
            throws InterruptedException {
        GetQueryExecutionRequest getQueryExecutionRequest =
 GetQueryExecutionRequest.builder()
                .queryExecutionId(queryExecutionId)
                .build();

        GetQueryExecutionResponse getQueryExecutionResponse;
        boolean isQueryStillRunning = true;
        while (isQueryStillRunning) {
            getQueryExecutionResponse =
 athenaClient.getQueryExecution(getQueryExecutionRequest);
            String queryState =
 getQueryExecutionResponse.queryExecution().status().state().toString();
            if (queryState.equals(QueryExecutionState.FAILED.toString())) {
                throw new RuntimeException(
                        "The Amazon Athena query failed to run with error message: " +
 getQueryExecutionResponse
                                .queryExecution().status().stateChangeReason());
            } else if (queryState.equals(QueryExecutionState.CANCELLED.toString())) {
                throw new RuntimeException("The Amazon Athena query was cancelled.");
            } else if (queryState.equals(QueryExecutionState.SUCCEEDED.toString())) {
                isQueryStillRunning = false;
            } else {
                // Sleep an amount of time before retrying again.
                Thread.sleep(ExampleConstants.SLEEP_AMOUNT_IN_MS);
            }
            System.out.println("The current status is: " + queryState);
        }
    }

    // This code retrieves the results of a query
    public static void processResultRows(AthenaClient athenaClient, String
 queryExecutionId) {
        try {
```

```
            // Max Results can be set but if its not set,
            // it will choose the maximum page size.
            GetQueryResultsRequest getQueryResultsRequest =
 GetQueryResultsRequest.builder()
                    .queryExecutionId(queryExecutionId)
                    .build();

            GetQueryResultsIterable getQueryResultsResults = athenaClient
                    .getQueryResultsPaginator(getQueryResultsRequest);
            for (GetQueryResultsResponse result : getQueryResultsResults) {
                List<ColumnInfo> columnInfoList =
 result.resultSet().resultSetMetadata().columnInfo();
                List<Row> results = result.resultSet().rows();
                processRow(results, columnInfoList);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static void processRow(List<Row> row, List<ColumnInfo> columnInfoList) {
        for (Row myRow : row) {
            List<Datum> allData = myRow.data();
            for (Datum data : allData) {
                System.out.println("The value of the column is " +
 data.varCharValue());
            }
        }
    }
}
```

Stop query execution

The `StopQueryExecutionExample` runs an example query, immediately stops the query, and checks the status of the query to ensure that it was canceled.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.StopQueryExecutionRequest;
```

```
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StopQueryExecutionExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleQueryExecutionId = submitAthenaQuery(athenaClient);
        stopAthenaQuery(athenaClient, sampleQueryExecutionId);
        athenaClient.close();
    }

    public static void stopAthenaQuery(AthenaClient athenaClient, String
 sampleQueryExecutionId) {
        try {
            StopQueryExecutionRequest stopQueryExecutionRequest =
 StopQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            athenaClient.stopQueryExecution(stopQueryExecutionRequest);
            GetQueryExecutionRequest getQueryExecutionRequest =
 GetQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            GetQueryExecutionResponse getQueryExecutionResponse = athenaClient
                    .getQueryExecution(getQueryExecutionRequest);
```

```
            if (getQueryExecutionResponse.queryExecution()
                    .status()
                    .state()
                    .equals(QueryExecutionState.CANCELLED)) {

                System.out.println("The Amazon Athena query has been cancelled!");
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    // Submits an example query and returns a query execution Id value
    public static String submitAthenaQuery(AthenaClient athenaClient) {
        try {
            QueryExecutionContext queryExecutionContext =
 QueryExecutionContext.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .build();

            ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                    .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                    .build();

            StartQueryExecutionRequest startQueryExecutionRequest =
 StartQueryExecutionRequest.builder()
                    .queryExecutionContext(queryExecutionContext)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .resultConfiguration(resultConfiguration).build();

            StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                    .startQueryExecution(startQueryExecutionRequest);
            return startQueryExecutionResponse.queryExecutionId();

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;

    }
```

```
}
```

List query executions

The `ListQueryExecutionsExample` shows how to obtain a list of query execution IDs.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsRequest;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsResponse;
import software.amazon.awssdk.services.athena.paginators.ListQueryExecutionsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListQueryExecutionsExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueryIds(athenaClient);
        athenaClient.close();
    }

    public static void listQueryIds(AthenaClient athenaClient) {
        try {
            ListQueryExecutionsRequest listQueryExecutionsRequest =
 ListQueryExecutionsRequest.builder().build();
            ListQueryExecutionsIterable listQueryExecutionResponses = athenaClient
                    .listQueryExecutionsPaginator(listQueryExecutionsRequest);
            for (ListQueryExecutionsResponse listQueryExecutionResponse :
 listQueryExecutionResponses) {
```

```
                List<String> queryExecutionIds =
 listQueryExecutionResponse.queryExecutionIds();
                System.out.println("\n" + queryExecutionIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Create a named query

The `CreateNamedQueryExample` shows how to create a named query.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class CreateNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;
```

```
        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        createNamedQuery(athenaClient, name);
        athenaClient.close();
    }

    public static void createNamedQuery(AthenaClient athenaClient, String name) {
        try {
            // Create the named query request.
            CreateNamedQueryRequest createNamedQueryRequest =
 CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .description("Sample Description")
                    .name(name)
                    .build();

            athenaClient.createNamedQuery(createNamedQueryRequest);
            System.out.println("Done");

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Delete a named query

The `DeleteNamedQueryExample` shows how to delete a named query by using the named query ID.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.DeleteNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class DeleteNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;

        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleNamedQueryId = getNamedQueryId(athenaClient, name);
        deleteQueryName(athenaClient, sampleNamedQueryId);
        athenaClient.close();
    }

    public static void deleteQueryName(AthenaClient athenaClient, String
 sampleNamedQueryId) {
        try {
```

```
            DeleteNamedQueryRequest deleteNamedQueryRequest =
 DeleteNamedQueryRequest.builder()
                    .namedQueryId(sampleNamedQueryId)
                    .build();

            athenaClient.deleteNamedQuery(deleteNamedQueryRequest);

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static String getNamedQueryId(AthenaClient athenaClient, String name) {
        try {
            CreateNamedQueryRequest createNamedQueryRequest =
 CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .name(name)
                    .description("Sample description")
                    .build();

            CreateNamedQueryResponse createNamedQueryResponse =
 athenaClient.createNamedQuery(createNamedQueryRequest);
            return createNamedQueryResponse.namedQueryId();

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;
    }
}
```

List named queries

The `ListNamedQueryExample` shows how to obtain a list of named query IDs.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
```

```
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesRequest;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesResponse;
import software.amazon.awssdk.services.athena.paginators.ListNamedQueriesIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListNamedQueryExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listNamedQueries(athenaClient);
        athenaClient.close();
    }

    public static void listNamedQueries(AthenaClient athenaClient) {
        try {
            ListNamedQueriesRequest listNamedQueriesRequest =
 ListNamedQueriesRequest.builder()
                    .build();

            ListNamedQueriesIterable listNamedQueriesResponses = athenaClient
                    .listNamedQueriesPaginator(listNamedQueriesRequest);
            for (ListNamedQueriesResponse listNamedQueriesResponse :
 listNamedQueriesResponses) {
                List<String> namedQueryIds = listNamedQueriesResponse.namedQueryIds();
                System.out.println(namedQueryIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
```

```
}
```

Use Apache Spark in Amazon Athena

Amazon Athena makes it easy to interactively run data analytics and exploration using Apache Spark without the need to plan for, configure, or manage resources. Running Apache Spark applications on Athena means submitting Spark code for processing and receiving the results directly without the need for additional configuration. You can use the simplified notebook experience in Amazon Athena console to develop Apache Spark applications using Python or Athena notebook APIs. Apache Spark on Amazon Athena is serverless and provides automatic, on-demand scaling that delivers instant-on compute to meet changing data volumes and processing requirements.

Amazon Athena offers the following features:

- **Console usage** – Submit your Spark applications from the Amazon Athena console.
- **Scripting** – Quickly and interactively build and debug Apache Spark applications in Python.
- **Dynamic scaling** – Amazon Athena automatically determines the compute and memory resources needed to run a job and continuously scales those resources accordingly up to the maximums that you specify. This dynamic scaling reduces cost without affecting speed.
- **Notebook experience** – Use the Athena notebook editor to create, edit, and run computations using a familiar interface. Athena notebooks are compatible with Jupyter notebooks and contain a list of cells that are executed in order as calculations. Cell content can include code, text, Markdown, mathematics, plots and rich media.

For additional information, see Run Spark SQL on Amazon Athena Spark and Explore your data lake using Amazon Athena for Apache Spark in the *AWS Big Data Blog*.

Topics

- Supported data and storage formats
- Monitor Apache Spark calculations with CloudWatch metrics
- Enable requester pays Amazon S3 buckets in Athena for Spark
- Enable Apache Spark encryption
- Configure cross-account AWS Glue access in Athena for Spark
- Understand service quotas for Athena for Spark
- Use Athena notebook APIs
- Troubleshoot Athena for Spark

Considerations and limitations

- Currently, Amazon Athena for Apache Spark is available in the following AWS Regions:
  - Asia Pacific (Mumbai)
  - Asia Pacific (Singapore)
  - Asia Pacific (Sydney)
  - Asia Pacific (Tokyo)
  - Europe (Frankfurt)
  - Europe (Ireland)
  - US East (N. Virginia)
  - US East (Ohio)
  - US West (Oregon)
- AWS Lake Formation is not supported.
- Tables that use partition projection are not supported.
- Apache Spark enabled workgroups can use the Athena notebook editor, but not the Athena query editor. Only Athena SQL workgroups can use the Athena query editor.
- Cross-engine view queries are not supported. Views created by Athena SQL are not queryable by Athena for Spark. Because views for the two engines are implemented differently, they are not compatible for cross-engine use.
- MLlib (Apache Spark machine learning library) and the `pyspark.ml` package are not supported. For a list of supported Python libraries, see the List of preinstalled Python libraries.
- Currently, `pip install` is not supported in Athena for Spark sessions.

- Only one active session per notebook is allowed.
- When multiple users use the console to open an existing session in a workgroup, they access the same notebook. To avoid confusion, only open sessions that you create yourself.
- The hosting domains for Apache Spark applications that you might use with Amazon Athena (for example, `analytics-gateway.us-east-1.amazonaws.com`) are registered in the internet Public Suffix List (PSL). If you ever need to set sensitive cookies in your domains, we recommend that you use cookies with a `__Host-` prefix to help defend your domain against cross-site request forgery (CSRF) attempts. For more information, see the Set-Cookie page in the Mozilla.org developer documentation.
- For information on troubleshooting Spark notebooks, sessions, and workgroups in Athena, see Troubleshoot Athena for Spark.

Get started with Apache Spark on Amazon Athena

To get started with Apache Spark on Amazon Athena, you must first create a Spark enabled workgroup. After you switch to the workgroup, you can create a notebook or open an existing notebook. When you open a notebook in Athena, a new session is started for it automatically and you can work with it directly in the Athena notebook editor.

> **Note**
>
> Make sure that you create a Spark enabled workgroup before you attempt to create a notebook.

Step 1: Create a Spark enabled workgroup in Athena

You can use workgroups in Athena to group users, teams, applications, or workloads, and to track costs. To use Apache Spark in Amazon Athena, you create an Amazon Athena workgroup that uses a Spark engine.

> **Note**
>
> Apache Spark enabled workgroups can use the Athena notebook editor, but not the Athena query editor. Only Athena SQL workgroups can use the Athena query editor.

To create a Spark enabled workgroup in Athena

1. Open the Athena console at https://console.aws.amazon.com/athena/
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the navigation pane, choose **Workgroups**.
4. On the **Workgroups** page, choose **Create workgroup**.
5. For **Workgroup name**, enter a name for your Apache Spark workgroup.
6. (Optional) For **Description**, enter a description for your workgroup.
7. For **Analytics engine**, choose **Apache Spark**.

> **Note**
>
> After you create a workgroup, the workgroup's type of analytics engine cannot be changed. For example, an Athena engine version 3 workgroup cannot be changed to a PySpark engine version 3 workgroup.

8. For the purposes of this tutorial, select **Turn on example notebook**. This optional feature adds an example notebook with the name `example-notebook-`*`random_string`* to your workgroup and adds AWS Glue-related permissions that the notebook uses to create, show, and delete specific databases and tables in your account, and read permissions in Amazon S3 for the sample dataset. To see the added permissions, choose **View additional permissions details**.

> **Note**
>
> Running the example notebook may incur some additional cost.

9. For **Calculation result settings**choose from the following options:

   - **Create a new S3 bucket** – This option creates an Amazon S3 bucket in your account for your calculation results. The bucket name has the format `*account_id*-*region*-athena-results-bucket-*alphanumeric_id*` and uses the settings ACLs disabled, public access blocked, versioning disabled, and bucket owner enforced.
   - **Choose an existing S3 location** – For this option, do the following:
     - Enter the S3 path to an existing the location in the search box, or choose **Browse S3** to choose a bucket from a list.

       > **Note**
       >
       > When you select an existing location in Amazon S3, do not append a forward slash (/) to the location. Doing so causes the link to the calculation results location on the [calculation details page](#) to point to the incorrect directory. If this occurs, edit the workgroup's results location to remove the trailing forward slash.

     - (Optional) Choose **View** to open the **Buckets** page of the Amazon S3 console where you can view more information about the existing bucket that you chose.
     - (Optional) For **Expected bucket owner**, enter the AWS account ID that you expect to be the owner of your query results output location bucket. We recommend that you choose this option as an added security measure whenever possible. If the account ID of the bucket owner does not match the ID that you specify, attempts to output to the bucket will fail. For in-depth information, see [Verifying bucket ownership with bucket owner condition](#) in the *Amazon S3 User Guide*.
     - (Optional) Select **Assign bucket owner full control over query results** if your calculation result location is owned by another account and you want to grant full control over your query results to the other account.

10. (Optional) Choose **Encrypt query results** if you want your query results to be encrypted.

    - For **Encryption type**, choose one of the following options:

- **SSE_S3** – This option uses server-side encryption (SSE) with Amazon S3-managed encryption keys.
- **SSE_KMS** – This option uses server-side encryption (SSE) with AWS KMS-managed keys.

  For **Choose an AWS KMS key**, choose one of the following options.

  - **Use AWS owned key** – The AWS KMS key is owned and managed by AWS. You are not charged an additional fee for using this key.
  - **Choose a different AWS KMS key (advanced)** – For this option, do one of the following:
    - To use an existing key, use the search box to choose an AWS KMS or enter a key ARN.
    - To create a key in the AWS KMS console, choose **Create an AWS KMS key**. Your execution role must have permission to use the key that you create. After you finish creating the key in the KMS console, return to the **Create workgroup** page in Athena console, and then use the **Choose an AWS KMS key or enter an ARN** search box to choose the key that you just created.

> **Important**
>
> When you change the AWS KMS key for a workgroup, notebooks managed before the update still reference the old KMS key. Notebooks managed after the update use the new KMS key. To update the old notebooks to reference the new KMS key, export and then import each of the old notebooks. If you delete the old KMS key before you update the old notebook references to the new KMS key, the old notebooks are no longer decryptable and cannot be recovered.
> This behavior also applies for updates to aliases, which are friendly names for KMS keys. When you update a KMS key alias to point to a new KMS key, notebooks managed before the alias update still reference the old KMS key, and notebooks managed after the alias update use the new KMS key. Consider these points before updating your KMS keys or aliases.

11. For **Additional configurations**, choose **Use defaults**. This option helps you get started with your Spark-enabled workgroup. When you use the defaults, Athena creates an IAM role and calculation results location in Amazon S3 for you. The name of the IAM role and the S3 bucket location to be created are displayed in the box below the **Additional configurations** heading.

    If you do not want to use the defaults, continue with the steps in the (Optional) Specify your own workgroup configurations section to configure your workgroup manually.

12. (Optional) **Tags** – Use this option to add tags to your workgroup. For more information, see Tag Athena resources.
13. Choose **Create workgroup**. A message informs you that the workgroup was created successfully, and the workgroup shows in the list of workgroups.

(Optional) Specify your own workgroup configurations

If you want to specify your own IAM role and calculation results location for your notebook, follow the steps in this section. If you chose **Use defaults** for the **Additional configurations** option, skip this section and go directly to Step 2: Open notebook explorer and switch workgroups .

The following procedure assumes you have completed steps 1 to 9 of the **To create a Spark enabled workgroup in Athena** procedure in the previous section.

To specify your own workgroup configurations

1. If you want to create or use your own IAM role or configure notebook encryption, expand **IAM role configuration**.
   - For **Service Role to authorize Athena**, choose one of the following:
     - **Create and use a new service role** – Choose this option to have Athena create a service role for you. To see the permissions the role grants, choose **View permission details**.
     - **Use an existing service role** – From the drop down menu, choose an existing role. The role that you choose must include the permissions in the first option. For more information about permissions for notebook-enabled workgroups, see Troubleshoot Spark-enabled workgroups.
   - For **Notebook and calculation code encryption key management**, choose one of the following options:
     - **Encrypt using AWS owned key (Default)** – The AWS KMS key is owned and managed by AWS. You are not charged an additional fee for using this key.
     - **Encrypt using your own AWS KMS key** – For this option, do one of the following:
       - To use an existing key, use the search box to choose an AWS KMS or enter a key ARN.
       - To create a key in the AWS KMS console, choose **Create an AWS KMS key**. Your execution role must have permission to use the key that you create. After you finish creating the key in the KMS console, return to the **Create workgroup** page in Athena

console, and then use the **Choose an AWS KMS key or enter an ARN** search box to choose the key that you just created.

> **Important**
>
> When you change the AWS KMS key for a workgroup, notebooks managed before the update still reference the old KMS key. Notebooks managed after the update use the new KMS key. To update the old notebooks to reference the new KMS key, export and then import each of the old notebooks. If you delete the old KMS key before you update the old notebook references to the new KMS key, the old notebooks are no longer decryptable and cannot be recovered.
> This behavior also applies for updates to aliases, which are friendly names for KMS keys. When you update a KMS key alias to point to a new KMS key, notebooks managed before the alias update still reference the old KMS key, and notebooks managed after the alias update use the new KMS key. Consider these points before updating your KMS keys or aliases.

2. (Optional) **Other settings** – Expand this option to enable or disable the **Publish CloudWatch metrics** option for the workgroup. This field is selected by default. For more information, see Monitor Apache Spark calculations with CloudWatch metrics.
3. (Optional) **Tags** – Use this option to add tags to your workgroup. For more information, see Tag Athena resources.
4. Choose **Create workgroup**. A message informs you that the workgroup was created successfully, and the workgroup shows in the list of workgroups.

Step 2: Open notebook explorer and switch workgroups

Before you can use the Spark enabled workgroup that you just created, you must switch to the workgroup. To switch Spark enabled workgroups, you can use the **Workgroup** option in Notebook explorer or Notebook editor.

> **Note**
>
> Before you start, check that your browser does not block third-party cookies. Any browser that blocks third party cookies either by default or as a user-enabled setting will prevent notebooks from launching. For more information on managing cookies, see:

- Chrome
- Firefox
- Safari

To open notebook explorer and switch workgroups

1. In the navigation pane,choose **Notebook explorer**.
2. Use the **Workgroup** option on the upper right of the console to choose the Spark enabled workgroup that you created. The example notebook is shown in the list of notebooks.

   You can use the notebook explorer in the following ways:

   - Choose the linked name of a notebook to open the notebook in a new session.
   - To rename, delete, or export your notebook, use the **Actions** menu.
   - To import a notebook file, choose **Import file**.
   - To create a notebook, choose **Create notebook**.

Step 3: Run the example notebook

The sample notebook queries data from a publicly available New York City taxi trip dataset. The notebook has examples that show how to work with Spark DataFrames, Spark SQL, and the AWS Glue Data Catalog.

To run the example notebook

1. In Notebook explorer, choose the linked name of the example notebook.

   This starts a notebook session with the default parameters and opens the notebook in the notebook editor. A message informs you that a new Apache Spark session has been started using the default parameters (20 maximum DPUs).
2. To run the cells in order and observe the results, choose the **Run** button once for each cell in the notebook.

   - Scroll down to see the results and bring new cells into view.

- For the cells that have a calculation, a progress bar shows the percentage completed, the time elapsed, and the time remaining.
- The example notebook creates a sample database and table in your account. The final cell removes these as a clean-up step.

> **Note**
>
> If you change folder, table, or database names in the example notebook, make sure those changes are reflected in the IAM roles that you use. Otherwise, the notebook can fail to run due to insufficient permissions.

Step 4: Edit session details

After you start a notebook session, you can edit session details like table format, encryption, session idle timeout, and the maximum concurrent number of data processing units (DPUs) that you want to use. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory.

To edit session details

1. In the notebook editor, from the **Session** menu on the upper right, choose **Edit session**.
2. In the **Edit session details** dialog box, in the **Spark properties** section, choose or enter values for the following options:

   - **Additional table format** – Choose **Linux Foundation Delta Lake**, **Apache Hudi**, **Apache Iceberg**, or **Custom**.
     - For the **Delta**, **Hudi**, or **Iceberg** table options, the required table properties for the corresponding table format are automatically provided for you in the **Edit in table** and **Edit in JSON** options. For more information about using these table formats, see Use non-Hive table formats in Athena for Spark.
     - To add or remove table properties for the **Custom** or other table types, use the **Edit in table** and **Edit in JSON** options.
     - For the **Edit in table** option, choose **Add property** to add a property, or **Remove** to remove a property. To enter property names and their values, use the **Key** and **Value** boxes.

    - For the **Edit in JSON** option, use the JSON text editor to edit the configuration directly.
        - To copy the JSON text to the clipboard, choose **Copy**.
        - To remove all text from the JSON editor, choose **Clear**.
        - To configure line wrapping or choose a color theme for the JSON editor, choose the settings (gear) icon.
  - **Turn on Spark encryption -** – Select this option to encrypt data that is written to disk and sent through Spark network nodes. For more information, see Enable Apache Spark encryption.

3. In the **Session parameters** section, choose or enter values for the following options:

   - **Session idle timeout** - Choose or enter a value between 1 and 480 minutes. The default is 20.
   - **Coordinator size** - A *coordinator* is a special executor that orchestrates processing work and manages other executors in a notebook session. Currently, 1 DPU is the default and only possible value.
   - **Executor size** - An *executor* is the smallest unit of compute that a notebook session can request from Athena. Currently, 1 DPU is the default and only possible value.
   - **Max concurrent value** - The maximum number of DPUs that can run concurrently. The default is 20, the minimum is 3, and the maximum is 60. Increasing this value does not automatically allocate additional resources, but Athena will attempt to allocate up to the maximum specified when the compute load requires it and when resources are available.

4. Choose **Save**.
5. At the **Confirm edit** prompt, choose **Confirm**.

   Athena saves your notebook and starts a new session with the parameters that you specified. A banner in the notebook editor informs you that a new session has started with the modified parameters.

   > **Note**
   >
   > Athena remembers your session settings for the notebook. If you edit a session's parameters and then terminate the session, Athena uses the session parameters that you configured the next time you start a session for the notebook.

Step 5: View session and calculation details

After you run the notebook, you can view your session and calculation details.

To view session and calculation details

1. From the **Session** menu on the upper right, choose **View details**.

   - The **Current session** tab shows information about the current session, including session ID, creation time, status, and workgroup.
   - The **History** tab lists the session IDs for previous sessions. To view details for a previous session, choose the **History** tab, and then choose a session ID in the list.
   - The **Calculations** section shows a list of calculations that ran in the session.
2. To view the details for a calculation, choose the calculation ID.
3. On the **Calculation details** page, you can do the following:

   - To view the code for the calculation, see the **Code** section.
   - To see the results for the calculation, choose the **Results** tab.
   - To download the results that you see in text format, choose **Download results**.
   - To view information about the calculation results in Amazon S3, choose **View in S3**.

Step 6: Terminate the session

To end the notebook session

1. In the notebook editor, from the **Session** menu on the upper right, choose **Terminate**.
2. At the **Confirm session termination** prompt, choose **Confirm**. Your notebook is saved and you are returned to the notebook editor.

> **Note**
>
> Closing a notebook tab in the notebook editor does not by itself terminate the session for an active notebook. If you want to ensure that the session is terminated, use the **Session**, **Terminate** option.

Step 7: Create your own notebook

After you have created a Spark enabled Athena workgroup, you can create your own notebook.

To create a notebook

1. If the console navigation pane is not visible, choose the expansion menu on the left.
2. In the Athena console navigation pane, choose **Notebook explorer** or **Notebook editor**.
3. Do one of the following:
   - In **Notebook explorer**, choose **Create notebook**.
   - In **Notebook editor**, choose **Create notebook**, or choose the plus icon (**+**) to add a notebook.
4. In the **Create notebook** dialog box, for **Notebook name**, enter a name.
5. (Optional) Expand **Spark properties**, and then choose or enter values for the following options:
   - **Additional table format** – Choose **Linux Foundation Delta Lake**, **Apache Hudi**, **Apache Iceberg**, or **Custom**.
     - For the **Delta**, **Hudi**, or **Iceberg** table options, the required table properties for the corresponding table format are automatically provided for you in the **Edit in table** and **Edit in JSON** options. For more information about using these table formats, see Use non-Hive table formats in Athena for Spark.
     - To add or remove table properties for the **Custom** or other table types, use the **Edit in table** and **Edit in JSON** options.
     - For the **Edit in table** option, choose **Add property** to add a property, or **Remove** to remove a property. To enter property names and their values, use the **Key** and **Value** boxes.
     - For the **Edit in JSON** option, use the JSON text editor to edit the configuration directly.
       - To copy the JSON text to the clipboard, choose **Copy**.
       - To remove all text from the JSON editor, choose **Clear**.
       - To configure line wrapping or choose a color theme for the JSON editor, choose the settings (gear) icon.
   - **Turn on Spark encryption -** – Select this option to encrypt data that is written to disk and sent through Spark network nodes. For more information, see Enable Apache Spark encryption.

6. (Optional) Expand **Session parameters**, and then choose or enter values for the following options:

   - **Session idle timeout** - choose or enter a value between 1 and 480 minutes. The default is 20.
   - **Coordinator size** - A *coordinator* is a special executor that orchestrates processing work and manages other executors in a notebook session. Currently, 1 DPU is the default and only possible value. A DPU (data processing unit) is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory.
   - **Executor size** - An *executor* is the smallest unit of compute that a notebook session can request from Athena. Currently, 1 DPU is the default and only possible value.
   - **Max concurrent value** - The maximum number of DPUs that can run concurrently. The default is 20 and the maximum is 60. Increasing this value does not automatically allocate additional resources, but Athena will attempt to allocate up to the maximum specified when the compute load requires it and when resources are available.

7. Choose **Create**. Your notebook opens in a new session in the notebook editor.

For information about managing your notebook files, see [Manage notebook files](#).

Manage notebook files

Besides using the notebook explorer to [create](#) notebooks, you can also use it to open, rename, delete, export, or import notebooks, or view the session history for a notebook.

To open a previously created notebook

1. If the console navigation pane is not visible, choose the expansion menu on the left.
2. In the Athena console navigation pane, choose **Notebook editor** or **Notebook explorer**.
3. Do one of the following:

   - In **Notebook editor**, choose a notebook in the **Recent notebooks** or **Saved notebooks** list. The notebook opens in a new session.
   - In **Notebook explorer**, choose the name of a notebook in the list. The notebook opens in a new session.

To rename a notebook

1. Terminate any active sessions for the notebook that you want to rename. The notebook's active sessions must be terminated before you can rename the notebook.
2. Open **Notebook explorer**.
3. In the **Notebooks** list, select the option button for the notebook that you want to rename.
4. From the **Actions** menu, choose **Rename**.
5. At the **Rename notebook** prompt, enter the new name, and then choose **Save**. The new notebook name appears in the list of notebooks.

To delete a notebook

1. Terminate any active sessions for the notebook that you want to delete. The notebook's active sessions must be terminated before you can delete the notebook.
2. Open **Notebook explorer**.
3. In the **Notebooks** list, select the option button for the notebook that you want to delete.
4. From the **Actions** menu, choose **Delete**.
5. At the **Delete notebook?** prompt, enter the name of the notebook, and then choose **Delete** to confirm the deletion. The notebook name is removed from the list of notebooks.

To export a notebook

1. Open **Notebook explorer**.
2. In the **Notebooks** list, select the option button for the notebook that you want to export.
3. From the **Actions** menu, choose **Export file**.

To import a notebook

1. Open **Notebook explorer**.
2. Choose **Import file**.
3. Browse to the location on your local computer of the file that you want to import, and then choose **Open**. The imported notebook appears in the list of notebooks.

To view the session history for a notebook

1. Open **Notebook explorer**.
2. In the **Notebooks** list, select the option button for the notebook whose session history you want to view.
3. From the **Actions** menu, choose **Session history**.
4. On the **History** tab, choose a **Session ID** to view information about the session and its calculations.

Use the Athena notebook editor

You manage your notebooks in the Athena notebook explorer and edit and run them in sessions using the Athena notebook editor. You can configure DPU usage for your notebook sessions according to your requirements.

When you stop a notebook, you terminate the associated session. All files are saved, but changes underway in declared variables, functions and classes are lost. When you restart the notebook, Athena reloads the notebook files and you can run your code again.

The Athena notebook editor is an interactive environment for writing and running code. The following sections describe the features of the environment.

Understand notebook sessions and calculations

Each notebook is associated with a single Python kernel and runs Python code. A notebook can have one or more cells that contain commands. To run the cells in a notebook, you first create a session for the notebook. Sessions keep track of the variables and state of notebooks.

Running a cell in a notebook means running a *calculation* in the current session. Calculations progress the state of the notebook and may perform tasks like reading from Amazon S3 or writing to other data stores. As long as a session is running, calculations use and modify the state that is maintained for the notebook.

When you no longer need the state, you can end a session. When you end a session, the notebook remains, but the variables and other state information are destroyed. If you need to work on multiple projects at the same time, you can create a session for each project, and the sessions will be independent from each other.

Sessions have dedicated compute capacity, measured in DPU. When you create a session, you can assign the session a number of DPUs. Different sessions can have different capacities depending on the requirements of the task.

Switch between command mode and edit mode

The notebook editor has a modal user interface: an *edit mode* for entering text into a cell, and a *command mode* for issuing commands to the editor itself like copy, paste, or run.

To use edit mode and command mode, you can perform the following tasks:

- To enter edit mode, press **`ENTER`**, or choose a cell. When a cell is in edit mode, the cell has a green left margin.
- To enter command mode, press **`ESC`**, or click outside of a cell. Note that commands typically apply only to the currently selected cell, not to all cells. When the editor is in command mode, the cell has a blue left margin.
- In command mode, you can use keyboard shortcuts and the menu above the editor, but not enter text into individual cells.
- To select a cell, choose the cell.
- To select all cells, press **`Ctrl+A`** (Windows) or **`Cmd+A`** (Mac).

Use actions in the notebook editor menu

The icons in the menu at the top of the notebook editor offer the following options:

- **Save** – Saves the current state of the notebook.
- **Insert cell below** – Adds a new (empty) cell below the currently selected one.
- **Cut selected cells** – Removes the selected cell from its current location and copies the cell to memory.
- **Copy selected cells** – Copies the selected cell to memory.
- **Paste cells below** – Pastes the copied cell below the current cell.
- **Move selected cells up** – Moves the current cell above the cell above.
- **Move selected cells down** – Moves the current cell below the cell below.
- **Run** – Runs the current (selected) cell. The output displays immediately below the current cell.
- **Run all** – Runs all cells in the notebook. The output for each cell displays immediately below the cell.

- **Stop (Interrupt the kernel)** – Stops the current notebook by interrupting the kernel.
- **Format option** – Selects the cell format, which can be one of the following:
  - **Code** – Use for Python code (the default).
  - **Markdown** – Use for entering text in GitHub-style markdown format. To render the markdown, run the cell.
  - **Raw NBConvert** – Use for entering text in unmodified form. Cells marked as **Raw NBConvert** can be converted into a different format like HTML by the Jupyter nbconvert command line tool.
- **Heading** – Use to change the heading level of the cell.
- **Command palette** – Contains Jupyter notebook commands and their keyboard shortcuts. For more information about the keyboard shortcuts, see the sections later in this document.
- **Session** – Use options in this menu to view the details for a session, edit session parameters, or terminate the session.

Use command mode keyboard shortcuts for productivity

The following are some common notebook editor command mode keyboard shortcuts. These shortcuts are available after pressing **ESC** to enter command mode. To see a full list of commands available in the editor, press **ESC + H**.

Key	Action
1 - 6	Change the cell type to markdown and set the heading level to the number typed
a	Create a cell above the current cell
b	Create a cell below the current cell
c	Copy the current cell to memory
d d	Delete the current cell
h	Display the keyboard shortcut help screen
j	Go one cell down

Key	Action
`k`	Go one cell up
`m`	Change the current cell format to markdown
`r`	Change the current cell format to raw
`s`	Save the notebook
`v`	Paste memory contents under the current cell
`x`	Cut the selected cell or cells
`y`	Change the cell format to code
`z`	Undo
`Ctrl+Enter`	Run the current cell and enter command mode
`Shift+Enter` or `Alt+Enter`	Run the current cell and create a new cell below the output, and enter the new cell in edit mode
`Space`	Page down
`Shift+Space`	Page up
`Shift + L`	Toggle the visibility of line numbers in cells

Customize command mode shortcuts

The notebook editor has an option to customize command mode keyboard shortcuts.

To edit command mode shortcuts

1. From the notebook editor menu, the choose the **Command palette**.
2. From the command palette, choose the **Edit command mode keyboard shortcuts** command.

3. Use the **Edit command mode shortcuts** interface to map or remap commands that you want to the keyboard.

   To see instructions for editing command mode shortcuts, scroll to the bottom of the **Edit command mode shortcuts** screen.

For information about using magic commands in Athena for Apache Spark, see Use magic commands.

Topics

Use magic commands

Magic commands, or *magics*, are special commands that you can run in a notebook cell. For example, `%env` shows the environment variables in a notebook session. Athena supports the magic functions in IPython 6.0.3.

This section shows some key magic commands in Athena for Apache Spark.

- To see a list of magic commands in Athena, run the command **`%lsmagic`** in a notebook cell.
- For information about using magics to create graphs in Athena notebooks, see Use magics to create data graphs.
- For information about additional magic commands, see Built-in magic commands in the IPython documentation.

> **Note**
>
> Currently, the `%pip` command fails when executed. This is a known issue.

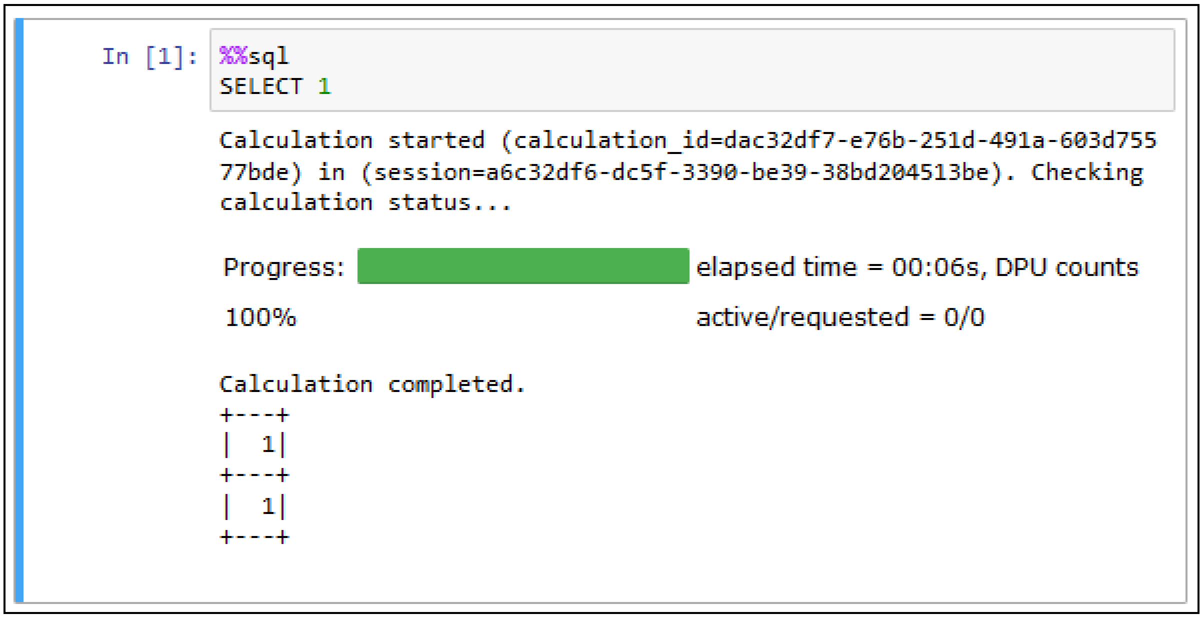
Topics

Use cell magics

Magics that are written on several lines are preceded by a double percent sign (%%) and are called *cell magic functions* or *cell magics*.

%%sql

This cell magic allows to run SQL statements directly without having to decorate it with Spark SQL statement. The command also displays the output by implicitly calling `.show()` on the returned dataframe.



The `%%sql` command auto truncates column outputs to a width of 20 characters. Currently, this setting is not configurable. To work around this limitation, use the following full syntax and modify the parameters of the `show` method accordingly.

```
spark.sql("""YOUR_SQL""").show(n=number, truncate=number, vertical=bool)
```

- **n** `int`, optional. The number of rows to show.
- **truncate –** `bool` or `int`, optional – If `true`, truncates strings longer than 20 characters. When set to a number greater than 1, truncates long strings to the length specified and right aligns cells.
- **vertical –** `bool`, optional. If `true`, prints output rows vertically (one line per column value).

Use line magics

Magics that are on a single line are preceded by a percent sign (%) and are called *line magic functions* or *line magics*.

%help

Displays descriptions of the available magic commands.

```
In [6]: %help

Available Magic Commands:
Magic | Input | Description
%session_id  | None | Return the session ID for the running session.
%status | None | Describes the current session and display SessionID, State,
WorkGroup, EngineVersion and StartTime
%help | None | Displays list of supported magics
%set_log_level | String | Sets the current log level to the provided log leve
ls (ERROR|INFO|WARNING etc)
%list_sessions | None | Lists the most recent sessions associated with the cu
rrent workgroup
%%sql | String | Run an SQL command against SparkSQL.
```

%list_sessions

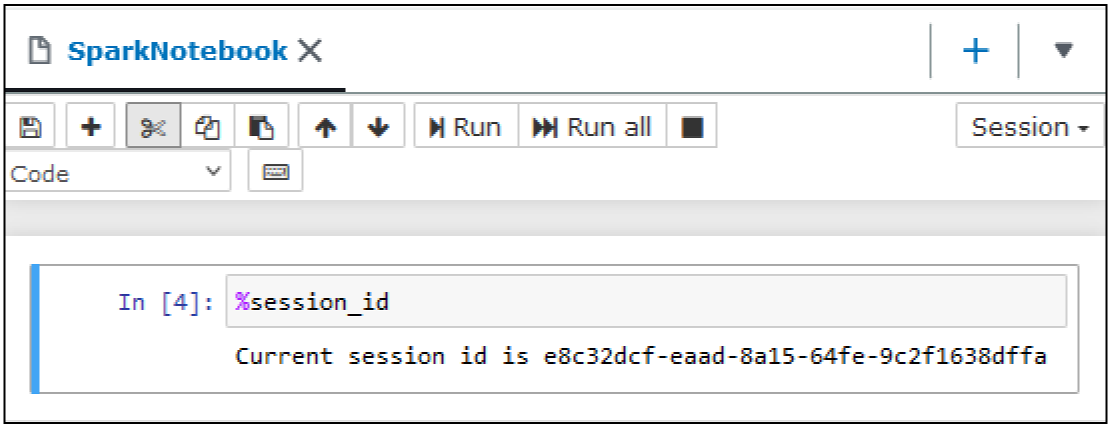
Lists the sessions associated with the notebook. The information for each session includes the session ID, session status, and the date and time that the session started and ended.

```
In [12]: %list_sessions
```

SessionId	Status	StartDateTime	EndDateTime
66c32de7-78b9-f2ee-6eb9-d8d9716c6ac8	IDLE	02/16/2023, 19:58:54	
ccc32dda-6dea-6277-d434-5c5da5e1a882	TERMINATED	02/16/2023, 19:30:24	02/16/2023, 19:51:53
e8c32dcf-eaad-8a15-64fe-9c2f1638dffa	TERMINATED	02/16/2023, 19:07:26	02/16/2023, 19:28:53

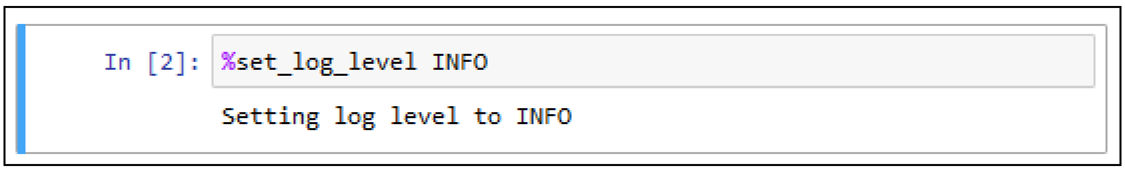
%session_id

Retrieves the current session ID.



%set_log_level

Sets or resets the logger to use the specified log level. Possible values are `DEBUG`, `ERROR`, `FATAL`,`INFO`, and `WARN` or `WARNING`. Values must be uppercase and must not be enclosed in single or double quotes.



%status

Describes the current session. The output includes the session ID, session state, workgroup name, PySpark engine version, and session start time. This magic command requires an active session to retrieve session details.

Following are the possible values for status:

CREATING – The session is being started, including acquiring resources.

CREATED – The session has been started.

IDLE – The session is able to accept a calculation.

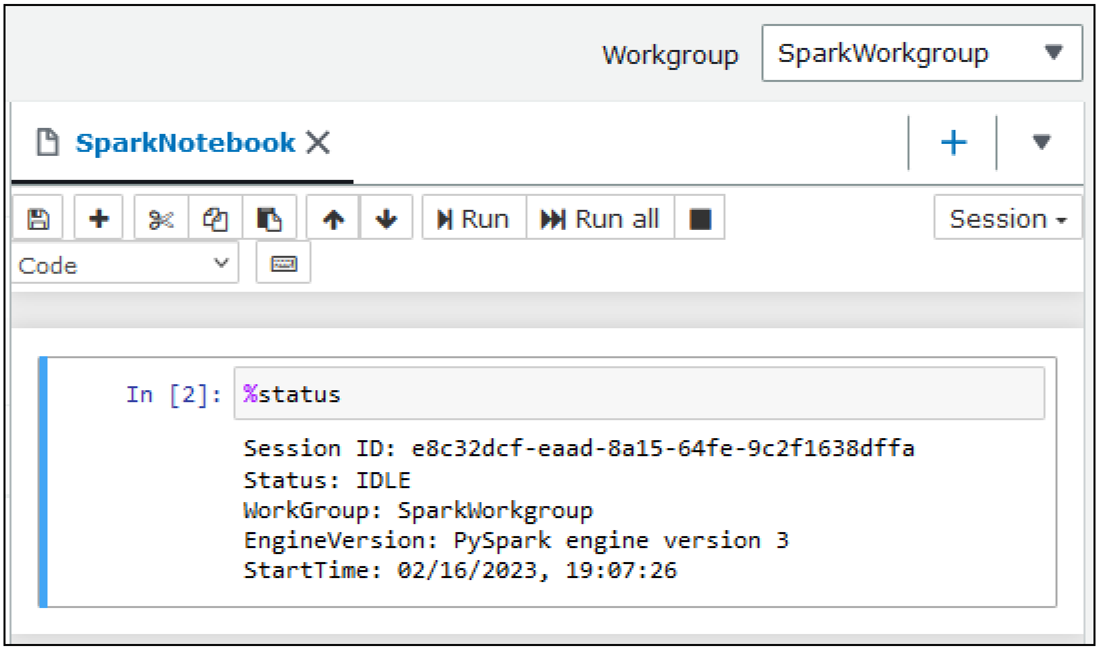
BUSY – The session is processing another task and is unable to accept a calculation.

TERMINATING – The session is in the process of shutting down.

TERMINATED – The session and its resources are no longer running.

DEGRADED – The session has no healthy coordinators.

FAILED – Due to a failure, the session and its resources are no longer running.



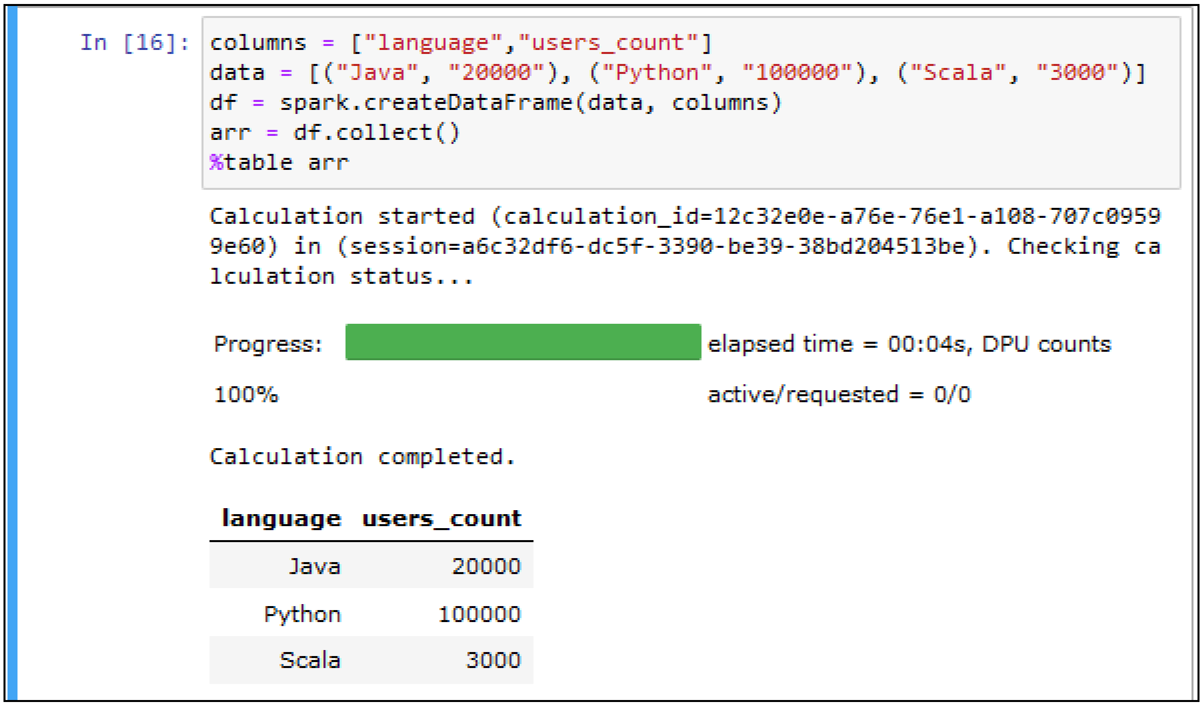
Use magics to create data graphs

The line magics in this section specialize in rendering data for particular types of data or in conjunction with graphing libraries.

%table

You can use the `%table` magic command to display dataframe data in table format.

The following example creates a dataframe with two columns and three rows of data, then displays the data in table format.



%matplot

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. You can use the `%matplot` magic command to create a graph after you import the matplotlib library into a notebook cell.

The following example imports the matplotlib library, creates a set of x and y coordinates, and then uses the use the `%matplot` magic command to create a graph of the points.

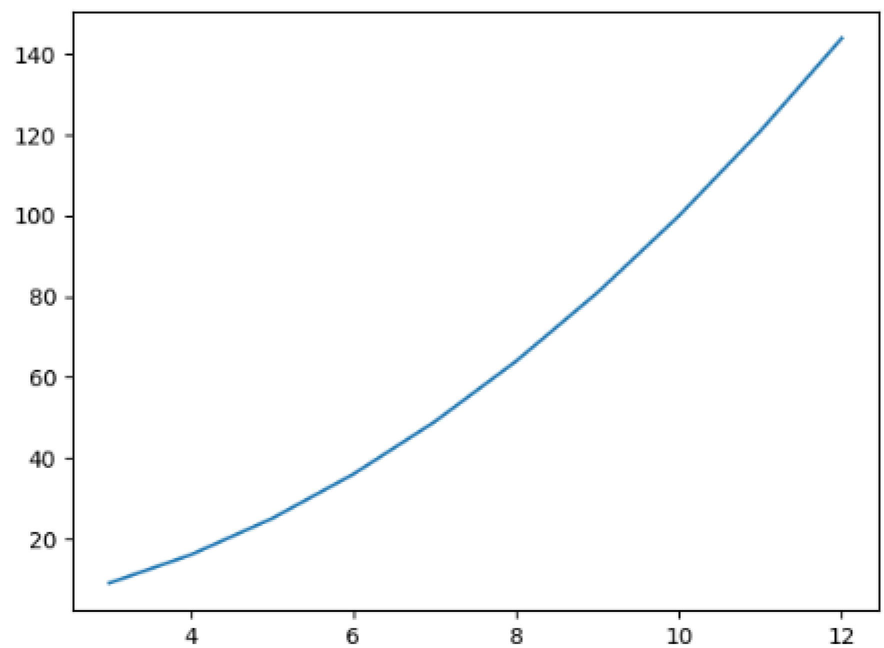
```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

In [12]:
```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

```
Calculation started (calculation_id=5ac32e04-81b6-9ee7-ce55-539ee2ce383e) in (session=a6c32df6-dc5f-3390-be39-38bd204513be). Checking calculation status...
```

Progress: 100% elapsed time = 00:02s

```
Calculation completed.
```




```
[<matplotlib.lines.Line2D object at 0x7f6e29e580>]
```

Use the matplotlib and seaborn libraries together

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas (Python data analysis) data structures. You can also use the `%matplot` magic command to render seaborn data.

The following example uses both the matplotlib and seaborn libraries to create a simple bar graph.

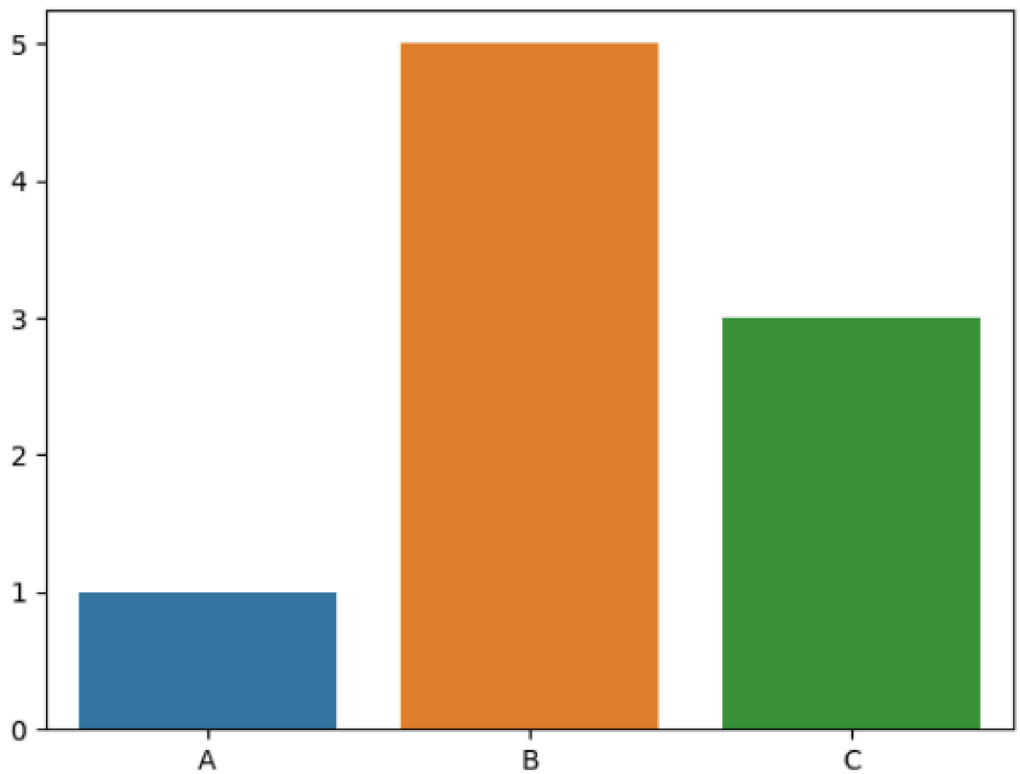
```
import matplotlib.pyplot as plt
import seaborn as sns

x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns


x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

```
Calculation started (calculation_id=08c32e1b-233b-4a72-6571-1ae7a28a7b78)
in (session=64c32e1a-f45e-1d52-b54b-85202e2a9233). Checking calculation st
atus...
```

Progress: 100% elapsed time = 00:04s

```
Calculation completed.
```

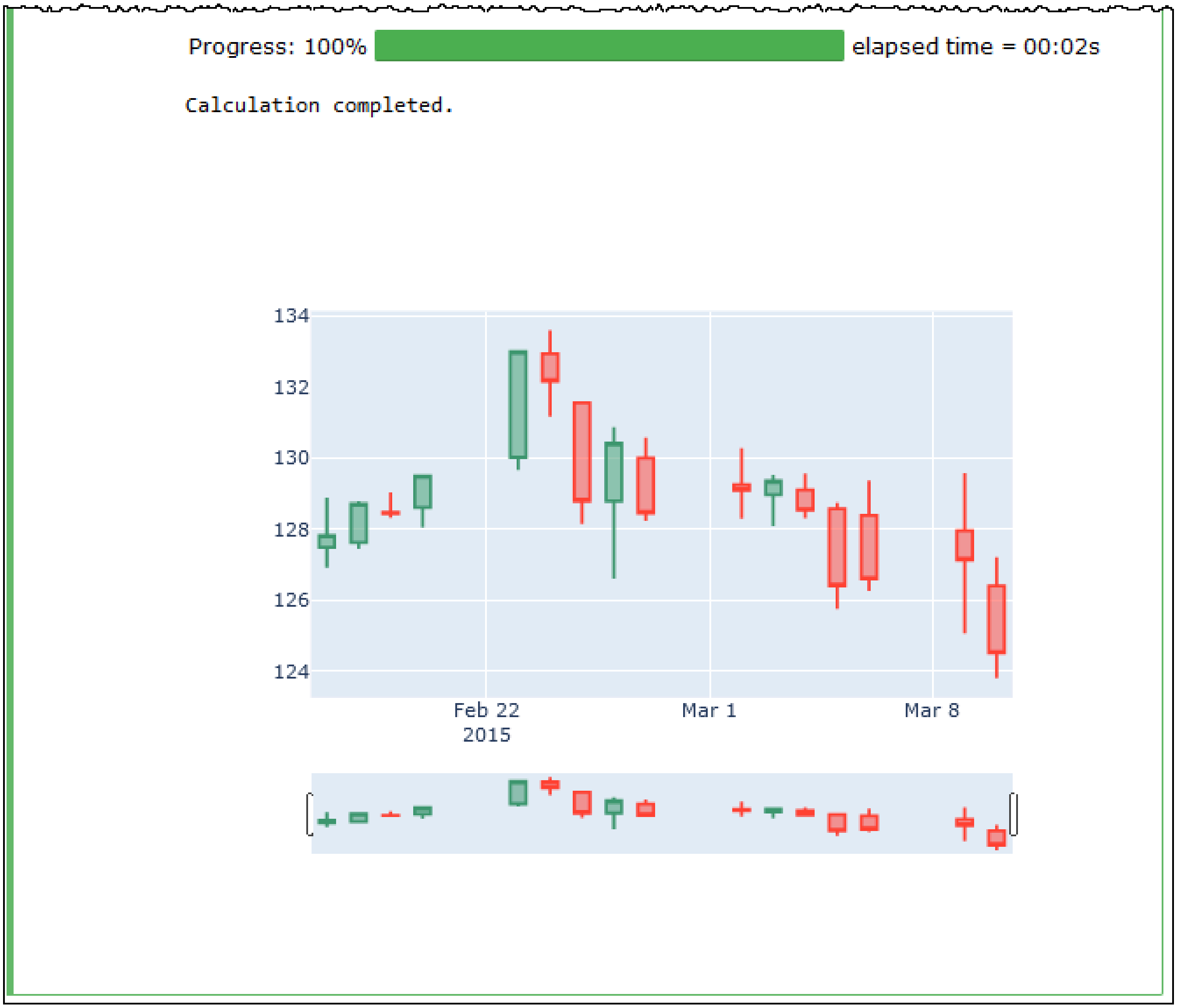


%plotly

Plotly is an open source graphing library for Python that you can use to make interactive graphs. You use the `%ploty` magic command to render ploty data.

The following example uses the StringIO, plotly, and pandas libraries on stock price data to create a graph of stock activity from February and March of 2015.

```
from io import StringIO
csvString = """
Date,AAPL.Open,AAPL.High,AAPL.Low,AAPL.Close,AAPL.Volume,AAPL.Adjusted,dn,mavg,up,direction
2015-02-17,127.489998,128.880005,126.919998,127.830002,63152400,122.905254,106.7410523,117.9276
2015-02-18,127.629997,128.779999,127.449997,128.720001,44891700,123.760965,107.842423,118.94033
2015-02-19,128.479996,129.029999,128.330002,128.449997,37362400,123.501363,108.8942449,119.8891
2015-02-20,128.619995,129.5,128.050003,129.5,48948400,124.510914,109.7854494,120.7635001,131.74
2015-02-23,130.020004,133,129.660004,133,70974100,127.876074,110.3725162,121.7201668,133.067817
2015-02-24,132.940002,133.600006,131.169998,132.169998,69228100,127.078049,111.0948689,122.6648
2015-02-25,131.559998,131.600006,128.149994,128.789993,74711700,123.828261,113.2119183,123.6296
2015-02-26,128.789993,130.869995,126.610001,130.419998,91287500,125.395469,114.1652991,124.2823
2015-02-27,130,130.570007,128.240005,128.460007,62014800,123.510987,114.9668484,124.8426669,134
2015-03-02,129.25,130.279999,128.300003,129.089996,48096700,124.116706,115.8770904,125.4036668,
2015-03-03,128.960007,129.520004,128.089996,129.360001,37816300,124.376308,116.9535132,125.9551
2015-03-04,129.100006,129.559998,128.320007,128.539993,31666300,123.587892,118.0874253,126.4730
2015-03-05,128.580002,128.75,125.760002,126.410004,56517100,121.539962,119.1048311,126.848667,1
2015-03-06,128.399994,129.369995,126.260002,126.599998,72842100,121.722637,120.190797,127.22883
2015-03-09,127.959999,129.570007,125.059998,127.139999,88528500,122.241834,121.6289771,127.6311
2015-03-10,126.410004,127.220001,123.800003,124.510002,68856600,119.71316,123.1164763,127.92350
"""
csvStringIO = StringIO(csvString)

from io import StringIO
import plotly.graph_objects as go
import pandas as pd
from datetime import datetime
df = pd.read_csv(csvStringIO)
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
open=df['AAPL.Open'],
high=df['AAPL.High'],
low=df['AAPL.Low'],
close=df['AAPL.Close'])])
%plotly fig
```



Use non-Hive table formats in Athena for Spark

When you work with sessions and notebooks in Athena for Spark, you can use Linux Foundation Delta Lake, Apache Hudi, and Apache Iceberg tables, in addition to Apache Hive tables.

Considerations and limitations

When you use table formats other than Apache Hive with Athena for Spark, consider the following points:

- In addition to Apache Hive, only one table format is supported per notebook. To use multiple table formats in Athena for Spark, create a separate notebook for each table format. For information about creating notebooks in Athena for Spark, see Step 7: Create your own notebook.

- The Delta Lake, Hudi, and Iceberg table formats have been tested on Athena for Spark by using AWS Glue as the metastore. You might be able to use other metastores, but such usage is not currently supported.
- To use the additional table formats, override the default `spark_catalog` property, as indicated in the Athena console and in this documentation. These non-Hive catalogs can read Hive tables, in addition to their own table formats.

Table versions

The following table shows supported non-Hive table versions in Amazon Athena for Apache Spark.

Table format	Supported version
Apache Iceberg	1.2.1
Apache Hudi	0.13
Linux Foundation Delta Lake	2.0.2

In Athena for Spark, these table format `.jar` files and their dependencies are loaded onto the classpath for Spark drivers and executors.

For an *AWS Big Data Blog* post that shows how to work with Iceberg, Hudi, and Delta Lake table formats using Spark SQL in Amazon Athena notebooks, see Use Amazon Athena with Spark SQL for your open-source transactional table formats.

Topics

- Use Apache Iceberg tables in Athena for Spark
- Use Apache Hudi tables in Athena for Spark
- Use Linux Foundation Delta Lake tables in Athena for Spark

Use Apache Iceberg tables in Athena for Spark

Apache Iceberg is an open table format for large datasets in Amazon Simple Storage Service (Amazon S3). It provides you with fast query performance over large tables, atomic commits, concurrent writes, and SQL-compatible table evolution.

To use Apache Iceberg tables in Athena for Spark, configure the following Spark properties. These properties are configured for you by default in the Athena for Spark console when you choose Apache Iceberg as the table format. For steps, see Step 4: Edit session details or Step 7: Create your own notebook.

```
"spark.sql.catalog.spark_catalog": "org.apache.iceberg.spark.SparkSessionCatalog",
"spark.sql.catalog.spark_catalog.catalog-impl":
 "org.apache.iceberg.aws.glue.GlueCatalog",
"spark.sql.catalog.spark_catalog.io-impl": "org.apache.iceberg.aws.s3.S3FileIO",
"spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
```

The following procedure shows you how to use an Apache Iceberg table in an Athena for Spark notebook. Run each step in a new cell in the notebook.

To use an Apache Iceberg table in Athena for Spark

1. Define the constants to use in the notebook.

   ```
   DB_NAME = "NEW_DB_NAME"
   TABLE_NAME = "NEW_TABLE_NAME"
   TABLE_S3_LOCATION = "s3://amzn-s3-demo-bucket"
   ```

2. Create an Apache Spark DataFrame.

   ```
   columns = ["language","users_count"]
   data = [("Golang", 3000)]
   df = spark.createDataFrame(data, columns)
   ```

3. Create a database.

   ```
   spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
   ```

4. Create an empty Apache Iceberg table.

```
spark.sql("""
CREATE TABLE {}.{} (
language string,
users_count int
) USING ICEBERG
""".format(DB_NAME, TABLE_NAME))
```

5. Insert a row of data into the table.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. Confirm that you can query the new table.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

For more information and examples on working with Spark DataFrames and Iceberg tables, see Spark Queries in the Apache Iceberg documentation.

Use Apache Hudi tables in Athena for Spark

Apache Hudi is an open-source data management framework that simplifies incremental data processing. Record-level insert, update, upsert, and delete actions are processed with greater precision, which reduces overhead.

To use Apache Hudi tables in Athena for Spark, configure the following Spark properties. These properties are configured for you by default in the Athena for Spark console when you choose Apache Hudi as the table format. For steps, see Step 4: Edit session details or Step 7: Create your own notebook.

```
"spark.sql.catalog.spark_catalog": "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
"spark.serializer": "org.apache.spark.serializer.KryoSerializer",
"spark.sql.extensions": "org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
```

The following procedure shows you how to use an Apache Hudi table in an Athena for Spark notebook. Run each step in a new cell in the notebook.

To use an Apache Hudi table in Athena for Spark

1. Define the constants to use in the notebook.

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://amzn-s3-demo-bucket"
```

2. Create an Apache Spark DataFrame.

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. Create a database.

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. Create an empty Apache Hudi table.

```
spark.sql("""
CREATE TABLE {}.{} (
language string,
users_count int
) USING HUDI
TBLPROPERTIES (
primaryKey = 'language',
type = 'mor'
);
""".format(DB_NAME, TABLE_NAME))
```

5. Insert a row of data into the table.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME,TABLE_NAME))
```

6. Confirm that you can query the new table.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Use Linux Foundation Delta Lake tables in Athena for Spark

Linux Foundation Delta Lake is a table format that you can use for big data analytics. You can use Athena for Spark to read Delta Lake tables stored in Amazon S3 directly.

To use Delta Lake tables in Athena for Spark, configure the following Spark properties. These properties are configured for you by default in the Athena for Spark console when you choose Delta Lake as the table format. For steps, see Step 4: Edit session details or Step 7: Create your own notebook.

```
"spark.sql.catalog.spark_catalog" : "org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension"
```

The following procedure shows you how to use a Delta Lake table in an Athena for Spark notebook. Run each step in a new cell in the notebook.

To use a Delta Lake table in Athena for Spark

1. Define the constants to use in the notebook.

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://amzn-s3-demo-bucket"
```

2. Create an Apache Spark DataFrame.

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. Create a database.

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. Create an empty Delta Lake table.

```
spark.sql("""
CREATE TABLE {}.{} (
  language string,
  users_count int
) USING DELTA
```

```
""".format(DB_NAME, TABLE_NAME))
```

5. Insert a row of data into the table.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. Confirm that you can query the new table.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Use Python libraries in Athena for Spark

This page describes the terminology used and lifecycle management followed for the runtimes, libraries, and packages used in Amazon Athena for Apache Spark.

Definitions

- **Amazon Athena for Apache Spark** is a customized version of open source Apache Spark. To see the current version, run the command `print(f'{spark.version}')` in a notebook cell.
- The **Athena runtime** is the environment in which your code runs. The environment includes a Python interpreter and PySpark libraries.
- An **external library or package** is a Java or Scala JAR or Python library that is not part of the Athena runtime but can be included in Athena for Spark jobs. External packages can be built by Amazon or by you.
- A **convenience package** is a collection of external packages selected by Athena that you can choose to include in your Spark applications.
- A **bundle** combines the Athena runtime and a convenience package.
- A **user library** is an external library or package that you explicitly add to your Athena for Spark job.
  - A user library is an external package that is not part of a convenience package. A user library requires loading and installation, as when you write some `.py` files, zip them up, and then add the `.zip` file to your application.
- An **Athena for Spark application** is a job or query that you submit to Athena for Spark.

Lifecycle management

The following sections describe the versioning and deprecation policies regarding the runtime and convenience packages used in Athena for Spark.

Runtime versioning and deprecation

The main component in the Athena runtime is the Python interpreter. Because Python is an evolving language, new versions are released regularly and support removed for older versions. Athena does not recommend that you run programs with deprecated Python interpreter versions and highly recommends that you use the latest Athena runtime whenever possible.

The Athena runtime deprecation schedule is as follows:

1. After Athena provides a new runtime, Athena will continue to support the previous runtime for 6 months. During that time, Athena will apply security patches and updates to the previous runtime.
2. After 6 months, Athena will end support for the previous runtime. Athena will no longer apply security patches and other updates to the previous runtime. Spark applications using the previous runtime will no longer be eligible for technical support.
3. After 12 months, you will no longer be able to update or edit Spark applications in a workgroup that uses the previous runtime. We recommend that you update your Spark applications before this time period ends. After the time period ends, you can still run existing notebooks, but any notebooks that still use the previous runtime will log a warning to that effect.
4. After 18 months, you will no longer be able to run jobs in the workgroup using the previous runtime.

Convenience package versioning and deprecation

The contents of convenience packages change over time. Athena occasionally adds, removes, or upgrades these convenience packages.

Athena uses the following guidelines for convenience packages:

- Convenience packages have a simple versioning scheme such as 1, 2, 3.
- Each convenience package version includes specific versions of external packages. After Athena creates a convenience package, the convenience package's set of external packages and their corresponding versions do not change.

- Athena creates a new convenience package version when it includes a new external package, removes an external package, or upgrades the version of one or more external packages.

Athena deprecates a convenience package when it deprecates the Athena runtime that the package uses. Athena can deprecate packages sooner to limit the number of bundles that it supports.

The convenience package deprecation schedule follows the Athena runtime deprecation schedule.

List of preinstalled Python libraries

Preinstalled Python libraries include the following.

```
boto3==1.24.31
botocore==1.27.31
certifi==2022.6.15
charset-normalizer==2.1.0
cycler==0.11.0
cython==0.29.30
docutils==0.19
fonttools==4.34.4
idna==3.3
jmespath==1.0.1
joblib==1.1.0
kiwisolver==1.4.4
matplotlib==3.5.2
mpmath==1.2.1
numpy==1.23.1
packaging==21.3
pandas==1.4.3
patsy==0.5.2
pillow==9.2.0
plotly==5.9.0
pmdarima==1.8.5
pyathena==2.9.6
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
requests==2.28.1
s3transfer==0.6.0
scikit-learn==1.1.1
scipy==1.8.1
seaborn==0.11.2
```

```
six==1.16.0
statsmodels==0.13.2
sympy==1.10.1
tenacity==8.0.1
threadpoolctl==3.1.0
urllib3==1.26.10
pyarrow==9.0.0
```

Notes

- MLlib (Apache Spark machine learning library) and the `pyspark.ml` package are not supported.
- Currently, `pip install` is not supported in Athena for Spark sessions.

For information on importing Python libraries to Amazon Athena for Apache Spark, see Import files and Python libraries to Athena for Spark.

Import files and Python libraries to Athena for Spark

This document provides examples of how to import files and Python libraries to Amazon Athena for Apache Spark.

Considerations and Limitations

- **Python version** – Currently, Athena for Spark uses Python version 3.9.16. Note that Python packages are sensitive to minor Python versions.
- **Athena for Spark architecture** – Athena for Spark uses Amazon Linux 2 on ARM64 architecture. Note that some Python libraries do not distribute binaries for this architecture.
- **Binary shared objects (SOs)** – Because the SparkContext addPyFile method does not detect binary shared objects, it cannot be used in Athena for Spark to add Python packages that depend on shared objects.
- **Resilient Distributed Datasets (RDDs)** – RDDs are not supported.
- **Dataframe.foreach** – The PySpark DataFrame.foreach method is not supported.

Examples

The examples use the following conventions.

- The placeholder Amazon S3 location `s3://amzn-s3-demo-bucket`. Replace this with your own S3 bucket location.
- All code blocks that execute from a Unix shell are shown as *`directory_name`* `$`. For example, the command `ls` in the directory `/tmp` and its output are displayed as follows:

  ```
  /tmp $ ls
  ```

  Output

  ```
  file1 file2
  ```

Import text files for use in calculations

The examples in this section show how to import text files for use in calculations in your notebooks in Athena for Spark.

Add a file to a notebook after you write it to local temporary directory

The following example shows how to write a file to a local temporary directory, add it to a notebook, and test it.

```
import os
from pyspark import SparkFiles
tempdir = '/tmp/'
path = os.path.join(tempdir, "test.txt")
with open(path, "w") as testFile:
    _ = testFile.write("5")
sc.addFile(path)

def func(iterator):
    with open(SparkFiles.get("test.txt")) as testFile:
        fileVal = int(testFile.readline())
        return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)
df = spark.createDataFrame([(1, "a"), (2, "b")])
```

```
df.withColumn("col", udf_with_import(col('_2'))).show()
```

Output

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Import a file from Amazon S3

The following example shows how to import a file from Amazon S3 into a notebook and test it.

To import a file from Amazon S3 into a notebook

1. Create a file named `test.txt` that has a single line that contains the value 5.
2. Add the file to a bucket in Amazon S3. This example uses the location `s3://amzn-s3-demo-bucket`.
3. Use the following code to import the file to your notebook and test the file.

```
from pyspark import SparkFiles
sc.addFile('s3://amzn-s3-demo-bucket/test.txt')

def func(iterator):
   with open(SparkFiles.get("test.txt")) as testFile:
       fileVal = int(testFile.readline())
       return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)
df = spark.createDataFrame([(1, "a"), (2, "b")])
df.withColumn("col", udf_with_import(col('_2'))).show()
```

Output

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Add Python files

The examples in this section show how to add Python files and libraries to your Spark notebooks in Athena.

Add Python files and register a UDF

The following example shows how to add Python files from Amazon S3 to your notebook and register a UDF.

To add Python files to your notebook and register a UDF

1. Using your own Amazon S3 location, create the file `s3://amzn-s3-demo-bucket/file1.py` with the following contents:

```
def xyz(input):
    return 'xyz  - udf ' + str(input);
```

2. In the same S3 location, create the file `s3://amzn-s3-demo-bucket/file2.py` with the following contents:

```
from file1 import xyz
def uvw(input):
    return 'uvw -> ' + xyz(input);
```

3. In your Athena for Spark notebook, run the following commands.

```
sc.addPyFile('s3://amzn-s3-demo-bucket/file1.py')
sc.addPyFile('s3://amzn-s3-demo-bucket/file2.py')

def func(iterator):
    from file2 import uvw
```

```
    return [uvw(x) for x in iterator]

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", udf_with_import(col('_2'))).show(10)
```

Output

```
Calculation started (calculation_id=1ec09e01-3dec-a096-00ea-57289cdb8ce7) in
 (session=c8c09e00-6f20-41e5-98bd-4024913d6cee). Checking calculation status...
Calculation completed.
+---+---+--------------------+
| _1| _2|                 col|
+---+---+--------------------+
| 1 |  a|[uvw -> xyz - ud... |
| 2 |  b|[uvw -> xyz - ud... |
+---+---+--------------------+
```

Import a Python .zip file

You can use the Python `addPyFile` and `import` methods to import a Python .zip file to your notebook.

> **Note**
>
> The `.zip` files that you import to Athena Spark may include only Python packages. For example, including packages with C-based files is not supported.

To import a Python `.zip` file to your notebook

1. On your local computer, in a desktop directory such as `\tmp`, create a directory called `moduletest`.
2. In the `moduletest` directory, create a file named `hello.py` with the following contents:

```
def hi(input):
    return 'hi ' + str(input);
```

3. In the same directory, add an empty file with the name `__init__.py`.

   If you list the directory contents, they should now look like the following.

```
/tmp $ ls moduletest
__init__.py       hello.py
```

4. Use the `zip` command to place the two module files into a file called `moduletest.zip`.

```
moduletest $ zip -r9 ../moduletest.zip *
```

5. Upload the `.zip` file to your bucket in Amazon S3.

6. Use the following code to import the Python `.zip` file into your notebook.

```
sc.addPyFile('s3://amzn-s3-demo-bucket/moduletest.zip')

from moduletest.hello import hi

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(hi)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

Output

```
Calculation started (calculation_id=6ec09e8c-6fe0-4547-5f1b-6b01adb2242c) in
 (session=dcc09e8c-3f80-9cdc-bfc5-7effa1686b76). Checking calculation status...
Calculation completed.
+---+---+----+
| _1| _2| col|
+---+---+----+
|  1|  a|hi a|
|  2|  b|hi b|
```

```
+---+---+----+
```

Import two versions of a Python library as separate modules

The following code examples show how to add and import two different versions of a Python library from a location in Amazon S3 as two separate modules. The code adds each the library file from S3, imports it, and then prints the library version to verify the import.

```
sc.addPyFile('s3://amzn-s3-demo-bucket/python-third-party-libs-test/
simplejson_v3_15.zip')
sc.addPyFile('s3://amzn-s3-demo-bucket/python-third-party-libs-test/
simplejson_v3_17_6.zip')

import simplejson_v3_15
print(simplejson_v3_15.__version__)
```

Output

```
3.15.0
```

```
import simplejson_v3_17_6
print(simplejson_v3_17_6.__version__)
```

Output

```
3.17.6
```

Import a Python .zip file from PyPI

This example uses the `pip` command to download a Python .zip file of the [bpabel/piglatin](https://github.com/bpabel/piglatin) project from the [Python Package Index (PyPI)](https://pypi.org/).

To import a Python .zip file from PyPI

1. On your local desktop, use the following commands to create a directory called `testpiglatin` and create a virtual environment.

   ```
   /tmp $ mkdir testpiglatin
   /tmp $ cd testpiglatin
   ```

```
testpiglatin $ virtualenv .
```

Output

```
created virtual environment CPython3.9.6.final.0-64 in 410ms
creator CPython3Posix(dest=/private/tmp/testpiglatin, clear=False,
 no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle,
 via=copy, app_data_dir=/Users/user1/Library/Application Support/virtualenv)
added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
activators
 BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonAct
```

2. Create a subdirectory named `unpacked` to hold the project.

```
testpiglatin $ mkdir unpacked
```

3. Use the `pip` command to install the project into the `unpacked` directory.

```
testpiglatin $ bin/pip install -t $PWD/unpacked piglatin
```

Output

```
Collecting piglatin
Using cached piglatin-1.0.6-py2.py3-none-any.whl (3.1 kB)
Installing collected packages: piglatin
Successfully installed piglatin-1.0.6
```

4. Check the contents of the directory.

```
testpiglatin $ ls
```

Output

```
bin lib pyvenv.cfg unpacked
```

5. Change to the `unpacked` directory and display the contents.

```
testpiglatin $ cd unpacked
unpacked $ ls
```

Output

```
piglatin piglatin-1.0.6.dist-info
```

6. Use the `zip` command to place the contents of the piglatin project into a file called `library.zip`.

```
unpacked $ zip -r9 ../library.zip *
```

Output

```
adding: piglatin/ (stored 0%)
adding: piglatin/__init__.py (deflated 56%)
adding: piglatin/__pycache__/ (stored 0%)
adding: piglatin/__pycache__/__init__.cpython-39.pyc (deflated 31%)
adding: piglatin-1.0.6.dist-info/ (stored 0%)
adding: piglatin-1.0.6.dist-info/RECORD (deflated 39%)
adding: piglatin-1.0.6.dist-info/LICENSE (deflated 41%)
adding: piglatin-1.0.6.dist-info/WHEEL (deflated 15%)
adding: piglatin-1.0.6.dist-info/REQUESTED (stored 0%)
adding: piglatin-1.0.6.dist-info/INSTALLER (stored 0%)
adding: piglatin-1.0.6.dist-info/METADATA (deflated 48%)
```

7. (Optional) Use the following commands to test the import locally.

   a. Set the Python path to the `library.zip` file location and start Python.

   ```
   /home $ PYTHONPATH=/tmp/testpiglatin/library.zip
   /home $ python3
   ```

   Output

   ```
   Python 3.9.6 (default, Jun 29 2021, 06:20:32)
   [Clang 12.0.0 (clang-1200.0.32.29)] on darwin
   Type "help", "copyright", "credits" or "license" for more information.
   ```

   b. Import the library and run a test command.

   ```
   >>> import piglatin
   >>> piglatin.translate('hello')
   ```

Output

```
'ello-hay'
```

8. Use commands like the following to add the `.zip` file from Amazon S3, import it into your notebook in Athena, and test it.

```
sc.addPyFile('s3://amzn-s3-demo-bucket/library.zip')

import piglatin
piglatin.translate('hello')

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(piglatin.translate)

df = spark.createDataFrame([(1, "hello"), (2, "world")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

Output

```
Calculation started (calculation_id=e2c0a06e-f45d-d96d-9b8c-ff6a58b2a525) in
 (session=82c0a06d-d60e-8c66-5d12-23bcd55a6457). Checking calculation status...
Calculation completed.
+---+-----+--------+
| _1|   _2|     col|
+---+-----+--------+
|  1|hello|ello-hay|
|  2|world|orld-way|
+---+-----+--------+
```

Import a Python .zip file from PyPI that has dependencies

This example imports the md2gemini package, which converts text in markdown to Gemini text format, from PyPI. The package has the following dependencies:

```
cjkwrap
mistune
```

```
wcwidth
```

To import a Python .zip file that has dependencies

1. On your local computer, use the following commands to create a directory called `testmd2gemini` and create a virtual environment.

   ```
   /tmp $ mkdir testmd2gemini
   /tmp $ cd testmd2gemini
   testmd2gemini$ virtualenv .
   ```

2. Create a subdirectory named `unpacked` to hold the project.

   ```
   testmd2gemini $ mkdir unpacked
   ```

3. Use the `pip` command to install the project into the `unpacked` directory.

   ```
   /testmd2gemini $ bin/pip install -t $PWD/unpacked md2gemini
   ```

   Output

   ```
   Collecting md2gemini
     Downloading md2gemini-1.9.0-py3-none-any.whl (31 kB)
   Collecting wcwidth
     Downloading wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
   Collecting mistune<3,>=2.0.0
     Downloading mistune-2.0.2-py2.py3-none-any.whl (24 kB)
   Collecting cjkwrap
     Downloading CJKwrap-2.2-py2.py3-none-any.whl (4.3 kB)
   Installing collected packages: wcwidth, mistune, cjkwrap, md2gemini
   Successfully installed cjkwrap-2.2 md2gemini-1.9.0 mistune-2.0.2 wcwidth-0.2.5
   ...
   ```

4. Change to the `unpacked` directory and check the contents.

   ```
   testmd2gemini $ cd unpacked
   unpacked $ ls -lah
   ```

   Output

   ```
   total 16
   ```

```
drwxr-xr-x  13 user1  wheel   416B Jun  7 18:43 .
drwxr-xr-x   8 user1  wheel   256B Jun  7 18:44 ..
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 CJKwrap-2.2.dist-info
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 __pycache__
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 bin
-rw-r--r--   1 user1  staff   5.0K Jun  7 18:43 cjkwrap.py
drwxr-xr-x   7 user1  staff   224B Jun  7 18:43 md2gemini
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 md2gemini-1.9.0.dist-info
drwxr-xr-x  12 user1  staff   384B Jun  7 18:43 mistune
drwxr-xr-x   8 user1  staff   256B Jun  7 18:43 mistune-2.0.2.dist-info
drwxr-xr-x  16 user1  staff   512B Jun  7 18:43 tests
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 wcwidth
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 wcwidth-0.2.5.dist-info
```

5. Use the `zip` command to place the contents of the md2gemini project into a file called `md2gemini.zip`.

```
unpacked $ zip -r9 ../md2gemini *
```

Output

```
  adding: CJKwrap-2.2.dist-info/ (stored 0%)
  adding: CJKwrap-2.2.dist-info/RECORD (deflated 37%)
  ....
  adding: wcwidth-0.2.5.dist-info/INSTALLER (stored 0%)
  adding: wcwidth-0.2.5.dist-info/METADATA (deflated 62%)
```

6. (Optional) Use the following commands to test that the library works on your local computer.

   a. Set the Python path to the `md2gemini.zip` file location and start Python.

   ```
   /home $ PYTHONPATH=/tmp/testmd2gemini/md2gemini.zip
   /home python3
   ```

   b. Import the library and run a test.

   ```
   >>> from md2gemini import md2gemini
   >>> print(md2gemini('[abc](https://abc.def)'))
   ```

   Output

```
https://abc.def abc
```

7. Use the following commands to add the `.zip` file from Amazon S3, import it into your notebook in Athena, and perform a non UDF test.

```
# (non udf test)
sc.addPyFile('s3://amzn-s3-demo-bucket/md2gemini.zip')
from md2gemini import md2gemini
print(md2gemini('[abc](https://abc.def)'))
```

Output

```
Calculation started (calculation_id=0ac0a082-6c3f-5a8f-eb6e-f8e9a5f9bc44) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
=> https://abc.def (https://abc.def/) abc
```

8. Use the following commands to perform a UDF test.

```
# (udf test)

from pyspark.sql.functions import udf
from pyspark.sql.functions import col
from md2gemini import md2gemini


hi_udf = udf(md2gemini)
df = spark.createDataFrame([(1, "[first website](https://abc.def)"), (2, "[second
 website](https://aws.com)")])
df.withColumn("col", hi_udf(col('_2'))).show()
```

Output

```
Calculation started (calculation_id=60c0a082-f04d-41c1-a10d-d5d365ef5157) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
+---+--------------------+--------------------+
| _1|                  _2|                 col|
+---+--------------------+--------------------+
|  1|[first website](h...|=> https://abc.de...|
```

```
|  2|[second website](...|=> https://aws.co...|
+---+--------------------+--------------------+
```

Use Spark properties to specify custom configuration

When you create or edit a session in Amazon Athena for Apache Spark, you can use Spark properties to specify `.jar` files, packages, or another custom configuration for the session. To specify your Spark properties, you can use the Athena console, the AWS CLI, or the Athena API.

Use the Athena console to specify Spark properties

In the Athena console, you can specify your Spark properties when you create a notebook or edit a current session.

To add properties in the Create notebook or Edit session details dialog box

1. Expand **Spark properties**.
2. To add your properties, use the **Edit in table** or **Edit in JSON** option.

   - For the **Edit in table** option, choose **Add property** to add a property, or choose **Remove** to remove a property. Use the **Key** and **Value** boxes to enter property names and their values.
     - To add a custom `.jar` file, use the `spark.jars` property.
     - To specify a package file, use the `spark.jars.packages` property.
   - To enter and edit your configuration directly, choose the **Edit in JSON** option. In the JSON text editor, you can perform the following tasks:
     - Choose **Copy** to copy the JSON text to the clipboard.
     - Choose **Clear** to remove all text from the JSON editor.
     - Choose the settings (gear) icon to configure line wrapping or choose a color theme for the JSON editor.

Notes

- You can set properties in Athena for Spark, which is the same as setting Spark properties directly on a SparkConf object.
- Start all Spark properties with the `spark.` prefix. Properties with other prefixes are ignored.

- Not all Spark properties are available for custom configuration on Athena. If you submit a `StartSession` request that has a restricted configuration, the session fails to start.
  - You cannot use the `spark.athena.` prefix because it is reserved.

Use the AWS CLI or Athena API to provide custom configuration

To use the AWS CLI or Athena API to provide your session configuration, use the StartSession API action or the start-session CLI command. In your `StartSession` request, use the `SparkProperties` field of EngineConfiguration object to pass your configuration information in JSON format. This starts a session with your specified configuration. For request syntax, see StartSession in the *Amazon Athena API Reference*.

Supported data and storage formats

The following table shows formats that are supported natively in Athena for Apache Spark.

Data format	Read	Write	Write compression
parquet	yes	yes	none, uncompressed, snappy, gzip
orc	yes	yes	none, snappy, zlib, lzo
json	yes	yes	bzip2, gzip, deflate
csv	yes	yes	bzip2, gzip, deflate
text	yes	yes	none, bzip2, gzip, deflate
binary file	yes	N/A	N/A

Monitor Apache Spark calculations with CloudWatch metrics

Athena publishes calculation-related metrics to Amazon CloudWatch when the **Publish CloudWatch metrics** option for your Spark-enabled workgroup is selected. You can create custom dashboards, set alarms and triggers on metrics in the CloudWatch console.

Athena publishes the following metric to the CloudWatch console under the `AmazonAthenaForApacheSpark` namespace:

- `DPUCount` – number of DPUs consumed during the session to execute the calculations.

This metric has the following dimensions:

- `SessionId` – The ID of the session in which the calculations are submitted.
- `WorkGroup` – Name of the workgroup.

To view metrics for Spark-enabled workgroups in the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select the **AmazonAthenaForApacheSpark** namespace.

To view metrics with the CLI

- Do one of the following:
  - To list the metrics for Athena Spark-enabled workgroups, open a command prompt, and use the following command:

    ```
    aws cloudwatch list-metrics --namespace "AmazonAthenaForApacheSpark"
    ```

  - To list all available metrics, use the following command:

    ```
    aws cloudwatch list-metrics
    ```

List of CloudWatch metrics and dimensions for Apache Spark calculations in Athena

If you've enabled CloudWatch metrics in your Spark-enabled Athena workgroup, Athena sends the following metric to CloudWatch per workgroup. The metric uses the `AmazonAthenaForApacheSpark` namespace.

Metric name	Description
DPUCount	Number of DPUs (data processing units) consumed during the session to execute the calculations. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory.

This metric has the following dimensions.

Dimension	Description
SessionId	The ID of the session in which the calculations are submitted.
WorkGroup	The name of the workgroup.

Enable requester pays Amazon S3 buckets in Athena for Spark

When an Amazon S3 bucket is configured as requester pays, the account of the user running the query is charged for data access and data transfer fees associated with the query. For more information, see Using Requester Pays buckets for storage transfers and usage in the *Amazon S3 User Guide*.

In Athena for Spark, requester pays buckets are enabled per session, not per workgroup. At a high level, enabling requester pays buckets includes the following steps:

1. In the Amazon S3 console, enable requester pays on the properties for the bucket and add a bucket policy to specify access.
2. In the IAM console, create an IAM policy to allow access to the bucket, and then attach the policy to the IAM role that will be used to access the requester pays bucket.
3. In Athena for Spark, add a session property to enable the requester pays feature.

Step 1: Enable requester pays on an Amazon S3 bucket and add a bucket policy

To enable requester pays on an Amazon S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the list of buckets, choose the link for the bucket that you want to enable requester pays for.
3. On the bucket page, choose the **Properties** tab.
4. Scroll down to the **Requester pays** section, and then choose **Edit**.
5. On the **Edit requester pays** page, choose **Enable**, and then choose **Save changes**.
6. Choose the **Permissions** tab.
7. In the **Bucket policy** section, choose **Edit**.
8. On the **Edit bucket policy** page, apply the bucket policy that you want to the source bucket. The following example policy gives access to all AWS principals (`"AWS": "*"`), but your access can be more granular. For example, you might want to specify only a specific IAM role in another account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

Step 2: Create an IAM policy and attach it to an IAM role

Next, you create an IAM policy to allow access to the bucket. Then you attach the policy to the role that will be used to access the requester pays bucket.

To create an IAM policy for the requester pays bucket and attach the policy to a role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. Choose **JSON**.
5. In the **Policy editor**, add a policy like the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

6. Choose **Next**.
7. On the **Review and create** page, enter a name for the policy and an optional description, and then choose **Create policy**.
8. In the navigation pane, choose **Roles**.
9. On the **Roles** page, find the role that you want to use, and then choose the role name link.
10. In the **Permissions policies** section, choose **Add permissions**, **Attach policies**.
11. In the **Other permissions policies** section, select the check box for the policy that you created, and then choose **Add permissions**.

Step 3: Add an Athena for Spark session property

After you have configured the Amazon S3 bucket and associated permissions for requester pays, you can enable the feature in an Athena for Spark session.

To enable requester pays buckets in an Athena for Spark session

1. In the notebook editor, from the **Session** menu on the upper right, choose **Edit session**.
2. Expand **Spark properties**.
3. Choose **Edit in JSON**.
4. In the JSON text editor, enter the following:

```
{
  "spark.hadoop.fs.s3.useRequesterPaysHeader":"true"
}
```

5. Choose **Save**.

Enable Apache Spark encryption

You can enable Apache Spark encryption on Athena. Doing so encrypts data in transit between Spark nodes and also encrypts data at rest stored locally by Spark. To enhance security for this data, Athena uses the following encryption configuration:

```
spark.io.encryption.keySizeBits="256"
spark.io.encryption.keygen.algorithm="HmacSHA384"
```

To enable Spark encryption, you can use the Athena console, the AWS CLI, or the Athena API.

Use the Athena console to enable Spark encryption in a new notebook

To create a new notebook that has Spark encryption enabled

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.
3. Do one of the following:

   - In **Notebook explorer**, choose **Create notebook**.

- In **Notebook editor**, choose **Create notebook**, or choose the plus icon (**+**) to add a notebook.

4. For **Notebook name**, enter a name for the notebook.
5. Expand the **Spark properties** option.
6. Select **Turn on Spark encryption**.
7. Choose **Create**.

The notebook session that you create is encrypted. Use the new notebook as you normally would. When you later launch new sessions that use the notebook, the new sessions will also be encrypted.

Use the Athena console to enable Spark encryption for an existing notebook

You can also use the Athena console to enable Spark encryption for an existing notebook.

To enable encryption for an existing notebook

1. Open a new session for a previously created notebook.
2. In the notebook editor, from the **Session** menu on the upper right, choose **Edit session**.
3. In the **Edit session details** dialog box, expand **Spark properties**.
4. Select **Turn on Spark encryption**.
5. Choose **Save**.

The console launches a new session that has encryption enabled. Later sessions that you create for this notebook will also have encryption enabled.

Use the AWS CLI to enable Spark encryption

You can use the AWS CLI to enable encryption when you launch a session by specifying the appropriate Spark properties.

To use the AWS CLI to enable Spark encryption

1. Use a command like the following to create an engine configuration JSON object that specifies Spark encryption properties.

```
ENGINE_CONFIGURATION_JSON=$(
  cat <<EOF
```

```
{
    "CoordinatorDpuSize": 1,
    "MaxConcurrentDpus": 20,
    "DefaultExecutorDpuSize": 1,
    "SparkProperties": {
      "spark.authenticate": "true",
      "spark.io.encryption.enabled": "true",
      "spark.network.crypto.enabled": "true"
    }
}
EOF
)
```

2. In the AWS CLI, use the `athena start-session` command and pass in the JSON object that you created to the `--engine-configuration` argument, as in the following example:

```
aws athena start-session \
   --region "region" \
   --work-group "your-work-group" \
   --engine-configuration "$ENGINE_CONFIGURATION_JSON"
```

Use the Athena API to enable Spark encryption

To enable Spark encryption with the Athena API, use the StartSession action and its EngineConfiguration `SparkProperties` parameter to specify the encryption configuration in your `StartSession` request.

Configure cross-account AWS Glue access in Athena for Spark

This topic shows how consumer account *666666666666* and owner account *999999999999* can be configured for cross-account AWS Glue access. When the accounts are configured, the consumer account can run queries from Athena for Spark on the owner's AWS Glue databases and tables.

Step 1: In AWS Glue, provide access to consumer roles

In AWS Glue, the owner creates a policy that provides the consumer's roles access to the owner's AWS Glue data catalog.

To add a AWS Glue policy that allows a consumer role access to the owner's data catalog

1. Using the catalog owner's account, sign in to the AWS Management Console.
2. Open the AWS Glue console at https://console.aws.amazon.com/glue/.
3. In the navigation pane, expand **Data Catalog**, and then choose **Catalog settings**.
4. On the **Data catalog settings** page, in the **Permissions** section, add a policy like the following. This policy provides roles for the consumer account *666666666666* access to the data catalog in the owner account *999999999999*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Cataloguers",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::666666666666:role/Admin",
          "arn:aws:iam::666666666666:role/AWSAthenaSparkExecutionRole"
        ]
      },
      "Action": "glue:*",
      "Resource": [
        "arn:aws:glue:us-west-2:999999999999:catalog",
        "arn:aws:glue:us-west-2:999999999999:database/*",
        "arn:aws:glue:us-west-2:999999999999:table/*"
      ]
    }
  ]
}
```

Step 2: Configure the consumer account for access

In the consumer account, create a policy to allow access to the owner's AWS Glue Data Catalog, databases, and tables, and attach the policy to a role. The following example uses consumer account *666666666666*.

To create a AWS Glue policy for access to the owner's AWS Glue Data Catalog

1. Using the consumer account, sign into the AWS Management Console.

2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, expand **Access management**, and then choose **Policies**.
4. Choose **Create policy**.
5. On the **Specify permissions** page, choose **JSON**.
6. In the **Policy editor**, enter a JSON statement like the following that allows AWS Glue actions on the owner account's data catalog.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/*",
                "arn:aws:glue:us-east-1:999999999999:table/*"
            ]
        }
    ]
}
```

7. Choose **Next**.
8. On the **Review and create** page, for **Policy name**, enter a name for the policy.
9. Choose **Create policy**.

Next, you use IAM console in the consumer account to attach the policy that you just created to the IAM role or roles that the consumer account will use to access the owner's data catalog.

To attach the AWS Glue policy to the roles in the consumer account

1. In the consumer account IAM console navigation pane, choose **Roles**.
2. On the **Roles** page, find the role that you want to attach the policy to.
3. Choose **Add permissions**, and then choose **Attach policies**.
4. Find the policy that you just created.
5. Select the check box for the policy, and then choose **Add permissions**.
6. Repeat the steps to add the policy to other roles that you want to use.

Step 3: Configure a session and create a query

In Athena Spark, in the requester account, using the role specified, create a session to test access by creating a notebook or editing a current session. When you configure the session properties, specify one of the following:

- **The AWS Glue catalog separator** – With this approach, you include the owner account ID in your queries. Use this method if you are going to use the session to query data catalogs from different owners.
- **The AWS Glue catalog ID** – With this approach, you query the database directly. This method is more convenient if you are going to use the session to query only a single owner's data catalog.

Use the AWS Glue catalog separator

When you edit the session properties, add the following:

```
{
    "spark.hadoop.aws.glue.catalog.separator": "/"
}
```

When you run a query in a cell, use syntax like that in the following example. Note that in the FROM clause, the catalog ID and separator are required before the database name.

```
df = spark.sql('SELECT requestip, uri, method, status FROM `999999999999/
mydatabase`.cloudfront_logs LIMIT 5')
df.show()
```

Use the AWS Glue catalog ID

When you edit the session properties, enter the following property. Replace *999999999999* with the owner account ID.

```
{
    "spark.hadoop.hive.metastore.glue.catalogid": "999999999999"
}
```

When you run a query in a cell, use syntax like the following. Note that in the FROM clause, the catalog ID and separator are not required before the database name.

```
df = spark.sql('SELECT * FROM mydatabase.cloudfront_logs LIMIT 10')
df.show()
```

Additional resources

Configure cross-account access to AWS Glue data catalogs

Managing cross-account permissions using both AWS Glue and Lake Formation in the *AWS Lake Formation Developer Guide*.

Configure cross-account access to a shared AWS Glue Data Catalog using Amazon Athena in *AWS Prescriptive Guidance Patterns*.

Understand service quotas for Athena for Spark

Service quotas, also known as *limits*, are the maximum number of service resources or operations that your AWS account can use. For more information about the service quotas for other AWS services that you can use with Amazon Athena for Spark, see AWS service quotas in the *Amazon Web Services General Reference*.

> **Note**
>
> New AWS accounts might have initial lower quotas that can increase over time. Amazon Athena for Apache Spark monitors account usage within each AWS Region, and then automatically increases the quotas based on your usage. If your requirements exceed the stated limits, contact customer support.

The following table lists the service quotas for Amazon Athena for Apache Spark.

Name	Default	Adjustable	Description
Apache Spark DPU concurrency	160	No	The maximum number of data processing units (DPUs) that you can consume concurrently for Apache Spark calculations for a single account in the current AWS Region. A DPU is

Name	Default	Adjustable	Description
			a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory.
Apache Spark session DPU concurrency	60	No	The maximum number of DPUs you can consume concurrently for an Apache Spark calculation within a session.

Use Athena notebook APIs

The following list contains reference links to the Athena notebook API actions. For data structures and other Athena API actions, see the *Amazon Athena API Reference*.

- CreateNotebook
- CreatePresignedNotebookUrl
- DeleteNotebook
- ExportNotebook
- GetCalculationExecution
- GetCalculationExecutionCode
- GetCalculationExecutionStatus
- GetNotebookMetadata
- GetSession
- GetSessionStatus
- ImportNotebook
- ListApplicationDPUSizes
- ListCalculationExecutions
- ListExecutors
- ListNotebookMetadata
- ListNotebookSessions
- ListSessions

- StartCalculationExecution
- StartSession
- StopCalculationExecution
- TerminateSession
- UpdateNotebook
- UpdateNotebookMetadata

Troubleshoot Athena for Spark

Use the following information to troubleshoot issues you may have when using notebooks and sessions on Athena.

Topics

Learn about known issues in Athena for Spark

This page documents some of the known issues in Athena for Apache Spark.

Illegal argument exception when creating a table

Although Spark does not allow databases to be created with an empty location property, databases in AWS Glue can have an empty `LOCATION` property if they are created outside of Spark.

If you create a table and specify a AWS Glue database that has an empty `LOCATION` field, an exception like the following can occur: IllegalArgumentException: Cannot create a path from an empty string.

For example, the following command throws an exception if the default database in AWS Glue contains an empty LOCATION field:

```
spark.sql("create table testTable (firstName STRING)")
```

Suggested solution A – Use AWS Glue to add a location to the database that you are using.

To add a location to an AWS Glue database

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the navigation pane, choose **Databases**.
3. In the list of databases, choose the database that you want to edit.
4. On the details page for the database, choose **Edit**.
5. On the **Update a database** page, for **Location**, enter an Amazon S3 location.
6. Choose **Update Database**.

Suggested solution B – Use a different AWS Glue database that has an existing, valid location in Amazon S3. For example, if you have a database named `dbWithLocation`, use the command `spark.sql("use dbWithLocation")` to switch to that database.

Suggested solution C – When you use Spark SQL to create the table, specify a value for `location`, as in the following example.

```
spark.sql("create table testTable (firstName STRING)
       location 's3://amzn-s3-demo-bucket/'").
```

Suggested solution D – If you specified a location when you created the table, but the issue still occurs, make sure the Amazon S3 path you provide has a trailing forward slash. For example, the following command throws an illegal argument exception:

```
spark.sql("create table testTable (firstName STRING)
       location 's3://amzn-s3-demo-bucket'")
```

To correct this, add a trailing slash to the location (for example, `'s3://amzn-s3-demo-bucket/'`).

Database created in a workgroup location

If you use a command like `spark.sql('create database db')` to create a database and do not specify a location for the database, Athena creates a subdirectory in your workgroup location and uses that location for the newly created database.

Issues with Hive managed tables in the AWS Glue default database

If the `Location` property of your default database in AWS Glue is nonempty and specifies a valid location in Amazon S3, and you use Athena for Spark to create a Hive managed table in your AWS Glue default database, data are written to the Amazon S3 location specified in your Athena Spark workgroup instead of to the location specified by the AWS Glue database.

This issue occurs because of how Apache Hive handles its default database. Apache Hive creates table data in the Hive warehouse root location, which can be different from the actual default database location.

When you use Athena for Spark to create a Hive managed table under the default database in AWS Glue, the AWS Glue table metadata can point to two different locations. This can cause unexpected behavior when you attempt an `INSERT` or `DROP TABLE` operation.

The steps to reproduce the issue are the following:

1. In Athena for Spark, you use one of the following methods to create or save a Hive managed table:
   - A SQL statement like `CREATE TABLE $tableName`
   - A PySpark command like `df.write.mode("overwrite").saveAsTable($tableName)` that does not specify the `path` option in the Dataframe API.

   At this point, the AWS Glue console may show an incorrect location in Amazon S3 for the table.
2. In Athena for Spark, you use the `DROP TABLE $table_name` statement to drop the table that you created.
3. After you run the `DROP TABLE` statement, you notice that the underlying files in Amazon S3 are still present.

To resolve this issue, do one of the following:

Solution A – Use a different AWS Glue database when you create Hive managed tables.

Solution B – Specify an empty location for the default database in AWS Glue. Then, create your managed tables in the default database.

CSV and JSON file format incompatibility between Athena for Spark and Athena SQL

Due to a known issue with open source Spark, when you create a table in Athena for Spark on CSV or JSON data, the table might not be readable from Athena SQL, and vice versa.

For example, you might create a table in Athena for Spark in one of the following ways:

- With the following `USING csv` syntax:

```
spark.sql('''CREATE EXTERNAL TABLE $tableName (
$colName1 $colType1,
$colName2 $colType2,
$colName3 $colType3)
USING csv
PARTITIONED BY ($colName1)
LOCATION $s3_location''')
```

- With the following DataFrame API syntax:

```
df.write.format('csv').saveAsTable($table_name)
```

Due to the known issue with open source Spark, queries from Athena SQL on the resulting tables might not succeed.

Suggested solution – Try creating the table in Athena for Spark using Apache Hive syntax. For more information, see CREATE HIVEFORMAT TABLE in the Apache Spark documentation.

Troubleshoot Spark-enabled workgroups

Use the following information to troubleshoot Spark-enabled workgroups in Athena.

Session stops responding when using an existing IAM role

If you did not create a new `AWSAthenaSparkExecutionRole` for your Spark enabled workgroup and instead updated or chose an existing IAM role, your session might stop responding. In this case, you may need to add the following trust and permissions policies to your Spark enabled workgroup execution role.

Add the following example trust policy. The policy includes a confused deputy check for the execution role. Replace the values for *111122223333*, *aws-region*, and *workgroup-name* with the AWS account ID, AWS Region, and workgroup that you are using.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "athena.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:athena:aws-
region:111122223333:workgroup/workgroup-name"
                }
            }
        }
    ]
}
```

Add a permissions policy like the following default policy for notebook enabled workgroups. Modify the placeholder Amazon S3 locations and AWS account IDs to correspond to the ones that you are using. Replace the values for `amzn-s3-demo-bucket`, *aws-region*, *111122223333*, and *workgroup-name* with the Amazon S3 bucket, AWS Region, AWS account ID, and workgroup that you are using.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:CreatePresignedNotebookUrl",
                "athena:TerminateSession",
                "athena:GetSession",
                "athena:GetSessionStatus",
                "athena:ListSessions",
                "athena:StartCalculationExecution",
                "athena:GetCalculationExecutionCode",
                "athena:StopCalculationExecution",
                "athena:ListCalculationExecutions",
                "athena:GetCalculationExecution",
                "athena:GetCalculationExecutionStatus",
                "athena:ListExecutors",
                "athena:ExportNotebook",
                "athena:UpdateNotebook"
            ],
            "Resource": "arn:aws:athena:aws-region:111122223333:workgroup/workgroup-
name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena:*",
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena*:log-
stream:*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": "logs:DescribeLogGroups",
            "Resource": "arn:aws:logs:aws-region:111122223333:log-group:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "AmazonAthenaForApacheSpark"
                }
            }
        }
    ]
}
```

Use the Spark EXPLAIN statement to troubleshoot Spark SQL

You can use the Spark EXPLAIN statement with Spark SQL to troubleshoot your Spark code. The following code and output examples show this usage.

Example – Spark SELECT statement

```
spark.sql("select * from select_taxi_table").explain(True)
```

Output

```
Calculation started (calculation_id=20c1ebd0-1ccf-ef14-db35-7c1844876a7e) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Project [*]
+- 'UnresolvedRelation [select_taxi_table], [], false

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Project [VendorID#202L, passenger_count#203L, count#204L]
+- SubqueryAlias spark_catalog.spark_demo_database.select_taxi_table
```

```
   +- Relation spark_demo_database.select_taxi_table[VendorID#202L,
       passenger_count#203L,count#204L] csv

== Optimized Logical Plan ==
Relation spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L] csv

== Physical Plan ==
FileScan csv spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L]
Batched: false, DataFilters: [], Format: CSV,
Location: InMemoryFileIndex(1 paths)
[s3://amzn-s3-demo-bucket/select_taxi],
PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<VendorID:bigint,passenger_count:bigint,count:bigint>
```

Example – Spark data frame

The following example shows how to use EXPLAIN with a Spark data frame.

```
taxi1_df=taxi_df.groupBy("VendorID", "passenger_count").count()
taxi1_df.explain("extended")
```

Output

```
Calculation started (calculation_id=d2c1ebd1-f9f0-db25-8477-3effc001b309) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Aggregate ['VendorID, 'passenger_count],
['VendorID, 'passenger_count, count(1) AS count#321L]
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,
extra#60,mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
```

```
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Optimized Logical Plan ==
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
+- Project [VendorID#49L, passenger_count#52L]
   +- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[VendorID#49L, passenger_count#52L], functions=[count(1)],
output=[VendorID#49L, passenger_count#52L, count#321L])
   +- Exchange hashpartitioning(VendorID#49L, passenger_count#52L, 1000),
      ENSURE_REQUIREMENTS, [id=#531]
      +- HashAggregate(keys=[VendorID#49L, passenger_count#52L],
         functions=[partial_count(1)], output=[VendorID#49L,
         passenger_count#52L, count#326L])
         +- FileScan parquet [VendorID#49L,passenger_count#52L] Batched: true,
            DataFilters: [], Format: Parquet,
            Location: InMemoryFileIndex(1 paths)[s3://amzn-s3-demo-bucket/
            notebooks/yellow_tripdata_2016-01.parquet], PartitionFilters: [],
            PushedFilters: [],
            ReadSchema: struct<VendorID:bigint,passenger_count:bigint>
```

Log Spark application events in Athena

The Athena notebook editor allows for standard Jupyter, Spark, and Python logging. You can use `df.show()` to display PySpark DataFrame contents or use `print("Output")` to display values in the cell output. The `stdout`, `stderr`, and `results` outputs for your calculations are written to your query results bucket location in Amazon S3.

Log Spark application events to Amazon CloudWatch

Your Athena sessions can also write logs to Amazon CloudWatch in the account that you are using.

Understand log streams and log groups

CloudWatch organizes log activity into log streams and log groups.

Log streams – A CloudWatch log stream is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream.

Log groups – In CloudWatch Logs, a log group is a group of log streams that share the same retention, monitoring, and access control settings.

There is no limit on the number of log streams that can belong to one log group.

In Athena, when you start a notebook session for the first time, Athena creates a log group in CloudWatch that uses the name of your Spark-enabled workgroup, as in the following example.

```
/aws-athena/workgroup-name
```

This log group receives one log stream for each executor in your session that produces at least one log event. An executor is the smallest unit of compute that a notebook session can request from Athena. In CloudWatch, the name of the log stream begins with the session ID and executor ID.

For more information about CloudWatch log groups and log streams, see [Working with log groups and log streams]() in the Amazon CloudWatch Logs User Guide.

Use standard logger objects in Athena for Spark

In an Athena for Spark session, you can use the following two global standard logger objects to write logs to Amazon CloudWatch:

- **athena_user_logger** – Sends logs to CloudWatch only. Use this object when you want to log information your Spark applications directly to CloudWatch, as in the following example.

  ```
  athena_user_logger.info("CloudWatch log line.")
  ```

  The example writes a log event to CloudWatch like the following:

  ```
  AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: CloudWatch log line.
  ```

- **athena_shared_logger** – Sends the same log both to CloudWatch and to AWS for support purposes. You can use this object to share logs with AWS service teams for troubleshooting, as in the following example.

```
athena_shared_logger.info("Customer debug line.")
var = [...some variable holding customer data...]
athena_shared_logger.info(var)
```

The example logs the debug line and the value of the var variable to CloudWatch Logs and sends a copy of each line to Support.

> **Note**
>
> For your privacy, your calculation code and results are not shared with AWS. Make sure that your calls to athena_shared_logger write only the information that you want to make visible to Support.

The provided loggers write events through Apache Log4j and inherit the logging levels of this interface. Possible log level values are DEBUG, ERROR, FATAL, INFO, and WARN or WARNING. You can use the corresponding named function on the logger to produce these values.

> **Note**
>
> Do not rebind the names athena_user_logger or athena_shared_logger. Doing so makes the logging objects unable to write to CloudWatch for the remainder of the session.

Example: Log notebook events to CloudWatch

The following procedure shows how to log Athena notebook events to Amazon CloudWatch Logs.

To log Athena notebook events to Amazon CloudWatch Logs

1. Follow Get started with Apache Spark on Amazon Athena to create a Spark enabled workgroup in Athena with a unique name. This tutorial uses the workgroup name athena-spark-example.
2. Follow the steps in Step 7: Create your own notebook to create a notebook and launch a new session.
3. In the Athena notebook editor, in a new notebook cell, enter the following command:

```
athena_user_logger.info("Hello world.")
```

4. Run the cell.
5. Retrieve the current session ID by doing one of the following:
   - View the cell output (for example, `... session=72c24e73-2c24-8b22-14bd-443bdcd72de4`).
   - In a new cell, run the [magic](https://docs.aws.amazon.com/athena/latest/ug/notebooks-spark-magics.html) command `%session_id`.
6. Save the session ID.
7. With the same AWS account that you are using to run the notebook session, open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
8. In the CloudWatch console navigation pane, choose **Log groups**.
9. In the list of log groups, choose the log group that has the name of your Spark-enabled Athena workgroup, as in the following example.

   ```
   /aws-athena/athena-spark-example
   ```

   The **Log streams** section contains a list of one or more log stream links for the workgroup. Each log stream name contains the session ID, executor ID, and unique UUID separated by forward slash characters.

   For example, if the session ID is `5ac22d11-9fd8-ded7-6542-0412133d3177` and the executor ID is `f8c22d11-9fd8-ab13-8aba-c4100bfba7e2`, the name of the log stream resembles the following example.

   ```
   5ac22d11-9fd8-ded7-6542-0412133d3177/f8c22d11-9fd8-ab13-8aba-c4100bfba7e2/f012d7cb-cefd-40b1-90b9-67358f003d0b
   ```

10. Choose the log stream link for your session.
11. On the **Log events** page, view the **Message** column.

    The log event for the cell that you ran resembles the following:

    ```
    AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: Hello world.
    ```

12. Return to the Athena notebook editor.

13. In a new cell, enter the following code. The code logs a variable to CloudWatch:

```
x = 6
athena_user_logger.warn(x)
```

14. Run the cell.
15. Return to the CloudWatch console **Log events** page for the same log stream.
16. The log stream now contains a log event entry with a message like the following:

```
AthenaForApacheSpark: 2022-01-01 12:00:00,000 WARN builtins: 6
```

Use CloudTrail to troubleshoot Athena notebook API calls

To troubleshoot notebook API calls, you can examine Athena CloudTrail logs to investigate anomalies or discover actions initiated by users. For detailed information about using CloudTrail with Athena, see Log Amazon Athena API calls with AWS CloudTrail.

The following examples demonstrate CloudTrail log entries for Athena notebook APIs.

StartSession

The following example shows the CloudTrail log for a notebook StartSession event.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
```

```
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:05:36Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.10",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "engineConfiguration": {
            "coordinatorDpuSize": 1,
            "maxConcurrentDpus": 20,
            "defaultExecutorDpuSize": 1,
            "additionalConfigs": {
                "NotebookId": "b8f5854b-1042-4b90-9d82-51d3c2fd5c04",
                "NotebookIframeParentUrl": "https://us-east-1.console.aws.amazon.com"
            }
        },
        "notebookVersion": "KeplerJupyter-1.x",
        "sessionIdleTimeoutInMinutes": 20,
        "clientRequestToken": "d646ff46-32d2-42f0-94d1-d060ec3e5d78"
    },
    "responseElements": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e",
        "state": "CREATED"
    },
    "requestID": "d646ff46-32d2-42f0-94d1-d060ec3e5d78",
    "eventID": "b58ce998-eb89-43e9-8d67-d3d8e30561c9",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
```

```
}
```

TerminateSession

The following example shows the CloudTrail log for a notebook TerminateSession event.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:21:03Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "TerminateSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.11",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e"
    },
    "responseElements": {
        "state": "TERMINATING"
    },
    "requestID": "438ea37e-b704-4cb3-9a76-391997cf42ee",
    "eventID": "49026c5a-bf58-4cdb-86ca-978e711ad238",
```

```
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```


ImportNotebook

The following example shows the CloudTrail log for a notebook ImportNotebook event. For security, some content is hidden.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:08:54Z",
    "eventSource": "athena.amazonaws.com",
```

```
    "eventName": "ImportNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "name": "example-notebook-name",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS"
    },
    "responseElements": {
        "notebookId": "05f6225d-bdcc-4935-bc25-a8e19434652d"
    },
    "requestID": "813e777f-6dac-41f4-82a7-e99b7b33f319",
    "eventID": "4abec837-143b-4458-9c1f-fa9fb88ab69b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

UpdateNotebook

The following example shows the CloudTrail log for a notebook UpdateNotebook event. For security, some content is hidden.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
```

```
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:22Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "UpdateNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.13",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "notebookId": "c87553ff-e740-44b5-884f-a70e575e08b9",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS",
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f"
    },
    "responseElements": null,
    "requestID": "baaba1d2-f73d-4df1-a82b-71501e7374f1",
    "eventID": "745cdd6f-645d-4250-8831-d0ffd2fe3847",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
```

```
    }
}
```

StartCalculationExecution

The following example shows the CloudTrail log for a notebook StartCalculationExecution event. For security, some content is hidden.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:37Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartCalculationExecution",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.14",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
```

```
        "description": "Calculation started via Jupyter notebook",
        "codeBlock": "HIDDEN_FOR_SECURITY_REASONS",
        "clientRequestToken": "0111cd63-4fd0-4ad8-a738-fd350115fc21"
    },
    "responseElements": {
        "calculationExecutionId": "82c1ebb4-bd08-e4c3-5631-a662fb2ff2c5",
        "state": "CREATING"
    },
    "requestID": "1a107461-3f1b-481e-b8a2-7fbd524e2373",
    "eventID": "b74dbd00-e839-4bd1-a1da-b75fbc70ab9a",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    }
}
```

Overcome the 68k code block size limit

Athena for Spark has a known calculation code block size limit of 68000 characters. When you run a calculation with a code block over this limit, you can receive the following error message:

'...' at 'codeBlock' failed to satisfy constraint: Member must have length less than or equal to 68000

The following image shows this error in the Athena console notebook editor.

The same error can occur when you use the AWS CLI to run a calculation that has a large code block, as in the following example.

```
aws athena start-calculation-execution \
    --session-id "{SESSION_ID}" \
```

```
    --description "{SESSION_DESCRIPTION}" \
    --code-block "{LARGE_CODE_BLOCK}"
```

The command gives the following error message:

{LARGE_CODE_BLOCK} at 'codeBlock' failed to satisfy constraint: Member must have length less than or equal to 68000

Workaround

To work around this issue, upload the file that has your query or calculation code to Amazon S3. Then, use boto3 to read the file and run your SQL or code.

The following examples assume that you have already uploaded the file that has your SQL query or Python code to Amazon S3.

SQL example

The following example code reads the `large_sql_query.sql` file from an Amazon S3 bucket and then runs the large query that the file contains.

```
s3 = boto3.resource('s3')
def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()

# SQL
sql = read_s3_content('bucket_name', 'large_sql_query.sql')
df = spark.sql(sql)
```

PySpark example

The following code example reads the `large_py_spark.py` file from Amazon S3 and then runs the large code block that is in the file.

```
s3 = boto3.resource('s3')

def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()

# PySpark
```

```
py_spark_code = read_s3_content('bucket_name', 'large_py_spark.py')
exec(py_spark_code)
```

Troubleshoot session errors

Use the information in this section to troubleshoot session issues.

When a custom configuration error occurs during a session start, the Athena for Spark console shows an error message banner. To troubleshoot session start errors, you can check session state change or logging information.

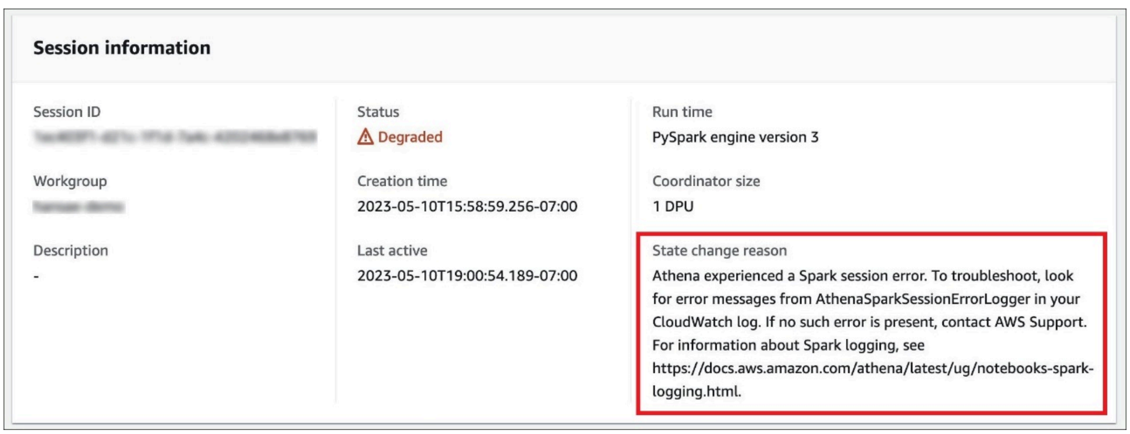
View session state change information

You can get details about a session state change from the Athena notebook editor or from the Athena API.

To view session state information in the Athena console

1. In the Athena notebook editor, from the **Session** menu on the upper right, choose **View details**.
2. View the **Current session** tab. The **Session information** section shows you information like session ID, workgroup, status, and state change reason.

   The following screen capture example shows information in the **State change reason** section of the **Session information** dialog box for a Spark session error in Athena.



To view session state information using the Athena API

- In the Athena API, you can find session state change information in the `StateChangeReason` field of [SessionStatus](#) object.

> **Note**
>
> After you manually stop a session, or if the session stops after an idle timeout (the default is 20 minutes), the value of **StateChangeReason** changes to Session was terminated per request.

Use logging to troubleshoot session start errors

Custom configuration errors that occur during a session start are logged by Amazon CloudWatch. In your CloudWatch Logs, search for error messages from `AthenaSparkSessionErrorLogger` to troubleshoot a failed session start.

For more information about Spark logging, see Log Spark application events in Athena.

For more information about troubleshooting sessions in Athena for Spark, see Troubleshoot session errors.

Specific session issues

Use the information in this section to troubleshoot some specific session issues.

Session in unhealthy state

If you receive the error message Session in unhealthy state. Please create a new session, terminate your existing session and create a new one.

A connection to the notebook server could not be established

When you open a notebook, you may see the following error message:

```
A connection to the notebook server could not be established.
The notebook will continue trying to reconnect.
Check your network connection or notebook server configuration.
```

Cause

When Athena opens a notebook, Athena creates a session and connects to the notebook using a pre-signed notebook URL. The connection to the notebook uses the WSS (WebSocket Secure) protocol.

The error can occur for the following reasons:

- A local firewall (for example, a company-wide firewall) is blocking WSS traffic.
- Proxy or anti-virus software on your local computer is blocking the WSS connection.

Solution

Assume you have a WSS connection in the `us-east-1` Region like the following:

```
wss://94c2bcdf-66f9-4d17-9da6-7e7338060183.analytics-gateway.us-east-1.amazonaws.com/
api/kernels/33c78c82-b8d2-4631-bd22-1565dc6ec152/channels?session_id=
7f96a3a048ab4917b6376895ea8d7535
```

To resolve the error, use one of the following strategies.

- Use wild card pattern syntax to allow list WSS traffic on port `443` across AWS Regions and AWS accounts.

  ```
  wss://*amazonaws.com
  ```

- Use wild card pattern syntax to allow list WSS traffic on port `443` in one AWS Region and across AWS accounts in the AWS Region that you specify. The following example uses `us-east-1`.

  ```
  wss://*analytics-gateway.us-east-1.amazonaws.com
  ```

Troubleshoot table errors

Use the information in this section to troubleshoot Athena for Spark table errors.

Cannot create a path error when creating a table

Error message: IllegalArgumentException: Cannot create a path from an empty string.

Cause: This error can occur when you use Apache Spark in Athena to create a table in an AWS Glue database, and the database has an empty `LOCATION` property.

Suggested Solution: For more information and solutions, see [Illegal argument exception when creating a table](#).

AccessDeniedException when querying AWS Glue tables

Error message: pyspark.sql.utils.AnalysisException: Unable to verify existence of default database: com.amazonaws.services.glue.model.AccessDeniedException: User: arn:aws:sts::*aws-account-id*:assumed-role/AWSAthenaSparkExecutionRole-*unique-identifier*/AthenaExecutor-*unique-identifier* is not authorized to perform: glue:GetDatabase on resource: arn:aws:glue:*aws-region*:*aws-account-id*:catalog because no identity-based policy allows the glue:GetDatabase action (Service: AWSGlue; Status Code: 400; Error Code: AccessDeniedException; Request ID: *request-id*; Proxy: null)

Cause: The execution role for your Spark-enabled workgroup is missing permissions to access AWS Glue resources.

Suggested Solution: To resolve this issue, grant your execution role access to AWS Glue resources, and then edit your Amazon S3 bucket policy to grant access to your execution role.

The following procedure describes these steps in greater detail.

To grant your execution role access to AWS Glue resources

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. If the console navigation pane is not visible, choose the expansion menu on the left.

3. In the Athena console navigation pane, choose **Workgroups**.
4. On the **Workgroups** page, choose the link of the workgroup that you want to view.

5. On the **Overview Details** page for the workgroup, choose the **Role ARN** link. The link opens the Spark execution role in the IAM console.
6. In the **Permissions policies** section, choose the linked role policy name.
7. Choose **Edit policy**, and then choose **JSON**.
8. Add AWS Glue access to the role. Typically, you add permissions for the `glue:GetDatabase` and `glue:GetTable` actions. For more information on configuring IAM roles, see Adding and removing IAM identity permissions in the IAM User Guide.
9. Choose **Review policy**, and then choose **Save changes**.
10. Edit your Amazon S3 bucket policy to grant access to the execution role. Note that you must grant the role access to both the bucket and the objects in the bucket. For steps, see Adding a bucket policy using the Amazon S3 console in the Amazon Simple Storage Service User Guide.

Get support

For assistance from AWS, choose **Support**, **Support Center** from the AWS Management Console. To facilitate your experience, please have the following information ready:

- Athena query ID
- Session ID
- Calculation ID

Release notes

Describes Amazon Athena features, improvements, and bug fixes by release date.

Topics

Athena release notes for 2025

June 30, 2025

Published on 2025-06-30

Amazon Athena announces the availability of Athena SQL in the Asia Pacific (Taipei) Region.

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

June 17, 2025

Published on 2024-06-17

Athena releases ODBC driver version 2.0.4.0. For more information about this version of the driver, see Amazon Athena ODBC 2.x release notes. To download the ODBC 2.x driver, see ODBC 2.x driver download.

June 03, 2025

Published on 2025-06-03

Athena introduces managed query results, a new feature that automatically stores, secures, and manages query result data for you at no cost. Managed query results helps you get started in fewer steps by removing the need for Amazon S3 buckets to store query results and clean up processes to delete query results you no longer need. Managed query results is generally available in all regions where Athena is available except AWS GovCloud (US) Region and China Regions.

When you use managed query results, you can continue to access query results through the same interfaces as with traditional S3 bucket storage. To get started, use the AWS Management Console, AWS SDK, or AWS CLI to configure your new or existing workgroups to use managed query results. For more information, see the section called “Managed query results”.

May 27, 2025

Published on 2025-05-27

Athena announces the following fixes and improvements:

Enhanced Error Messaging for Delta Lake Metadata Tables – We've improved error messaging for Delta Lake metadata tables, now providing clearer feedback when you attempt to query these unsupported tables.

May 14, 2025

Published on 2025-05-14

Athena announces the following fixes and improvements:

- We've resolved an issue that could cause ORC files to be created with larger than intended file sizes.
- Improved performance of scans on Delta Lake tables that use v2 checkpoints.

April 18, 2025

Published on 2025-04-18

Athena announces the following fixes and improvements:

Capacity Reservation tracking – We have resolved an issue in our capacity reservation tracking system where reservations were not being properly released during query cancellation scenarios, specifically after query was planned and before workers are acquired by Athena for query execution. The fix allows Athena query engine to explicitly release capacity reservation when the above scenario is encountered.

April 16, 2025

Published on 2025-04-16

Amazon Athena announces the availability of Athena SQL in the Asia Pacific (Thailand) and Mexico (Central) Regions.

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

April 09, 2025

Published on 2025-04-09

Athena announces the following features and improvements.

JDBC 2.2.1 driver

JDBC 2.2.1 driver release for Athena.

Updates and enhancements:

- Updated Logback libraries to use version 1.3.15.

For more information, and to download the JDBC 2.x driver, release notes, and documentation, see Athena JDBC 2.x driver.

March 18, 2025

Published on 2025-03-18

Athena releases JDBC driver version 3.5.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the latest JDBC driver, see JDBC 3.x driver download.

March 14, 2025

Published on March 14, 2025

Amazon Athena releases capabilities to create and query table operations directly from the S3 console.

For more information, see Register S3 table bucket catalogs and query Tables from Athena.

March 07, 2025

Published on March 07, 2025

Provisioned capacity is now generally available in the Asia Pacific (Mumbai) Region. Provisioned capacity allows you to run SQL queries on fully-managed compute capacity and provides workload management capabilities that help you prioritize, control, and scale your most important interactive workloads. You can add capacity at any time to increase the number of queries that you run concurrently, control which workloads use the capacity, and share capacity among workloads.

For more information, see Manage query processing capacity. For pricing information, visit the Amazon Athena pricing page.

February 18, 2025

Published on 2025-02-18

Athena releases JDBC driver version 3.4.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the latest JDBC driver, see JDBC 3.x driver download.

January 22, 2025

Published on 2025-01-22

Athena now supports federated queries through Lambda and encryption of query results using KMS on TIP enabled workgroups. For more information, see Use IAM Identity Center enabled Athena workgroups.

Athena release notes for 2024

December 17, 2024

Published on 2024-12-17

Amazon Athena announces the availability of Athena SQL in the Asia Pacific (Malaysia).

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

December 16, 2024

Published on 2024-12-16

Deletion vectors fix – Fixed an issue with deletion vectors where partitioned tables returned incorrect results in the Delta Lake connector.

December 3, 2024

Published on 2024-12-03

Athena announces the following features and improvements.

- **Data source connections–** Amazon Athena announces a streamlined console and API workflow for creating data source connections. You can now create and manage Athena data connections entirely within the Athena console, and the properties for your connections are now centrally stored in the AWS Glue Data Catalog.

  Storing connection properties in AWS Glue lets you reuse the connections across other AWS services. For example, after you configure an Athena connector to Amazon DynamoDB, you can reuse the properties and permissions that you specified for the connection for an AWS Glue ETL job that accesses your data in DynamoDB. For more information, see Use the Athena console to connect to a data source in the *Amazon Athena User Guide* and CreateDataCatalog in the *Amazon Athena API Reference*.
- **Querying Redshift data registered in AWS Glue Data Catalog** – Athena now supports reading and writing to Redshift tables that are registered in Glue Data Catalog. For more information, see Register Redshift data catalogs in Athena.
- **Querying S3 tables from Athena** S3 Table Buckets are a bucket type in Amazon S3 that is purpose-built to store tabular data in Apache Iceberg tables. Athena now supports DQL and DML queries on S3 tables. For more information, see Register S3 table bucket catalogs and query Tables from Athena.

October 30, 2024

Published on 2024-10-30

Athena releases JDBC driver version 3.3.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the JDBC 3.x driver, see JDBC 3.x driver download.

August 23, 2024

Published on 2024-09-05

Athena announces the following:

- **Querying federated views with passthrough queries** – Federated passthrough queries are now supported for views. For more information, see Query federated views.
- **Multiple passthrough queries** – You can now run more than one federated passthrough query in the same query execution. For more information, see Use federated passthrough queries.
- **Iceberg table OPTIMIZE fix** – Fixed an issue where running `OPTIMIZE` on an Iceberg table would not remove "delete" files when re-writing data files that had an associated delete file. For more information, see OPTIMIZE.
- **Parquet LZ4 and LZO write support** – Athena no longer supports writing Parquet files compressed with LZ4 or LZO formats. Reads for these compression formats are still supported. For information about compression formats in Athena, see Use compression in Athena.

July 29, 2024

Published on 2024-07-29

Athena releases JDBC driver version 3.2.2. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the JDBC 3.x driver, see JDBC 3.x driver download.

July 26, 2024

Published on 2024-08-01

Athena announces the following improvement.

- **Delta Lake table deletion vector support** – Athena now supports reading from Delta Lake tables with deletion vectors. For more information, see Query Linux Foundation Delta Lake tables.

July 3, 2024

Published on 2024-07-03

Athena releases JDBC driver version 3.2.1. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the JDBC 3.x driver, see JDBC 3.x driver download.

June 26, 2024

Published on 2024-06-26

Provisioned capacity is now generally available in the South America (São Paulo) and Europe (Spain) Regions. Provisioned capacity allows you to run SQL queries on fully-managed compute capacity and provides workload management capabilities that help you prioritize, control, and scale your most important interactive workloads. You can add capacity at any time to increase the number of queries that you run concurrently, control which workloads use the capacity, and share capacity among workloads.

For more information, see Manage query processing capacity. For pricing information, visit the Amazon Athena pricing page.

May 10, 2024

Published on 2024-07-15

Athena announces the following features and improvements.

- **Delta Lake –** Athena added optimizations that filter out unneeded entries from checkpoint files. These optimizations enable significantly improved performance for queries with large checkpoint files that reference many Parquet data files.

For information about using Linux Foundation Delta Lake tables with Athena, see Query Linux Foundation Delta Lake tables.

April 26, 2024

Published on 2024-04-26

Athena releases JDBC driver version 3.2.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes. To download the JDBC 3.x driver, see JDBC 3.x driver download.

April 24, 2024

Published on 2024-04-24

Athena announces the following fixes and improvements.

- **Parquet** – Athena now supports backwards compatible reads in Parquet for unannotated, repeated primitive fields that are not contained within a list or map group. This change prevents silently incorrect results from being returned and improves error messaging for schema mismatches.

  For more information, see Support backwards compatible reads for unannotated repeated primitive fields in Parquet on GitHub.com.
- **Iceberg OPTIMIZE** – Resolved an issue with `OPTIMIZE` queries that caused data to be lost when a non-partition key filter was used in a `WHERE` clause. For more information, see OPTIMIZE.

April 16, 2024

Published on 2024-04-16

Use the new Amazon Athena federated query passthrough feature to run entire queries directly on the underlying data source. Federated passthrough queries help you take advantage of the unique functions, query language, and performance capabilities of the original data source. For example, you can run Athena queries on DynamoDB using the PartiQL language. Federated passthrough queries are also useful when you want to run `SELECT` queries that aggregate, join, or invoke functions of your data source that are not available in Athena. Using passthrough queries can reduce the amount of data processed by Athena and result in faster query times.

For more information, see Use federated passthrough queries. To upgrade the connectors that you use today to the latest version, see Update a data source connector.

April 10, 2024

Published on 2024-04-10

Athena announces the following features and improvements.

ODBC 1.2.3.1000 driver

ODBC 1.2.3.1000 driver release for Athena.

Resolved issues:

- **Proxy server connection issue –** When a proxy server was used without the root certificate, the connector failed to establish a connection.

For more information, and to download the ODBC 1.x driver, release notes, and documentation, see Athena ODBC 1.x driver.

JDBC 2.1.5 driver

JBDC 2.1.5 driver release for Athena.

Updates and enhancements:

- Updated the AWS Java SDK to use version 1.12.687.
- Updated Jackson libraries to use version 2.16.0.
- Updated Logback libraries to use version 1.3.14.

For more information, and to download the JDBC 2.x driver, release notes, and documentation, see Athena JDBC 2.x driver.

April 8, 2024

Published on 2024-04-08

Athena announces ODBC driver version 2.0.3.0. For more information, see the 2.0.3.0 release notes. To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see the Amazon Athena ODBC 2.x.

March 15, 2024

Published on 2024-03-18

Amazon Athena announces the availability of Athena SQL in the Canada West (Calgary) Region.

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

February 15, 2024

Published on 2024-02-15

Athena releases JDBC driver version 3.1.0.

Amazon Athena JDBC driver version 3.1.0 adds support for Microsoft Active Directory Federation Services (AD FS) Windows Integrated Authentication and form-based authentication. The 3.1.0 release also includes other minor improvements and bug fixes.

To download the JDBC v3 driver, see JDBC 3.x driver download.

January 31, 2024

Published on 2024-01-31

Athena announces the following features and improvements.

- **Hudi upgrade** – You can now use Athena SQL to query Hudi 0.14.0 tables. For information about using Athena SQL to query Hudi tables, see Query Apache Hudi datasets.

Athena release notes for 2023

December 14, 2023

Published on 2023-12-14

Athena announces the following fixes and improvements.

Athena releases JDBC driver version 2.1.3. The driver resolves the following issues:

- Logging has been improved to avoid conflicts with Spring Boot and Gradle application logging.
- When using the `executeBatch()` JDBC method to insert records, the driver incorrectly inserted only one record. Because Athena does not support batch execution of queries, the driver now reports an error when you use `executeBatch()`. To work around the limitation, you can submit single queries in a loop.

To download the new JDBC driver, release notes, and documentation, see Athena JDBC 2.x driver.

December 9, 2023

Published on 2023-12-09

Released the ODBC 1.2.1.1000 driver for Athena.

Features and enhancements:

- **Updated RStudio support –** The ODBC driver now supports RStudio on macOS.
- **Single catalog and schema support –** The connector can now return a single catalog and schema. For more information, see the downloadable installation and configuration guide.

Resolved issues:

- **Prepared statements –** When prepared statements with an array of parameters using column-wise schema were run, the connector returned an incorrect query result.
- **Column size –** When the `$file_modified_time` system column was selected, the connector returned an incorrect column size.
- **SQLPrepare –** When binding parameters related to `SQLPrepare` in `SELECT` queries, the connector returned an error.

For more information, and to download the new drivers, release notes, and documentation, see Athena ODBC 1.x driver.

December 7, 2023

Published on 2023-12-07

Athena announces ODBC driver version 2.0.2.1. For more information, see the 2.0.2.1 release notes. To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see the Amazon Athena ODBC 2.x.

December 5, 2023

Published on 2023-12-05

You can now create Athena SQL workgroups that use AWS IAM Identity Center authentication mode. These workgroups support the trusted identity propagation feature of IAM Identity Center.

Trusted identity propagation permits identities to be used across AWS analytics services like Amazon Athena and Amazon EMR Studio.

For more information, see Use IAM Identity Center enabled Athena workgroups.

November 28, 2023

Published on 2023-11-28

You can now query data in the Amazon S3 Express One Zone storage class for fast query results. S3 Express One Zone is a high-performance, single-Availability Zone storage class purpose-built to deliver consistent, single-digit millisecond data access for your most frequently accessed data and latency-sensitive applications. To get started, move your data to S3 Express One Zone storage and catalog the data with AWS Glue Data Catalog for a seamless query experience in Athena.

For more information, see Query S3 Express One Zone data.

November 27, 2023

Published on 2023-11-27

Athena announces the following features and improvements.

- **Glue Data Catalog views –** Glue Data Catalog views provide a single common view across AWS services like Amazon Athena and Amazon Redshift. In Glue Data Catalog views, access permissions are defined by the user who created the view instead of the user who queries the view. These views provide greater access control, help to ensure complete records, offer enhanced security, and can prevent access to underlying tables.

  For more information, see Use Data Catalog views in Athena.
- **CloudTrail Lake support –** You can now use Amazon Athena to analyze data in AWS CloudTrail Lake. AWS CloudTrail Lake is a managed data lake for CloudTrail that you can use to aggregate, immutably store, and analyze activity logs for audit, security, and operational investigations. To query your CloudTrail Lake activity logs from Athena, you do not have to move data or build separate data processing pipelines. No ETL operations are required.

  To get started, enable data federation in CloudTrail Lake. When you share your CloudTrail Lake event data store metadata with AWS Glue Data Catalog, CloudTrail creates the necessary AWS Glue Data Catalog resources and registers the data with AWS Lake Formation. In Lake Formation, you can specify the users and roles that can use Athena to query your event data store.

For more information, see Enable Lake query federation in the *AWS CloudTrail User Guide*.

November 17, 2023

Published on 2023-11-17

Athena announces the following features and improvements.

Features

- **Cost-based optimizer** – Athena announces general availability of cost-based optimization using statistics from AWS Glue. To optimize your queries in Athena SQL, you can request that Athena gather table or column-level statistics for your tables in AWS Glue. If all of the tables in your query have statistics, Athena uses the statistics to examine alternate execution plans and select the one that is most likely to be the fastest.

  For more information, see Use the cost-based optimizer.
- **Amazon EMR Studio integration –** You can now use Athena in an Amazon EMR Studio without having to use the Athena console directly. With the Athena integration in Amazon EMR, you can perform the following tasks:
  - Perform Athena SQL queries
  - View query results
  - View query history
  - View saved queries
  - Perform parameterized queries
  - View databases, tables, and views for a data catalog

  For more information, see Amazon EMR Studio in the AWS service integrations with Athena topic.
- **Nested access control** – Athena announces support for Lake Formation access control for nested data. In Lake Formation, you can define and apply data filters on nested columns that have `struct` data types. You can use data filtering to restrict user access to sub-structures of nested columns. For information on how to create data filters for nested data, see Creating a data filter in the *AWS Lake Formation Developer Guide*.
- **Provisioned capacity usage metrics –** Athena announces new CloudWatch metrics for capacity reservations. You can use the new metrics to keep track of the number of DPUs you have

provisioned and the number of DPUs being used by your queries. When queries finish, you can also view the number of DPUs the query consumed.

For more information, see Monitor Athena query metrics with CloudWatch.

Improvements

- **Error message change –** The `Insufficient Lake Formation permissions` error message now reads `Table not found` or `Schema not found`. This change was made to prevent malicious actors from inferring the existence of table or database resources from the error message.

November 16, 2023

Published on 2023-11-16

Athena releases a new JDBC driver that improves the experience of connecting to, querying, and visualizing data from compatible SQL development and business intelligence applications. The new driver is straightforward to upgrade. The driver can read query results directly from Amazon S3, making query results available to you sooner.

For more information, see Athena JDBC 3.x driver.

October 31, 2023

Published on 2023-10-31

Amazon Athena announces 1-hour reservations for provisioned capacity. Starting today, you can reserve and release provisioned capacity after one hour. This change makes it simpler to optimize cost for workloads whose demand changes over time.

Provisioned capacity is a feature of Athena that provides workload management capabilities that help you prioritize, control, and scale your most important interactive workloads. You can add capacity at any time to increase the number of queries that you run concurrently, control which workloads use the capacity, and share capacity among workloads.

For more information, see Manage query processing capacity. For pricing information, visit the Amazon Athena Pricing page.

October 25, 2023

Published on 2023-10-26

Athena announces the following fixes and improvements.

jackson-core package – JSON text with a numerical value larger than 1000 characters will now fail. This fix addresses the security issue sonatype-2022-6438.

October 17, 2023

Published on 2023-10-17

Athena announces ODBC driver version 2.0.2.0. For more information, see the 2.0.2.0 release notes. To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see the Amazon Athena ODBC 2.x.

September 26, 2023

Published on 2023-09-26

Athena announces the following features and improvements.

- Lake Formation read support for Delta Lake tables. For more information about using Delta Lake tables with Athena, see Query Linux Foundation Delta Lake tables.

August 23, 2023

Published on 2023-08-23

Amazon Athena announces the availability of Athena SQL in the Israel (Tel Aviv) Region.

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

August 10, 2023

Published on 2023-08-10

Athena announces the following fixes and improvements.

ODBC driver version 2.0.1.1

Athena announces ODBC driver version 2.0.1.1. For more information, see the 2.0.1.1 release notes. To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see the Amazon Athena ODBC 2.x.

JDBC driver version 2.1.1

Athena releases JDBC driver version 2.1.1. The driver resolves the following issues:

- An error that occurred when a table was created with a statement that contained a regular expression.
- An issue that caused the `ApplicationName` connection parameter to be applied incorrectly.

To download the new JDBC driver, release notes, and documentation, see Connect to Amazon Athena with JDBC.

July 31, 2023

Published on 2023-07-31

Amazon Athena announces the availability of Athena SQL in additional AWS Regions.

This release expands the availability of Athena SQL to include Asia Pacific (Hyderabad), Asia Pacific (Melbourne), Europe (Spain), and Europe (Zurich).

For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

July 27, 2023

Published on 2023-07-27

Athena releases Google BigQuery connector version 2023.30.1. This version of the connector reduces query execution time and adds support for querying against BigQuery private endpoints.

For information about the Google BigQuery connector, see Amazon Athena Google BigQuery connector. For information about updating your existing data source connectors, see Update a data source connector.

July 24, 2023

Published on 2023-07-24

Athena announces the following fixes and improvements.

- **Queries with unions –** Improved the performance of certain queries with unions.
- **Joins with type comparisons –** Fixed a potential query failure for `JOIN` statements that included a comparison between two different types.
- **Subqueries on nested columns –** Fixed an issue related to query failures when subqueries were correlated on nested columns.
- **Iceberg views –** Fixed a compatibility issue with the precision of timestamp columns in Apache Iceberg views. Iceberg views that have timestamp columns are now readable regardless of whether the columns were created on previous engine versions or Athena engine version 3.

July 20, 2023

Published on 2023-07-20

Athena releases JDBC driver version 2.1.0. The driver includes new enhancements and resolved an issue.

Enhancements

The following Jackson JSON parser libraries have been upgraded:

- jackson-annotations 2.15.2 (previously 2.14.0)
- jackson-core 2.15.2 (previously 2.14.0)
- jackson-databind 2.15.2 (previously 2.14.0)

Resolved issues

- Fixed an issue with passing array parameters when the sql2o library was used.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

July 13, 2023

Published on 2023-09-19

Athena announces the following features and improvements.

- **EXPLAIN ANALYZE** – Added support for queue, analysis, planning, and execution time to the output of `EXPLAIN ANALYZE`.
- **EXPLAIN** – `EXPLAIN` output now shows statistics when the query contains aggregations.
- **Parquet Hive SerDe** – Added the `parquet.ignore.statistics` property to enable processing statistics to be ignored when reading Parquet data. For information, see Ignore Parquet statistics.

For more information about `EXPLAIN` and `EXPLAIN ANALYZE`, see Using EXPLAIN and EXPLAIN ANALYZE in Athena. For more information about the Parquet Hive SerDe, see Parquet SerDe.

July 3, 2023

Published on 2023-07-25

As of July 3, 2023, Athena started redacting the query strings from CloudTrail logs. The query string now has a value of `***OMITTED***`. This change has been made to prevent unintended disclosure of table names or filter values that could include sensitive information. If you previously relied on CloudTrail logs to access full query strings, we recommend using the `Athena::GetQueryExecution` API and passing in the value of `responseElements.queryExecutionId` from the CloudTrail log. For more information, see the GetQueryExecution action in the *Amazon Athena API Reference*.

June 30, 2023

Published on 2023-06-30

The Athena query editor now supports typeahead code suggestions for a faster query authoring experience. You can now write SQL queries with enhanced accuracy and increased efficiency using the following features:

- As you type, suggestions appear in real time for keywords, local variables, snippets, and catalog items.
- When you type a database name or table name followed by a dot, the editor conveniently displays a list of tables or columns to choose from.

- When you hover over a snippet suggestion, a synopsis shows a brief overview of the snippet's syntax and usage.
- To improve code readability, keywords and their highlighting rules have also been updated to align with latest syntax of Trino and Hive.

This feature is enabled by default. You can enable or disable the feature in the code editor preferences settings.

To try the typeahead code suggestions in the Athena query editor, visit the Athena console at https://console.aws.amazon.com/athena/.

June 29, 2023

Published on 2023-06-29

- Athena announces ODBC driver version 2.0.1.0. For more information, see the 2.0.1.0 release notes. To download the new ODBC v2 driver, see ODBC 2.x driver download. For connection information, see the Amazon Athena ODBC 2.x.
- Athena and its features are now available in the Middle East (UAE) Region. For a complete list of the AWS services available in each AWS Region, see AWS Services by Region.

June 28, 2023

Published on 2023-06-28

You can now use Amazon Athena to query restored objects from the S3 Glacier Flexible Retrieval (formerly Glacier) and S3 Glacier Deep Archive Amazon S3 storage classes. You configure this capability on a per-table basis. The feature is supported only for Apache Hive tables on Athena engine version 3.

For more information, see Query restored Amazon S3 Glacier objects.

June 12, 2023

Published on 2023-06-12

Athena announces the following fixes and improvements.

- **Parquet Reader timestamps** – Added support for reading timestamps as `bigint` (millis) for Parquet Reader. This update provides parity with the support in previous engine versions.
- **EXPLAIN ANALYZE** – Added physical input read time to the query statistics and output of `EXPLAIN ANALYZE`. For information about `EXPLAIN ANALYZE`, see Using EXPLAIN and EXPLAIN ANALYZE in Athena.
- **INSERT** – Improved query performance on tables written to with `INSERT`. For information about `INSERT`, see INSERT INTO.
- **Delta Lake tables** – Corrected an issue with `DROP TABLE` on Delta Lake tables that prevented them from being fully deleted when subject to concurrent modifications.

June 8, 2023

Published on 2023-06-08

Amazon Athena for Apache Spark announces the following new features.

- **Support for custom Java libraries and configuration** – You can now use your own Java packages and custom configuration for your Apache Spark sessions in Athena. Use Spark properties to specify `.jar` files, packages, or other custom configuration with the Athena console, the AWS CLI, or the Athena API. For more information, see Use Spark properties to specify custom configuration.
- **Support for Apache Hudi, Apache Iceberg, and Delta Lake tables** – Athena for Spark now supports the Apache Iceberg, Apache Hudi, and Linux Foundation Delta Lake open-source data lake storage table formats. For more information, see Use non-Hive table formats in Athena for Spark and the individual topics for using Use Apache Iceberg tables in Athena for Spark, Use Apache Hudi tables in Athena for Spark, and Use Linux Foundation Delta Lake tables in Athena for Spark tables in Athena for Spark.
- **Encryption support for Apache Spark** – In Athena for Spark, you can now enable encryption on data in transit between Spark nodes and on local data at rest stored on disk by Spark. To enable Spark encryption, you can use the Athena console, the AWS CLI, or the Athena API. For more information, see Enable Apache Spark encryption.

For more information about Amazon Athena for Apache Spark, see Use Apache Spark in Amazon Athena.

June 2, 2023

Published on 2023-06-02

You can now delete capacity reservations in Athena and use AWS CloudFormation templates to specify Athena capacity reservations.

- **Delete capacity reservations** – You can now delete cancelled capacity reservations in Athena. A reservation must be cancelled before it can be deleted. Deleting a capacity reservation removes the reservation from your account immediately. The deleted reservation can no longer be referenced, including by its ARN. To delete a reservation, you can use the Athena console or the Athena API. For more information, see Delete a capacity reservation in the *Amazon Athena User Guide* and DeleteCapacityReservation in the *Amazon Athena API Reference*.
- **Use AWS CloudFormation templates for capacity reservations** – You can now use AWS CloudFormation templates to specify Athena capacity reservations using the `AWS::Athena::CapacityReservation` resource. For more information, see AWS::Athena::CapacityReservation in the *AWS CloudFormation User Guide*.

For more information about using capacity reservations to provision your capacity in Athena, see Manage query processing capacity.

May 25, 2023

Published on 2023-05-25

Athena has released data source connector updates that improve federated query performance. New push-down optimizations and dynamic filtering enable more operations to be performed in the source database rather than in Athena. These optimizations reduce query runtime and the amount of data scanned. These improvements require Athena engine version 3.

The following connectors have been updated:

- Azure Data Lake Storage
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- Db2
- DynamoDB

- Google BigQuery
- Hortonworks
- MySQL
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server
- Teradata

For information about upgrading your data source connectors, see Update a data source connector.

May 18, 2023

Published on 2023-05-18

You can now use AWS PrivateLink for IPv6 inbound connections to Amazon Athena.

Amazon Athena has expanded its support for inbound connections through Internet Protocol Version 6 (IPv6) endpoints to include AWS PrivateLink. Starting today, you can connect to Athena securely and privately using AWS PrivateLink from your Amazon Virtual Private Cloud (Amazon VPC), in addition to the public IPv6 endpoints that were previously available.

The rapid growth of the Internet is exhausting the availability of Internet Protocol version 4 (IPv4) addresses. IPv6 increases the number of available addresses by several times so that you no longer have to manage overlapping address spaces in your VPCs. With this release, you can now combine the benefits of IPv6 addressing with the security and performance advantages of AWS PrivateLink.

To connect programmatically to an AWS service, you can use the AWS CLI or AWS SDK to specify an endpoint. For more information on service endpoints and Athena service endpoints, see AWS service endpoints and Amazon Athena endpoints and quotas in the *Amazon Web Services General Reference*.

May 15, 2023

Published on 2023-05-15

Athena announces the release of Apache Spark DataSourceV2 (DSV2) connectors for DynamoDB, CloudWatch Logs, CloudWatch Metrics, and AWS CMDB. Use the new DSV2 connectors to query these data sources using Spark. DSV2 connectors use the same parameters as their corresponding Athena federated connectors. The DSV2 connectors run directly on Spark workers and do not require you to deploy a Lambda function to use them.

For more information, see Work with data source connectors for Apache Spark.

May 10, 2023

Published on 2023-05-10

Released the ODBC 1.1.20 driver for Athena.

Features and enhancements:

- Lake Formation endpoint override support.
- The ADFS authentication plugin has a new parameter for setting the Relying Party value (`LoginToRP`).
- AWS library updates.

Bug fixes:

- Prepared statement deallocation failure when the `SQLPrepare()` method failed to submit.
- Error in binding prepared statement parameters when converting a C type to SQL type.
- Failure to return data when `EXPLAIN` and `EXPLAIN ANALYZE` queries used `SQLPrepare()` and `SQLExecute()`.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with ODBC.

May 8, 2023

Published on 2023-05-08

Athena announces the following fixes and improvements.

- **Updated Hudi integration –** Athena has updated its integration with Apache Hudi. You can now use Athena to query Hudi 0.12.2 tables, and Hudi metadata listing for Hudi tables is now supported. For information, see Query Apache Hudi datasets and Use Hudi metadata for improved performance.

- **Timestamp conversion fix –** Corrected the handling of timestamp conversions to a lower precision data type. Previously, Athena engine version 3 incorrectly rounded the value to the target type instead of truncating it during casting.

  The following examples illustrate the incorrect handling prior to the fix.

  Example 1: Casting from a timestamp in microseconds to milliseconds

  Sample data

```
A, 2020-06-10 15:55:23.383
B, 2020-06-10 15:55:23.382
C, 2020-06-10 15:55:23.383345
D, 2020-06-10 15:55:23.383945
E, 2020-06-10 15:55:23.383345734
F, 2020-06-10 15:55:23.383945278
```

  The following query tries to retrieve the timestamps that match a specific value.

```
SELECT *
FROM table
WHERE timestamps.col = timestamp'2020-06-10 15:55:23.383'
```

  The query returned the following results.

```
A, 2020-06-10 15:55:23.383
C, 2020-06-10 15:55:23.383
E, 2020-06-10 15:55:23.383
```

  Prior to the fix, Athena did not include the values `2020-06-10 15:55:23.383945` or `2020-06-10 15:55:23.383945278` because they got rounded to `2020-06-10 15:55:23.384`.

  Example 2: Casting from a timestamp to date

  The following query returned an erroneous result.

```
SELECT date(timestamp '2020-12-31 23:59:59.999')
```

Result

`2021-01-01`

Prior to the fix, Athena rounded up the value, therefore moving the day forward. Such values are now truncated rather than rounded up.

April 28, 2023

Published on 2023-04-28

You can now use capacity reservations on Amazon Athena to run SQL queries on fully-managed compute capacity.

Provisioned capacity provides workload management capabilities that help you prioritize, control, and scale your most important interactive workloads. You can add capacity at any time to increase the number of queries that you run concurrently, control which workloads use the capacity, and share capacity among workloads.

For more information, see Manage query processing capacity. For pricing information, visit the Amazon Athena pricing page.

April 17, 2023

Published on 2023-04-17

Athena releases JDBC driver version 2.0.36. The driver includes new features and resolved an issue.

New features

- You can now use customizable relying party identifiers with AD FS authentication.
- You can now add the name of the application that is using the connector to the user agent string.

Resolved issues

- Fixed an error that occurred when `getSchema()` was used to retrieve a non-existent schema.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

April 14, 2023

Published on 2023-06-20

Athena announces the following fixes and improvements.

- When you cast a string to timestamp, a space is required between the day and time or timezone. For more information, see Space required between date and time values when casting from string to timestamp.
- Removed a breaking change in the way timestamp precision was handled. To maintain consistency between previous engine versions and Athena engine version 3, timestamp precision now defaults to milliseconds rather than microseconds.
- Athena now consistently enforces access for the query output bucket when it runs queries. Please make sure that all IAM principals that run the StartQueryExecution action have the S3:GetBucketLocation permission on the query output bucket.

April 4, 2023

Published on 2023-04-04

You can now use Amazon Athena to create and query views on federated data sources. Use a single federated view to query multiple external tables or subsets of data. This simplifies the SQL required and gives you the flexibility of obfuscating sources of data from end users who must use SQL to query the data.

For more information, see Work with views and Run federated queries.

March 30, 2023

Published on 2023-03-30

Amazon Athena announces the availability of Amazon Athena for Apache Spark in additional AWS Regions.

This release expands the availability of Amazon Athena for Apache Spark to include Asia Pacific (Mumbai), Asia Pacific (Singapore), Asia Pacific (Sydney), and Europe (Frankfurt).

For more information about Amazon Athena for Apache Spark, see Use Apache Spark in Amazon Athena.

March 28, 2023

Published on 2023-03-28

Athena announces the following fixes and improvements.

- In the responses to the `GetQueryExecution` and `BatchGetQueryExecution` Athena API actions, the new `subStatementType` field shows the type of query that ran (for example, `SELECT`, `INSERT`, `UNLOAD`, `CREATE_TABLE`, or `CREATE_TABLE_AS_SELECT`).
- Fixed a bug in which manifest files were not encrypted correctly for Apache Hive write operations.
- Athena engine version 3 now correctly handles `NaN` and `Infinity` values in the `approx_percentile` function. The `approx_percentile` function returns the approximate percentile for a dataset at the given percentage.

  Athena engine version 2 incorrectly treats `NaN` as a value greater than `Infinity`. Athena engine version 3 now handles `NaN` and `Infinity` in accordance with the treatment of these values in other analytic and statistical functions. The following points describe the new behavior in greater detail.

  - If `NaN` is present in the dataset, Athena returns `NaN`.
  - If `NaN` is not present, but `Infinity` is present, Athena treats `Infinity` as a very large number.
  - If multiple `Infinity` values are present, Athena treats them as the same very large number. If necessary, Athena outputs `Infinity`.
  - If a single dataset has both `-Infinity` and `-Double.MAX_VALUE`, and a percentile result is `-Double.MAX_VALUE`, Athena returns `-Infinity`.
  - If a single dataset has both `Infinity` and `Double.MAX_VALUE`, and a percentile result is `Double.MAX_VALUE`, Athena returns `Infinity`.
  - To exclude `Infinity` and `NaN` from a calculation, use the `is_finite()` function, as in the following example.

  ```
  approx_percentile(x, 0.5) FILTER (WHERE is_finite(x))
  ```

March 27, 2023

Published on 2023-03-27

You can now specify a minimum level of encryption for Athena SQL workgroups in Amazon Athena. This feature ensures that the results from all queries in the Athena SQL workgroup are encrypted at or above the level of encryption that you specify. You can choose among several levels of encryption strength to safeguard your data. To configure the minimum level of encryption that you want, you can use the Athena console, AWS CLI, API, or SDK.

The minimum encryption feature is not available for Apache Spark enabled workgroups. For more information, see Configure minimum encryption for a workgroup.

March 17, 2023

Published on 2023-03-17

Athena announces the following fixes and improvements.

- Fixed an issue with the Amazon Athena DynamoDB connector that caused queries to fail with the error message KeyConditionExpressions must only contain one condition per key.

  This issue occurs because Athena engine version 3 recognizes the opportunity to push down more kinds of predicates than Athena engine version 2. In Athena engine version 3, clauses like `some_column LIKE 'someprefix%` are pushed down as filter predicates that apply a lower and upper bound on a given column. Athena engine version 2 did not push these predicates down. In Athena engine version 3, when `some_column` is a sort key column, the engine pushes the filter predicate down to the DynamoDB connector. The filter predicate then gets further pushed down to the DynamoDB service. Because DynamoDB does not support more than one filter condition on a sort key, DynamoDB returns the error.

  To correct this issue, update your Amazon Athena DynamoDB connector to version 2023.11.1. For instructions on updating the connector, see Update a data source connector.

March 8, 2023

Published on 2023-03-08

Athena announces the following fixes and improvements.

- Fixed an issue with federated queries that caused timestamp predicate values to be sent as microseconds instead of milliseconds.

February 15, 2023

Published on 2023-02-15

Athena announces the following fixes and improvements.

- You can now use client-side encryption to encrypt data in Amazon S3 for Iceberg write operations.
- Fixed an issue that affected server-side encryption in Amazon S3 for Iceberg write operations.

January 31, 2023

Published on 2023-01-31

You can now use Amazon Athena to query data in Google Cloud Storage. Like Amazon S3, Google Cloud Storage is a managed service that stores data in buckets. Use the Athena connector for Google Cloud Storage to run interactive federated queries on your external data.

For more information, see Amazon Athena Google Cloud Storage connector.

January 20, 2023

Published on 2023-01-20

You can now see expanded documentation for Athena compression support. Individual topics have been added for Hive table compression, Iceberg table compression, and ZSTD compression levels.

For more information, see Use compression in Athena.

January 3, 2023

Published on 2023-01-03

Athena announces the following updates:

- **Additional commands for Hive metastores** – You can use Athena to connect to your self-managed Apache Hive Metastore as a metadata catalog and query data stored in Amazon S3.

With this release, you can use `CREATE TABLE AS` (CTAS), `INSERT INTO`, and 12 additional Data Definition Language (DDL) commands to interact with the Apache Hive Metastore. You can manage your Hive Metastore schemas directly from Athena using this expanded set of SQL capabilities.

For more information, see Use an external Hive metastore.

- **JDBC driver version 2.0.35 –** Athena releases JDBC driver version 2.0.35. The JDBC 2.0.35 driver contains the following updates:
  - The driver now uses the following libraries for the Jackson JSON parser.
    - jackson-annotations 2.14.0 (previously 2.13.2)
    - jackson-core 2.14.0 (previously 2.13.2)
    - jackson-databind 2.14.0 (previously 2.13.2.2)
  - Support for JDBC version 4.1 has been discontinued.

  For more information, and to download the new driver, release notes, and documentation, see Connect to Amazon Athena with JDBC.

Athena release notes for 2022

December 14, 2022

Published on 2022-12-14

You can now use the Amazon Athena connector for Kafka to run SQL queries on streaming data. For example, you can run analytical queries on real-time streaming data in Amazon Managed Streaming for Apache Kafka (Amazon MSK) and join it with historical data in your data lake in Amazon S3.

The Amazon Athena connector for Kafka supports queries on multiple streaming engines. You can use Athena to run SQL queries on Amazon MSK provisioned and serverless clusters, on self-managed Kafka deployments, and on streaming data in Confluent Cloud.

For more information, see Amazon Athena MSK connector.

December 2, 2022

Published on 2022-12-02

Athena releases JDBC driver version 2.0.34. The JDBC 2.0.34 driver includes the following new features and resolved issues:

- **Query result reuse support** – You can now reuse the results of previously executed queries up to a time limit that you specify instead of having Athena recompute the results each time the query is run. For more information, see the Installation and Configuration Guide, available from the JDBC download page, and Reuse query results in Athena.
- **Ec2InstanceMetadata support** – The JDBC driver now supports the Ec2InstanceMetadata authentication method using IAM instance profiles .
- **Character-based exception fix** – Fixed an exception that occurred with queries containing certain language characters.
- **Vulnerability fix** – Corrected a vulnerability related to AWS dependencies packaged with the connector.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

November 30, 2022

Published on 2022-11-30

You can now interactively create and run Apache Spark applications and Jupyter compatible notebooks on Athena. Run data analytics on Athena using Spark without having to plan for, configure, or manage resources. Submit Spark code for processing and receive the results directly. Use the simplified notebook experience in Amazon Athena console to develop Apache Spark applications using Python or Use Athena notebook APIs.

Apache Spark on Amazon Athena is serverless and provides automatic, on-demand scaling that delivers instant-on compute to meet changing data volumes and processing requirements.

For more information, see Use Apache Spark in Amazon Athena.

November 18, 2022

Published on 2022-11-18

You can now use the Amazon Athena connector for IBM Db2 to query Db2 from Athena. For example, you can run analytical queries over a data warehouse on Db2 and a data lake in Amazon S3.

The Amazon Athena Db2 connector exposes several configuration options through Lambda environment variables. For information about configuration options, parameters, connection strings, deployment, and limitations, see Amazon Athena IBM Db2 connector.

November 17, 2022

Published on 2022-11-17

Apache Iceberg support in Athena engine version 3 now offers the following enhanced ACID transaction features:

- **ORC and Avro support** – Create Iceberg tables using the Apache Avro and Apache ORC row and column-based file formats. Support for these formats is in addition to the existing support for Parquet.
- **MERGE INTO** – Use the `MERGE INTO` command to merge data at scale efficiently. `MERGE INTO` combines the `INSERT`, `UPDATE`, and `DELETE` operations into one transaction. This reduces the processing overhead in your data pipeline and takes less SQL to write. For more information, see Update Iceberg table data and MERGE INTO.
- **CTAS and VIEW support** – Use the `CREATE TABLE AS SELECT` (CTAS) and `CREATE VIEW` statements with Iceberg tables. For more information, see CREATE TABLE AS and CREATE VIEW and CREATE DIALECT VIEW.
- **VACUUM support** – You can use the `VACUUM` statement to optimize your data lake by deleting snapshots and data that are no longer required. You can use this feature to improve read performance and meet regulatory requirements like GDPR. For more information, see Optimize Iceberg tables and VACUUM.

These new features require Athena engine version 3 and are available in all Regions where Athena is supported. You can use them with the Athena console, drivers, or API.

For information about using Iceberg in Athena, see Query Apache Iceberg tables.

November 14, 2022

Published on 2022-11-14

Amazon Athena now supports IPv6 endpoints for inbound connections that you can use to invoke Athena functions over IPv6. You can use this feature to meet IPv6 compliance requirements. It also removes the need for additional networking equipment to handle address translation between IPv4 and IPv6.

To use this feature, configure your applications to use the new Athena dual-stack endpoints, which support both IPv4 and IPv6. Dual-stack endpoints use the format `athena.region.api.aws`. For example, the dual-stack endpoint in the US East (N. Virginia) Region is `athena.us-east-1.api.aws`.

When you make a request to a dual-stack Athena endpoint, the endpoint resolves to an IPv6 or an IPv4 address depending on the protocol used by your network and client. To connect programmatically to an AWS service, you can use the AWS CLI or AWS SDK to specify an endpoint.

For more information on service endpoints, see AWS service endpoints. To learn more about Athena's service endpoints, see Amazon Athena endpoints and quotas in the AWS documentation.

You can use the new Athena dual-stack endpoints for inbound connections at no additional cost. Dual-stack endpoints are generally available in all AWS Regions.

November 11, 2022

Published on 2022-11-11

Athena announces the following fixes and improvements.

- **Expanded Lake Formation fine-grained access control** – You can now use AWS Lake Formation fine-grained access control policies in Athena queries for data stored in any supported file or table format. You can use fine-grained access control in Lake Formation to restrict access to data in query results using data filters to achieve column-level, row-level, and cell-level security. Supported table formats in Athena include Apache Iceberg, Apache Hudi, and Apache Hive. Expanded fine-grained access control is available in all regions supported by Athena. The expanded table and file format support requires Athena engine version 3, which offers new features and improved query performance, but does not change how you set up fine-grained access control policies in Lake Formation.

  Use of this expanded fine-grained access control in Athena has the following considerations:

  - **EXPLAIN** – Row or cell filtering information defined in Lake Formation and query statistics information are not shown in the output of `EXPLAIN` and `EXPLAIN ANALYZE`. For information about `EXPLAIN` in Athena, see Using EXPLAIN and EXPLAIN ANALYZE in Athena.
  - **External Hive metastores** – Apache Hive hidden columns cannot be used for fine-grained access control filtering, and Apache Hive hidden system tables are not supported by fine-

grained access control. For more information, see Considerations and limitations in the topic Use an external Hive metastore.

- **Query statistics** – Stage-level input and output row count and data size information are not shown in Athena query statistics when a query has row-level filters defined in Lake Formation. For information about seeing statistics for Athena queries, see View statistics and execution details for completed queries and GetQueryRuntimeStatistics.
- **Workgroups** – Users in the same Athena workgroup can see the data that Lake Formation fine-grained access control has configured to be accessible to the workgroup. For information about using Athena to query data registered with Lake Formation, see Use Athena to query data registered with AWS Lake Formation.

For information about using fine-grained access control in Lake Formation, see Manage fine-grained access control using AWS Lake Formation in the *AWS Big Data Blog*.

- **Athena Federated Query** – Athena Federated Query now preserves the original casing of field names in `struct` objects. Previously, `struct` field names were automatically made lower case.

November 8, 2022

Published on 2022-11-08

You can now use the query result reuse caching feature to accelerate repeat queries in Athena. A repeat query is a SQL query identical to one submitted just recently that produces the same results. When you need to run identical multiple queries, result reuse caching can decrease the time required to produce results. Result reuse caching also lowers costs by reducing the number of bytes scanned.

For more information, see Reuse query results in Athena.

October 13, 2022

Published on 2022-10-13

Athena announces Athena engine version 3.

Athena has upgraded its SQL query engine to include the latest features from the Trino open source project. In addition to supporting all the features of Athena engine version 2, Athena engine version 3 includes over 50 new SQL functions, 30 new features, and more than 90 query performance improvements. With today's launch, Athena is also introducing a continuous integration approach to open source software management that improves currency with the Trino and Presto projects so that you get faster access to community improvements, integrated and tuned within the Athena engine.

For more information, see Athena engine version 3.

October 10, 2022

Published on 2022-10-10

Athena releases JDBC driver version 2.0.33. The JDBC 2.0.33 driver includes the following changes:

- New driver version, JDBC version, and plugin name properties were added to the user-agent string in the credentials provider class.
- Error messages were corrected and necessary information added.
- Prepared statements are now deallocated if the connection is closed or the Athena prepared statement execution fails.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

September 23, 2022

Published on 2022-09-26

The Amazon Athena Neptune connector now supports case insensitive matching on column and table names.

- The Neptune data source connector can resolve column names on Neptune tables that use casing even if the column names are all lower cased in the table in AWS Glue. To enable this behavior, set the `enable_caseinsensitivematch` environment variable to `true` on the Neptune connector Lambda function.
- Because AWS Glue supports only lower case table names, when you create a AWS Glue table for Neptune, specify the AWS Glue table parameter `"glabel" = table_name`.

For more information about the Neptune connector, see Amazon Athena Neptune connector.

September 13, 2022

Published on 2022-09-13

Athena announces the following fixes and improvements.

- **External Hive metastore** – Athena now returns NULL instead of throwing an exception when a WHERE clause includes a partition that doesn't exist in an external Hive metastore (EHMS). The new behavior matches that of the AWS Glue Data Catalog.
- **Parameterized queries** – Values in parameterized queries can now be cast to the DOUBLE data type.
- **Apache Iceberg** – Write operations to Iceberg tables now succeed when Object Lock is enabled on an Amazon S3 bucket.

August 31, 2022

Published on 2022-08-31

Amazon Athena announces availability of Athena and its features in the Asia Pacific (Jakarta) Region.

This release expands Athena's availability in Asia Pacific to include Asia Pacific (Hong Kong), Asia Pacific (Jakarta), Asia Pacific (Mumbai), Asia Pacific (Osaka), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), and Asia Pacific (Tokyo). For a complete list of AWS services available in these and other Regions, refer to the AWS Regional Services List.

August 23, 2022

Published on 2022-08-23

Release v2022.32.1 of the Athena Query Federation SDK includes the following changes:

- Added support to the Amazon Athena Oracle data source connector for SSL based connections to Amazon RDS instances. Support is limited to the Transport Layer Security (TLS) protocol and to authentication of the server by the client. Because mutual authentication it is not supported in Amazon RDS, the update does not include support for mutual authentication.

  For more information, see Amazon Athena Oracle connector.

August 3, 2022

Published on 2022-08-03

Athena releases JDBC driver version 2.0.32. The JDBC 2.0.32 driver includes the following changes:

- The `User-Agent` string sent to the Athena SDK has been extended to contain the driver version, JDBC specification version, and the name of the authentication plugin.
- Fixed a `NullPointerException` that was thrown when no value was provided for the `CheckNonProxyHost` parameter.
- Fixed an issue with `login_url` parsing in the BrowserSaml authentication plugin.
- Fixed a proxy host issue that occurred when the `UseProxyforIdp` parameter was set to `true`.

For more information, and to download the new drivers, release notes, and documentation, see [Connect to Amazon Athena with JDBC](#).

August 1, 2022

Published on 2022-08-01

Athena announces improvements to the Athena Query Federation SDK and Athena prebuilt data source connectors. The improvements include the following:

- **Struct parsing** – Fixed a `GlueFieldLexer` parsing issue in the Athena Query Federation SDK that prevented certain complicated structs from displaying all of their data. This issue affected connectors built on the Athena Query Federation SDK.
- **AWS Glue tables** – Added additional support for the `set` and `decimal` column types in AWS Glue tables.
- **DynamoDB connector** – Added the ability to ignore casing on DynamoDB attribute names. For more information, see `disable_projection_and_casing` in the [Parameters](#) section of the [Amazon Athena DynamoDB connector](#) page.

For more information, see [Release v2022.30.2 of Athena Query Federation](#) on GitHub.

July 21, 2022

Published on 2022-07-21

You can now analyze and debug your queries using performance metrics and interactive, visual query analysis tools in the Athena console. The query performance data and execution details can help you identify bottlenecks in queries, inspect the operators and statistics for each stage of a query, trace the volume of data flowing between stages, and validate the impact of query predicates. You can now:

- Access the distributed and logical execution plan for your query in a single click.
- Explore the operations at each stage before the stage is run.
- Visualize the performance of completed queries with metrics for time spent in the queuing, planning, and execution stages.
- Get information about the number of rows and amount of source data processed and output by your query.
- See granular execution details for your queries presented in context and formatted as an interactive graph.
- Use precise, stage-level execution details to understand the flow of data through your query.
- Analyze query performance data programmatically using new APIs to get query runtime statistics, also released today.

To learn how to use these capabilities on your queries, watch the video tutorial Optimize Amazon Athena Queries with New Query Analysis Tools on the AWS YouTube channel.

For documentation, see View execution plans for SQL queries and View statistics and execution details for completed queries.

July 11, 2022

Published on 2022-07-11

You can now run parameterized queries directly from the Athena console or API without preparing SQL statements in advance.

When you run queries in the Athena console that have parameters in the form of question marks, the user interface now prompts you to enter values for the parameters directly. This eliminates the need to modify literal values in the query editor every time you want to run the query.

If you use the enhanced query execution API, you can now provide the execution parameters and their values in a single call.

For more information, see Use parameterized queries in this user guide and the AWS Big Data Blog post Use Amazon Athena parameterized queries to provide data as a service.

July 8, 2022

Published on 2022-07-08

Athena announces the following fixes and improvements.

- Fixed an issue with DATE column conversion handling for SageMaker AI endpoints (UDF) that was causing query failures.

June 6, 2022

Published on 2022-06-06

Athena releases JDBC driver version 2.0.31. The JDBC 2.0.31 driver includes the following changes:

- **log4j dependency issue** – Resolved a Cannot find driver class error message caused by a log4j dependency.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

May 25, 2022

Published on 2022-05-25

Athena announces the following fixes and improvements.

- **Iceberg support**
  - Introduced support for cross-region queries. Now you can query Iceberg tables in an AWS Region that is different from the AWS Region that you are using. Cross-region querying is not supported in the China Regions.
  - Introduced support for server side encryption configuration. Now you can use SSE-S3/SSE-KMS to encrypt data from Iceberg write operations in Amazon S3.

  For more information about using Apache Iceberg in Athena, see Query Apache Iceberg tables.

- **JDBC 2.0.30 driver release**

  The JDBC 2.0.30 driver for Athena has the following improvements:

  - Fixes a data race issue that affected parameterized prepared statements.
  - Fixes an application start up issue that occurred in Gradle build environments.

  To download the JDBC 2.0.30 driver, release notes, and documentation, see Connect to Amazon Athena with JDBC.

May 6, 2022

Published on 2022-05-06

Released the JDBC 2.0.29 and ODBC 1.1.17 drivers for Athena.

These drivers include the following changes:

- Updated the SAML plugin browser launch process.

For more information about these changes, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

April 22, 2022

Published on 2022-04-22

Athena announces the following fixes and improvements.

- Fixed an issue in the partition indices and filtering feature with the partition cache that occurred when the following conditions were met:
  - The `partition_filtering.enabled` key was set to `true` in the AWS Glue table properties for a table.
  - The same table was used multiple times with different partition filter values.

April 21, 2022

Published on 2022-04-21

You can now use Amazon Athena to run federated queries on new data sources, including Google BigQuery, Azure Synapse, and Snowflake. New data source connectors include:

- Azure Data Lake Storage (ADLS) Gen2
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- Google BigQuery
- Hortonworks
- Microsoft SQL Server
- Oracle
- SAP HANA (Express Edition)
- Snowflake
- Teradata

For a complete list of data sources supported by Athena, see Available data source connectors.

To make it easier to browse the available sources and connect to your data, you can now search, sort, and filter the available connectors from an updated **Data Sources** screen in the Athena console.

To learn about querying federated sources, see Use Amazon Athena Federated Query and Run federated queries.

April 13, 2022

Published on 2022-04-13

Athena releases JDBC driver version 2.0.28. The JDBC 2.0.28 driver includes the following changes:

- **JWT support** – The driver now supports JSON web tokens (JWT) for authentication. For information about using JWT with the JDBC driver, see the installation and configuration guide, downloadable from the JDBC driver page.
- **Updated Log4j libraries** – The JDBC driver now uses the following Log4j libraries:
  - Log4j-api 2.17.1 (previously 2.17.0)
  - Log4j-core 2.17.1 (previously 2.17.0)

  - Log4j-jcl 2.17.2
- **Other improvements –** The new driver also includes the following improvements and bug fixes:
  - The Athena prepared statements feature is now available through JDBC. For information about prepared statements, see Use parameterized queries.
  - Athena JDBC SAML federation is now functional for the China Regions.
  - Additional minor improvements.

For more information, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC.

March 30, 2022

Published on 2022-03-30

Athena announces the following fixes and improvements.

- **Cross-region querying –** You can now use Athena to query data located in an Amazon S3 bucket across AWS Regions including Asia Pacific (Hong Kong), Middle East (Bahrain), Africa (Cape Town), and Europe (Milan). Cross-region querying is not supported in the China Regions.
  - For a list of AWS Regions in which Athena is available, see Amazon Athena endpoints and quotas.
  - For information about enabling an AWS Region that is disabled by default, see Enabling a Region.
  - For information about querying across Regions, see Query across regions.

March 18, 2022

Published on 2022-03-18

Athena announces the following fixes and improvements.

- **Dynamic filtering –** Dynamic filtering has been improved for integer columns by efficiently applying the filter to each record of a corresponding table.
- **Iceberg –** Fixed an issue that caused failures when writing Iceberg Parquet files larger than 2GB.
- **Uncompressed output –** CREATE TABLE statements now support writing uncompressed files. To write uncompressed files, use the following syntax:

  - CREATE TABLE (text file or JSON) – In `TBLPROPERTIES`, specify `write.compression = NONE`.
  - CREATE TABLE (Parquet) – In `TBLPROPERTIES`, specify `parquet.compression = UNCOMPRESSED`.
  - CREATE TABLE (ORC) – In `TBLPROPERTIES`, specify `orc.compress = NONE`.
- **Compression** – Fixed an issue with inserts for text file tables that created files compressed in one format but used another compression format file extension when non-default compression methods were used.
- **Avro** – Fixed issues that occurred when reading decimals of the fixed type from Avro files.

March 2, 2022

Published on 2022-03-02

Athena announces the following features and improvements.

- You can now grant the Amazon S3 bucket owner full control access over query results when ACLs are enabled for the query result bucket. For more information, see Specify a query result location.
- You can now update existing named queries. For more information, see Use saved queries.

February 23, 2022

Published on 2022-02-23

Athena announces the following fixes and performance improvements.

- Memory handling improvements to enhance performance and reduce memory errors.
- Athena now reads ORC timestamp columns with time zone information stored in stripe footers and writes ORC files with time zone (UTC) in footers. This only impacts the behavior of ORC timestamp reads if the ORC file to be read was created in a non-UTC time zone environment.
- Fixed incorrect symlink table size estimates that resulted in suboptimal query plans.
- Lateral exploded views can now be queried in the Athena console from Hive metastore data sources.
- Improved Amazon S3 read error messages to include more detailed Amazon S3 error code information.

- Fixed an issue that caused output files in ORC format to become incompatible with Apache Hive 3.1.
- Fixed an issue that caused table names with quotes to fail in certain DML and DDL queries.

February 15, 2022

Published on 2022-02-15

Amazon Athena has increased the active DML query quota in all AWS Regions. Active queries include both running and queued queries. With this change, you can now have more DML queries in an active state than before.

For information about Athena service quotas, see Service Quotas. For the query quotas in the Region where you use Athena, see Amazon Athena endpoints and quotas in the *AWS General Reference*.

To monitor your quota usage, you can use CloudWatch usage metrics. Athena publishes the `ActiveQueryCount` metric in the `AWS/Usage` namespace. For more information, see Monitor Athena usage metrics with CloudWatch.

After reviewing your usage, you can use the Service Quotas console to request a quota increase. If you previously requested a quota increase for your account, your requested quota still applies if it exceeds the new default active DML query quota. Otherwise, all accounts use the new default.

February 14, 2022

Published on 2022-02-14

This release adds the `ErrorType` subfield to the AthenaError response object in the Athena GetQueryExecution API action.

While the existing `ErrorCategory` field indicates the general source of a failed query (system, user, or other), the new `ErrorType` field provides more granular information about the error that occurred. Combine the information from both fields to gain insight into the causes of query failure.

For more information, see Athena error catalog.

February 9, 2022

Published on 2022-02-09

The old Athena console is no longer available. Athena's new console supports all of the features of the earlier console, but with an easier to use, modern interface and includes new features that improve the experience of developing queries, analyzing data, and managing your usage. To use the new Athena console, visit https://console.aws.amazon.com/athena/.

February 8, 2022

Published on 2022-02-08

Expected bucket owner – As an added security measure, you can now optionally specify the AWS account ID that you expect to be the owner of your query results output location bucket in Athena. If the account ID of the query results bucket owner does not match the account ID that you specify, attempts to output to the bucket will fail with an Amazon S3 permissions error. You can make this setting at the client or workgroup level.

For more information, see Specify a query result location.

January 28, 2022

Published on 2022-01-28

Athena announces the following engine feature enhancements.

- **Apache Hudi –** Snapshot queries on Hudi Merge on Read (MoR) tables can now read timestamp columns that have the `INT64` data type.
- **UNION queries –** Performance improvement and data scan reduction for certain `UNION` queries that scan the same table multiple times.
- **Disjunct queries –** Performance improvement for queries that have only disjunct values for each partition column on the filter.
- **Partition projection enhancements**
  - Multiple disjunct values are now allowed on the filter condition for columns of the `injected` type. For more information, see Injected type.
  - Performance improvement for columns of string-based types like `CHAR` or `VARCHAR` that have only disjunct values on the filter.

January 13, 2022

Published on 2022-01-13

Released the JDBC 2.0.27 and ODBC 1.1.15 drivers for Athena.

The JDBC 2.0.27 driver includes the following changes:

- The driver has been updated to retrieve external catalogs.
- The extended driver version number is now included in the `user-agent` string as part of the Athena API call.

The ODBC 1.1.15 driver includes the following changes:

- Corrects an issue with second calls to `SQLParamData()`.

For more information about these changes, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

Athena release notes for 2021

November 26, 2021

Published on 2021-11-26

Athena announces the public preview of Athena ACID transactions, which add write, delete, update, and time travel operations to Athena's SQL data manipulation language (DML). Athena ACID transactions enable multiple concurrent users to make reliable, row-level modifications to Amazon S3 data. Built on the Apache Iceberg table format, Athena ACID transactions are compatible with other services and engines such as Amazon EMR and Apache Spark that also support the Iceberg table formats.

Athena ACID transactions and familiar SQL syntax simplify updates to your business and regulatory data. For example, to respond to a data erasure request, you can perform a SQL `DELETE` operation. To make manual record corrections, you can use a single `UPDATE` statement. To recover data that was recently deleted, you can issue time travel queries using a `SELECT` statement. Athena transactions are available through Athena's console, API operations, and ODBC and JDBC drivers.

For more information, see Use Athena ACID transactions.

November 24, 2021

Published on 2021-11-24

Athena announces support for reading and writing ZStandard compressed ORC, Parquet, and textfile data. Athena uses ZStandard compression level 3 when writing ZStandard compressed data.

For information about data compression in Athena, see Use compression in Athena.

November 22, 2021

Published on 2021-11-22

You can now manage AWS Step Functions workflows from the Amazon Athena console, making it easier to build scalable data processing pipelines, execute queries based on custom business logic, automate administrative and alerting tasks, and more.

Step Functions is now integrated with Athena's upgraded console, and you can use it to view an interactive workflow diagram of your state machines that invoke Athena. To get started, select **Workflows** from the left navigation panel. If you have existing state machines with Athena queries, select a state machine to view an interactive diagram of the workflow. If you are new to Step Functions, you can get started by launching a sample project from the Athena console and customizing it to suit your use cases.

For more information, see Build and orchestrate ETL pipelines using Amazon Athena and AWS Step Functions, or consult the Step Functions documentation.

November 18, 2021

Published on 2021-11-18

Athena announces new features and improvements.

- Support for spill-to-disk for aggregation queries that contain `DISTINCT`, `ORDER BY`, or both, as in the following example:

```
SELECT array_agg(orderstatus ORDER BY orderstatus)
FROM orders
```

```
GROUP BY orderpriority, custkey
```

- Addressed memory handling issues for queries that use `DISTINCT`. To avoid error messages like Query exhausted resources at this scale factor when you use `DISTINCT` queries, choose columns that have a low cardinality for `DISTINCT`, or reduce the data size of the query.
- In `SELECT COUNT(*)` queries that do not specify a specific column, improved performance and memory usage by keeping only the count without row buffering.
- Introduced the following string functions.
  - `translate(source, from, to)` – Returns the `source` string with the characters found in the `from` string replaced by the corresponding characters in the `to` string. If the `from` string contains duplicates, only the first is used. If the `source` character does not exist in the `from` string, the `source` character is copied without translation. If the index of the matching character in the `from` string is greater than the length of the `to` string, the character is omitted from the resulting string.
  - `concat_ws(string0, array(varchar))` – Returns the concatenation of elements in the array using `string0` as a separator. If `string0` is null, then the return value is null. Any null values in the array are skipped.
- Fixed a bug in which queries failed when trying to access a missing subfield in a `struct`. Queries now return a null for the missing subfield.
- Fixed an issue with inconsistent hashing for the decimal data type.
- Fixed an issue that caused exhausted resources when there were too many columns in a partition.

November 17, 2021

Published on 2021-11-17

Amazon Athena now supports partition indexing to accelerate queries on partitioned tables in the AWS Glue Data Catalog.

When querying partitioned tables, Athena retrieves and filters the available table partitions to the subset relevant to your query. As new data and partitions are added, more time is required to process the partitions and query runtime can increase. To optimize partition processing and improve query performance on highly partitioned tables, Athena now supports AWS Glue partition indexes.

For more information, see Optimize queries with AWS Glue partition indexing and filtering.

November 16, 2021

Published on 2021-11-16

The new and improved Amazon Athena console is now generally available in the AWS commercial and GovCloud regions where Athena is available. Athena's new console supports all of the features of the earlier console, but with an easier to use, modern interface and includes new features that improve the experience of developing queries, analyzing data, and managing your usage. You can now:

- Rearrange, navigate to, or close multiple query tabs from a redesigned query tab bar.
- Read and edit queries more easily with improved SQL and text formatting.
- Copy query results to your clipboard in addition to downloading the full result set.
- Sort your query history, saved queries, and workgroups, and choose which columns to show or hide.
- Use a simplified interface to configure data sources and workgroups in fewer clicks.
- Set preferences for displaying query results, query history, line wrapping, and more.
- Increase your productivity with new and improved keyboard shortcuts and embedded product documentation.

With today's announcement, the redesigned console is now the default. To tell us about your experience, choose **Feedback** in the bottom-left corner of the console.

If desired, you may use the earlier console by logging into your AWS account, choosing Amazon Athena, and deselecting **New Athena experience** from the navigation panel on the left.

November 12, 2021

Published on 2021-11-12

You can now use Amazon Athena to run federated queries on data sources located in an AWS account other than your own. Until today, querying this data required the data source and its connector to use the same AWS account as the user that queried the data.

As a data administrator, you can enable cross-account federated queries by sharing your data connector with a data analyst's account. As a data analyst, you can add a data connector that a data administrator has shared with you to your account. Configuration changes to the connector in the originating account apply automatically to the shared connector.

For information about enabling cross-account federated queries, see Enable cross-account federated queries. To learn about querying federated sources, see Use Amazon Athena Federated Query and Run federated queries.

November 2, 2021

Published on 2021-11-02

You can now use the `EXPLAIN ANALYZE` statement in Athena to view the distributed execution plan and cost of each operation for your SQL queries.

For more information, see Using EXPLAIN and EXPLAIN ANALYZE in Athena.

October 29, 2021

Published on 2021-10-29

Athena releases JDBC 2.0.25 and ODBC 1.1.13 drivers and announces features and improvements.

JDBC and ODBC Drivers

Released JDBC 2.0.25 and ODBC 1.1.13 drivers for Athena. Both drivers offer support for browser SAML multi-factor authentication, which can be configured to work with any SAML 2.0 provider.

The JDBC 2.0.25 driver includes the following changes:

- Support for browser SAML authentication. The driver includes a browser SAML plugin which can be configured to work with any SAML 2.0 provider.
- Support for AWS Glue API calls. You can use the `GlueEndpointOverride` parameter to override the AWS Glue endpoint.
- Changed the `com.simba.athena.amazonaws` class path to `com.amazonaws`.

The ODBC 1.1.13 driver includes the following changes:

- Support for browser SAML authentication. The driver includes a browser SAML plugin which can be configured to work with any SAML 2.0 provider. For an example of how to use the browser SAML plugin with the ODBC driver, see Configure single sign-on using ODBC, SAML 2.0, and the Okta Identity Provider.

- You can now configure the role session duration when you use ADFS, Azure AD, or Browser Azure AD for authentication.

For more information about these and other changes, and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

Features and Improvements

Athena announces the following features and improvements.

- A new optimization rule has been introduced to avoid duplicate table scans in certain cases.

October 4, 2021

Published on 2021-10-04

Athena announces the following features and improvements.

- **SQL OFFSET** – The SQL `OFFSET` clause is now supported in `SELECT` statements. For more information, see SELECT.
- **CloudWatch usage metrics** – Athena now publishes the `ActiveQueryCount` metric in the `AWS/Usage` namespace. For more information, see Monitor Athena usage metrics with CloudWatch.
- **Query planning** – Fixed a bug that could in rare cases cause query planning timeouts.

September 16, 2021

Published on 2021-09-16

Athena announces the following new features and improvements.

Features

- Added support for specifying text file and JSON compression in CTAS using the `write_compression` table property. You can also specify the `write_compression` property in CTAS for the Parquet and ORC formats. For more information, see CTAS table properties.
- The BZIP2 compression format is now supported for writing text file and JSON files. For more information about the compression formats in Athena, see Use compression in Athena.

Improvements

- Fixed a bug in which identity information failed to be sent to the UDF Lambda function.
- Fixed a predicate pushdown issue with disjunct filter conditions.
- Fixed a hashing issue for decimal types.
- Fixed an unnecessary statistics collection issue.
- Removed an inconsistent error message.
- Improved broadcast join performance by applying dynamic partition pruning in the worker node.
- For federated queries:
  - Altered configuration to reduce the occurrence of `CONSTRAINT_VIOLATION` errors in federated queries.

September 15, 2021

Published on 2021-09-15

You can now use a redesigned Amazon Athena console (Preview). A new Athena JDBC driver has been released.

Athena Console Preview

You can now use a redesigned [Amazon Athena](https://console.aws.amazon.com/athena/) console (Preview) from any AWS Region where Athena is available. The new console supports all of the features of the existing console, but from an easier to use, modern interface.

To switch to the new [console](https://console.aws.amazon.com/athena/), log into your AWS account and choose Amazon Athena. From the AWS console navigation bar, choose **Switch to the new console**. To return to the default console, deselect **New Athena experience** from the navigation panel on the left.

Get started with the new [console](https://console.aws.amazon.com/athena/) today. Choose **Feedback** in the bottom-left corner to tell us about your experience.

Athena JDBC Driver 2.0.24

Athena announces availability of JDBC driver version 2.0.24 for Athena. This release updates proxy support for all credentials providers. The driver now supports proxy authentication for all hosts that are not supported by the `NonProxyHosts` connection property.

As a convenience, this release includes downloads of the JDBC driver both with and without the AWS SDK. This JDBC driver version allows you to have both the AWS-SDK and the Athena JDBC driver embedded in project.

For more information and to download the new driver, release notes, and documentation, see [Connect to Amazon Athena with JDBC](#).

August 31, 2021

Published on 2021-08-31

Athena announces the following feature enhancements and bug fixes.

- **Athena federation enhancements** – Athena has added support to map types and better support for complex types as part of the [Athena Query Federation SDK](#). This version also includes some memory enhancements and performance optimizations.
- **New error categories** – Introduced the `USER` and `SYSTEM` error categories in error messages. These categories help you distinguish between errors that you can fix yourself (`USER`) and errors that can require assistance from Athena support (`SYSTEM`).
- **Federated query error messaging** – Updated `USER_ERROR` categorizations for federated query related errors.
- **JOIN** – Fixed spill-to-disk related bugs and memory issues to enhance performance and reduce memory errors in `JOIN` operations.

August 12, 2021

Published on 2021-08-12

Released the ODBC 1.1.12 driver for Athena. This version corrects issues related to `SQLPrepare()`, `SQLGetInfo()`, and `EndpointOverride`.

To download the new driver, release notes, and documentation, see [Connect to Amazon Athena with ODBC](#).

August 6, 2021

Published on 2021-08-06

Amazon Athena announces availability of Athena and its features in the Asia Pacific (Osaka) Region.

This release expands Athena's availability in Asia Pacific to include Asia Pacific (Hong Kong), Asia Pacific (Mumbai), Asia Pacific (Osaka), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), and Asia Pacific (Tokyo). For a complete list of AWS services available in these and other Regions, refer to the AWS Regional Services List.

August 5, 2021

Published on 2021-08-05

You can use the UNLOAD statement to write the output of a SELECT query to the PARQUET, ORC, AVRO, and JSON formats.

For more information, see UNLOAD.

July 30, 2021

Published on 2021-07-30

Athena announces the following feature enhancements and bug fixes.

- **Dynamic filtering and partition pruning –** Improvements increase performance and reduce the amount of data scanned in certain queries, as in the following example.

  This example assumes that Table_B is an unpartitioned table that has file sizes that add up to less than 20 MB. For queries like this, less data is read from Table_A and the query completes more quickly.

  ```
  SELECT *
  FROM Table_A
  JOIN Table_B ON Table_A.date = Table_B.date
  WHERE Table_B.column_A = "value"
  ```
- **ORDER BY with LIMIT, DISTINCT with LIMIT –** Performance improvements to queries that use ORDER BY or DISTINCT followed by a LIMIT clause.
- **S3 Glacier Deep Archive files –** When Athena queries a table that contains a mix of S3 Glacier Deep Archive files and non-S3 Glacier files, Athena now skips the S3 Glacier Deep Archive files for you. Previously, you were required to manually move these files from the query location or the query would fail. If you want to use Athena to query objects in S3 Glacier Deep Archive

storage, you must restore them. For more information, see Restoring an archived object in the *Amazon S3 User Guide*.

- Fixed a bug in which empty files created by the CTAS `bucketed_by` table property were not encrypted correctly.

July 21, 2021

Published on 2021-07-21

With the July 2021 release of Microsoft Power BI Desktop, you can create reports and dashboards using a native data source connector for Amazon Athena. The connector for Amazon Athena is available as a standard connector in Power BI, supports DirectQuery, and enables analysis on large datasets and content refresh through Power BI Gateway.

Because the connector uses your existing ODBC data source name (DSN) to connect to and run queries on Athena, it requires the Athena ODBC driver. To download the latest ODBC driver, see Connect to Amazon Athena with ODBC.

For more information, see Use the Amazon Athena Power BI connector.

July 16, 2021

Published on 2021-07-16

Amazon Athena has updated its integration with Apache Hudi. Hudi is an open-source data management framework used to simplify incremental data processing in Amazon S3 data lakes. The updated integration enables you to use Athena to query Hudi 0.8.0 tables managed through Amazon EMR, Apache Spark, Apache Hive or other compatible services. In addition, Athena now supports two additional features: snapshot queries on Merge-on-Read (MoR) tables and read support on bootstrapped tables.

Apache Hudi provides record-level data processing that can help you simplify development of Change Data Capture (CDC) pipelines, comply with GDPR-driven updates and deletes, and better manage streaming data from sensors or devices that require data insertion and event updates. The 0.8.0 release makes it easier to migrate large Parquet tables to Hudi without copying data so you can query and analyze them through Athena. You can use Athena's new support for snapshot queries to have near real-time views of your streaming table updates.

To learn more about using Hudi with Athena, see Query Apache Hudi datasets.

July 8, 2021

Published on 2021-07-08

Released the ODBC 1.1.11 driver for Athena. The ODBC driver can now authenticate the connection using a JSON Web Token (JWT). On Linux, the default value for the Workgroup property has been set to Primary.

For more information and to download the new driver, release notes, and documentation, see Connect to Amazon Athena with ODBC.

July 1, 2021

Published on 2021-07-01

On July 1, 2021, special handling of preview workgroups ended. While `AmazonAthenaPreviewFunctionality` workgroups retain their name, they no longer have special status. You can continue to use `AmazonAthenaPreviewFunctionality` workgroups to view, modify, organize, and run queries. However, queries that use features that were formerly in preview are now subject to standard Athena billing terms and conditions. For billing information, see Amazon Athena pricing.

June 23, 2021

Published on 2021-06-23

Released JDBC 2.0.23 and ODBC 1.1.10 drivers for Athena. Both drivers offer improved read performance and support EXPLAIN statements and parameterized queries.

`EXPLAIN` statements show the logical or distributed execution plan of a SQL query. Parameterized queries enable the same query to be used multiple times with different values supplied at run time.

The JDBC release also adds support for Active Directory Federation Services 2019 and a custom endpoint override option for AWS STS. The ODBC release fixes an issue with IAM profile credentials.

For more information and to download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

May 12, 2021

Published on 2021-05-12

You can now use Amazon Athena to register an AWS Glue catalog from an account other than your own. After you configure the required IAM permissions for AWS Glue, you can use Athena to run cross-account queries.

For more information, see Register a Data Catalog from another account and Configure cross-account access to AWS Glue data catalogs.

May 10, 2021

Published on 2021-05-10

Released ODBC driver version 1.1.9.1001 for Athena. This version fixes an issue with the `BrowserAzureAD` authentication type when using Azure Active Directory (AD).

To download the new drivers, release notes, and documentation, see Connect to Amazon Athena with ODBC.

May 5, 2021

Published on 2021-05-05

You can now use the Amazon Athena Vertica connector in federated queries to query Vertica data sources from Athena. For example, you can run analytical queries over a data warehouse on Vertica and a data lake in Amazon S3.

To deploy the Athena Vertica connector, visit the AthenaVerticaConnector page in the AWS Serverless Application Repository.

The Amazon Athena Vertica connector exposes several configuration options through Lambda environment variables. For information about configuration options, parameters, connection strings, deployment, and limitations, see Amazon Athena Vertica connector.

For in-depth information about using the Vertica connector, see Querying a Vertica data source in Amazon Athena using the Athena Federated Query SDK in the *AWS Big Data Blog*.

April 30, 2021

Published on 2021-04-30

Released drivers JDBC 2.0.21 and ODBC 1.1.9 for Athena. Both releases support SAML authentication with Azure Active Directory (AD) and SAML authentication with PingFederate. The

JDBC release also supports parameterized queries. For information about parameterized queries in Athena, see Use parameterized queries.

To download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

April 29, 2021

Published on 2021-04-29

Amazon Athena announces availability of Athena engine version 2 in the China (Beijing) and China (Ningxia) Regions.

April 26, 2021

Published on 2021-04-26

Window value functions in Athena engine version 2 now support `IGNORE NULLS` and `RESPECT NULLS`.

For more information, see Value Functions in the Presto documentation.

April 21, 2021

Published on 2021-04-21

Amazon Athena announces availability of Athena engine version 2 in the Europe (Milan) and Africa (Cape Town) Regions.

April 5, 2021

Published on 2021-04-05

EXPLAIN Statement

You can now use the `EXPLAIN` statement in Athena to view the execution plan for your SQL queries.

For more information, see Using EXPLAIN and EXPLAIN ANALYZE in Athena and Understand Athena EXPLAIN statement results.

SageMaker AI Machine Learning Models in SQL Queries

Machine learning model inference with Amazon SageMaker AI is now generally available for Amazon Athena. Use machine learning models in SQL queries to simplify complex tasks such as anomaly detection, customer cohort analysis, and time-series predictions by invoking a function in a SQL query.

For more information, see Use Machine Learning (ML) with Amazon Athena.

User Defined Functions (UDF)

User defined functions (UDFs) are now generally available for Athena. Use UDFs to leverage custom functions that process records or groups of records in a single SQL query.

For more information, see Query with user defined functions.

March 30, 2021

Published on 2021-03-30

Amazon Athena announces availability of Athena engine version 2 in the Asia Pacific (Hong Kong) and Middle East (Bahrain) Regions.

March 25, 2021

Published on 2021-03-25

Amazon Athena announces availability of Athena engine version 2 in the Europe (Stockholm) Region.

March 5, 2021

Published on 2021-03-05

Amazon Athena announces availability of Athena engine version 2 in the Canada (Central), Europe (Frankfurt), and South America (São Paulo) Regions.

February 25, 2021

Published on 2021-02-25

Amazon Athena announces general availability of Athena engine version 2 in the Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Europe (London), and Europe (Paris) Regions.

Athena release notes for 2020

December 16, 2020

Published on 2020-12-16

Amazon Athena announces availability of Athena engine version 2, Athena Federated Query, and AWS PrivateLink in additional Regions.

Athena engine version 2 and Athena Federated Query

Amazon Athena announces general availability of Athena engine version 2 and Athena Federated Query in the Asia Pacific (Mumbai), Asia Pacific (Tokyo), Europe (Ireland), and US West (N. California) Regions. Athena engine version 2 and federated queries are already available in the US East (N. Virginia), US East (Ohio), and US West (Oregon) Regions.

AWS PrivateLink

AWS PrivateLink for Athena is now supported in the Europe (Stockholm) Region. For information about AWS PrivateLink for Athena, see Connect to Amazon Athena using an interface VPC endpoint.

November 24, 2020

Published on 2020-11-24

Released drivers JDBC 2.0.16 and ODBC 1.1.6 for Athena. These releases support, at the account level, Okta Verify multifactor authentication (MFA). You can also use Okta MFA to configure SMS authentication and Google Authenticator authentication as factors.

To download the new drivers, release notes, and documentation, see Connect to Amazon Athena with JDBC and Connect to Amazon Athena with ODBC.

November 11, 2020

Published on 2020-11-11

Amazon Athena announces general availability in the US East (N. Virginia), US East (Ohio), and US West (Oregon) Regions for Athena engine version 2 and federated queries.

Athena engine version 2

Amazon Athena announces general availability of a new query engine version, Athena engine version 2, in the US East (N. Virginia), US East (Ohio), and US West (Oregon) Regions.

Athena engine version 2 includes performance enhancements and new feature capabilities such as schema evolution support for Parquet format data, additional geospatial functions, support for reading nested schema to reduce cost, and performance enhancements in JOIN and AGGREGATE operations.

- For information about how to upgrade, see Change Athena engine versions.
- For information about testing queries, see Test queries in advance of an engine version upgrade.

Federated SQL Queries

You can now use Athena's federated query in the US East (N. Virginia), US East (Ohio), and US West (Oregon) Regions without using the `AmazonAthenaPreviewFunctionality` workgroup.

Use Federated SQL queries to run SQL queries across relational, non-relational, object, and custom data sources. With federated querying, you can submit a single SQL query that scans data from multiple sources running on premises or hosted in the cloud.

Running analytics on data spread across applications can be complex and time consuming for the following reasons:

- Data required for analytics is often spread across relational, key-value, document, in-memory, search, graph, object, time-series and ledger data stores.
- To analyze data across these sources, analysts build complex pipelines to extract, transform, and load into a data warehouse so that the data can be queried.
- Accessing data from various sources requires learning new programming languages and data access constructs.

Federated SQL queries in Athena eliminate this complexity by allowing users to query the data in-place from wherever it resides. Analysts can use familiar SQL constructs to `JOIN` data across multiple data sources for quick analysis, and store results in Amazon S3 for subsequent use.

Data Source Connectors

To process federated queries, Athena uses Athena Data Source Connectors that run on AWS Lambda. The following open sourced, pre-built connectors were written and tested by Athena. Use them to run SQL queries in Athena on their corresponding data sources.

- CloudWatch
- CloudWatch Metrics
- DocumentDB
- DynamoDB
- OpenSearch
- HBase
- Neptune
- Redis
- Timestream
- TPC Benchmark DS (TPC-DS)

Custom Data Source Connectors

Using Athena Query Federation SDK, developers can build connectors to any data source to enable Athena to run SQL queries against that data source. Athena Query Federation Connector extends the benefits of federated querying beyond AWS provided connectors. Because connectors run on AWS Lambda, you do not have to manage infrastructure or plan for scaling to peak demands.

Next Steps

- To learn more about the federated query feature, see Use Amazon Athena Federated Query.
- To get started with using an existing connector, see Create a data source connection.
- To learn how to build your own data source connector using the Athena Query Federation SDK, see Example Athena Connector on GitHub.

October 22, 2020

Published on 2020-10-22

You can now call Athena with AWS Step Functions. AWS Step Functions can control certain AWS services directly using the Amazon States Language. You can use Step Functions with Athena to start and stop query execution, get query results, run ad-hoc or scheduled data queries, and retrieve results from data lakes in Amazon S3.

For more information, see Call Athena with Step Functions in the *AWS Step Functions Developer Guide*.

July 29, 2020

Published on 2020-07-29

Released JDBC driver version 2.0.13. This release supports using multiple data catalogs registered with Athena, Okta service for authentication, and connections to VPC endpoints.

To download and use the new version of the driver, see Connect to Amazon Athena with JDBC.

July 9, 2020

Published on 2020-07-09

Amazon Athena adds support for querying compacted Hudi datasets and adds the AWS CloudFormation `AWS::Athena::DataCatalog` resource for creating, updating, or deleting data catalogs that you register in Athena.

Querying Apache Hudi Datasets

Apache Hudi is an open-source data management framework that simplifies incremental data processing. Amazon Athena now supports querying the read-optimized view of an Apache Hudi dataset in your Amazon S3-based data lake.

For more information, see Query Apache Hudi datasets.

AWS CloudFormation Data Catalog Resource

To use Amazon Athena's federated query feature to query any data source, you must first register your data catalog in Athena. You can now use the AWS CloudFormation `AWS::Athena::DataCatalog` resource to create, update, or delete data catalogs that you register in Athena.

For more information, see AWS::Athena::DataCatalog in the *AWS CloudFormation User Guide*.

June 1, 2020

Published on 2020-06-01

Using Apache Hive Metastore as a Metacatalog with Amazon Athena

You can now connect Athena to one or more Apache Hive metastores in addition to the AWS Glue Data Catalog with Athena.

To connect to a self-hosted Hive metastore, you need an Athena Hive metastore connector. Athena provides a reference implementation connector that you can use. The connector runs as an AWS Lambda function in your account.

For more information, see Use an external Hive metastore.

May 21, 2020

Published on 2020-05-21

Amazon Athena adds support for partition projection. Use partition projection to speed up query processing of highly partitioned tables and automate partition management. For more information, see Use partition projection with Amazon Athena.

April 1, 2020

Published on 2020-04-01

In addition to the US East (N. Virginia) Region, the Amazon Athena federated query, user defined functions (UDFs), machine learning inference, and external Hive metastore features are now available in preview in the Asia Pacific (Mumbai), Europe (Ireland), and US West (Oregon) Regions.

March 11, 2020

Published on 2020-03-11

Amazon Athena now publishes Amazon EventBridge events for query state transitions. When a query transitions between states -- for example, from Running to a terminal state such as Succeeded or Cancelled -- Athena publishes a query state change event to EventBridge. The event contains information about the query state transition. For more information, see [Monitor Athena query events with EventBridge](#).

March 6, 2020

Published on 2020-03-06

You can now create and update Amazon Athena workgroups by using the AWS CloudFormation `AWS::Athena::WorkGroup` resource. For more information, see [AWS::Athena::WorkGroup](#) in the *AWS CloudFormation User Guide*.

Athena release notes for 2019

November 26, 2019

Published on 2019-12-17

Amazon Athena adds support for running SQL queries across relational, non-relational, object, and custom data sources, invoking machine learning models in SQL queries, User Defined Functions (UDFs) (Preview), using Apache Hive Metastore as a metadata catalog with Amazon Athena (Preview), and four additional query-related metrics.

Federated SQL Queries

Use Federated SQL queries to run SQL queries across relational, non-relational, object, and custom data sources.

You can now use Athena's federated query to scan data stored in relational, non-relational, object, and custom data sources. With federated querying, you can submit a single SQL query that scans data from multiple sources running on premises or hosted in the cloud.

Running analytics on data spread across applications can be complex and time consuming for the following reasons:

- Data required for analytics is often spread across relational, key-value, document, in-memory, search, graph, object, time-series and ledger data stores.
- To analyze data across these sources, analysts build complex pipelines to extract, transform, and load into a data warehouse so that the data can be queried.

- Accessing data from various sources requires learning new programming languages and data access constructs.

Federated SQL queries in Athena eliminate this complexity by allowing users to query the data in-place from wherever it resides. Analysts can use familiar SQL constructs to `JOIN` data across multiple data sources for quick analysis, and store results in Amazon S3 for subsequent use.

Data Source Connectors

Athena processes federated queries using Athena Data Source Connectors that run on AWS Lambda. Use these open sourced data source connectors to run federated SQL queries in Athena across Amazon DynamoDB, Apache HBase, Amazon Document DB, Amazon CloudWatch, Amazon CloudWatch Metrics, and JDBC-compliant relational databases such MySQL, and PostgreSQL under the Apache 2.0 license.

Custom Data Source Connectors

Using Athena Query Federation SDK, developers can build connectors to any data source to enable Athena to run SQL queries against that data source. Athena Query Federation Connector extends the benefits of federated querying beyond AWS provided connectors. Because connectors run on AWS Lambda, you do not have to manage infrastructure or plan for scaling to peak demands.

Preview Availability

Athena federated query is available in preview in the US East (N. Virginia) Region.

Next Steps

- To begin your preview, follow the instructions in the Athena Preview Features FAQ.
- To learn more about the federated query feature, see Using Amazon Athena Federated Query (Preview).
- To get started with using an existing connector, see Create a data source connection.
- To learn how to build your own data source connector using the Athena Query Federation SDK, see Example Athena Connector on GitHub.

Invoking Machine Learning Models in SQL Queries

You can now invoke machine learning models for inference directly from your Athena queries. The ability to use machine learning models in SQL queries makes complex tasks such anomaly detection, customer cohort analysis, and sales predictions as simple as invoking a function in a SQL query.

ML Models

You can use more than a dozen built-in machine learning algorithms provided by Amazon SageMaker, train your own models, or find and subscribe to model packages from AWS Marketplace and deploy on Amazon SageMaker Hosting Services. There is no additional setup required. You can invoke these ML models in your SQL queries from the Athena console, Athena APIs, and through Athena's preview JDBC driver.

Preview Availability

Athena's ML functionality is available today in preview in the US East (N. Virginia) Region.

Next Steps

- To begin your preview, follow the instructions in the Athena Preview Features FAQ.
- To learn more about the machine learning feature, see Using Machine Learning (ML) with Amazon Athena (Preview).

User Defined Functions (UDFs) (Preview)

You can now write custom scalar functions and invoke them in your Athena queries. You can write your UDFs in Java using the Athena Query Federation SDK. When a UDF is used in a SQL query submitted to Athena, it is invoked and run on AWS Lambda. UDFs can be used in both `SELECT` and `FILTER` clauses of a SQL query. You can invoke multiple UDFs in the same query.

Preview Availability

Athena UDF functionality is available in Preview mode in the US East (N. Virginia) Region.

Next Steps

- To begin your preview, follow the instructions in the Athena Preview Features FAQ.
- To learn more, see Querying with User Defined Functions (Preview).
- For example UDF implementations, see Amazon Athena UDF Connector on GitHub.
- To learn how to write your own functions using the Athena Query Federation SDK, see Creating and Deploying a UDF Using Lambda.

Using Apache Hive Metastore as a Metacatalog with Amazon Athena (Preview)

You can now connect Athena to one or more Apache Hive Metastores in addition to the AWS Glue Data Catalog with Athena.

Metastore Connector

To connect to a self-hosted Hive Metastore, you need an Athena Hive Metastore connector. Athena provides a reference implementation connector that you can use. The connector runs as an AWS Lambda function in your account. For more information, see Using Athena Data Connector for External Hive Metastore (Preview).

Preview Availability

The Hive Metastore feature is available in Preview mode in the US East (N. Virginia) Region.

Next Steps

- To begin your preview, follow the instructions in the Athena Preview Features FAQ.
- To learn more about this feature, please visit our Using Athena Data Connector for External Hive Metastore (Preview).

New Query-Related Metrics

Athena now publishes additional query metrics that can help you understand Amazon Athena performance. Athena publishes query-related metrics to Amazon CloudWatch. In this release, Athena publishes the following additional query metrics:

- **Query Planning Time –** The time taken to plan the query. This includes the time spent retrieving table partitions from the data source.
- **Query Queuing Time –** The time that the query was in a queue waiting for resources.
- **Service Processing Time –** The time taken to write results after the query engine finishes processing.
- **Total Execution Time –** The time Athena took to run the query.

To consume these new query metrics, you can create custom dashboards, set alarms and triggers on metrics in CloudWatch, or use pre-populated dashboards directly from the Athena console.

Next Steps

For more information, see Monitoring Athena Queries with CloudWatch Metrics.

November 12, 2019

Published on 2019-12-17

Amazon Athena is now available in the Middle East (Bahrain) Region.

November 8, 2019

Published on 2019-12-17

Amazon Athena is now available in the US West (N. California) Region and the Europe (Paris) Region.

October 8, 2019

Published on 2019-12-17

Amazon Athena now allows you to connect directly to Athena through an interface VPC endpoint in your Virtual Private Cloud (VPC). Using this feature, you can submit your queries to Athena securely without requiring an Internet Gateway in your VPC.

To create an interface VPC endpoint to connect to Athena, you can use the AWS Management Console or AWS Command Line Interface (AWS CLI). For information about creating an interface endpoint, see Creating an Interface Endpoint.

When you use an interface VPC endpoint, communication between your VPC and Athena APIs is secure and stays within the AWS network. There are no additional Athena costs to use this feature. Interface VPC endpoint charges apply.

To learn more about this feature, see Connect to Amazon Athena Using an Interface VPC Endpoint.

September 19, 2019

Published on 2019-12-17

Amazon Athena adds support for inserting new data to an existing table using the `INSERT INTO` statement. You can insert new rows into a destination table based on a `SELECT` query statement that runs on a source table, or based on a set of values that are provided as part of the query statement. Supported data formats include Avro, JSON, ORC, Parquet, and text files.

`INSERT INTO` statements can also help you simplify your ETL process. For example, you can use `INSERT INTO` in a single query to select data from a source table that is in JSON format and write to a destination table in Parquet format.

`INSERT INTO` statements are charged based on the number of bytes that are scanned in the `SELECT` phase, similar to how Athena charges for `SELECT` queries. For more information, see Amazon Athena pricing.

For more information about using `INSERT INTO`, including supported formats, SerDes and examples, see INSERT INTO in the Athena User Guide.

September 12, 2019

Published on 2019-12-17

Amazon Athena is now available in the Asia Pacific (Hong Kong) Region.

August 16, 2019

Published on 2019-12-17

Amazon Athena adds support for querying data in Amazon S3 Requester Pays buckets.

When an Amazon S3 bucket is configured as Requester Pays, the requester, not the bucket owner, pays for the Amazon S3 request and data transfer costs. In Athena, workgroup administrators can now configure workgroup settings to allow workgroup members to query S3 Requester Pays buckets.

For information about how to configure the Requester Pays setting for your workgroup, refer to Create a Workgroup in the Amazon Athena User Guide. For more information about Requester Pays buckets, see Requester Pays Buckets in the Amazon Simple Storage Service Developer Guide.

August 9, 2019

Published on 2019-12-17

Amazon Athena now supports enforcing AWS Lake Formation policies for fine-grained access control to new or existing databases, tables, and columns defined in the AWS Glue Data Catalog for data stored in Amazon S3.

You can use this feature in the following AWS Regions: US East (Ohio), US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland). There are no additional charges to use this feature.

For more information about using this feature, see Use Athena to query data registered with AWS Lake Formation. For more information about AWS Lake Formation, see AWS Lake Formation.

June 26, 2019

Amazon Athena is now available in the Europe (Stockholm) Region. For a list of supported Regions, see AWS Regions and Endpoints.

May 24, 2019

Published on *2019-05-24*

Amazon Athena is now available in the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. For a list of supported Regions, see AWS Regions and Endpoints.

March 05, 2019

Published on *2019-03-05*

Amazon Athena is now available in the Canada (Central) Region. For a list of supported Regions, see AWS Regions and Endpoints. Released the new version of the ODBC driver with support for Athena workgroups. For more information, see the ODBC Driver Release Notes.

To download the ODBC driver version 1.0.5 and its documentation, see Connect to Amazon Athena with ODBC. For information about this version, see the ODBC Driver Release Notes.

To use workgroups with the ODBC driver, set the new connection property, `Workgroup`, in the connection string as shown in the following example:

```
Driver=Simba Athena ODBC
 Driver;AwsRegion=[Region];S3OutputLocation=[S3Path];AuthenticationType=IAM
 Credentials;UID=[YourAccessKey];PWD=[YourSecretKey];Workgroup=[WorkgroupName]
```

For more information, search for "workgroup" in the ODBC Driver Installation and Configuration Guide version 1.0.5. There are no changes to the ODBC driver connection string when you use tags on workgroups. To use tags, upgrade to the latest version of the ODBC driver, which is this current version.

This driver version lets you use Athena API workgroup actions to create and manage workgroups, and Athena API tag actions to add, list, or remove tags on workgroups. Before you begin, make sure that you have resource-level permissions in IAM for actions on workgroups and tags.

For more information, see:

- Use workgroups to control query access and costs and Example workgroup policies.
- Tag Athena resources and Use tag-based IAM access control policies.

If you use the JDBC driver or the AWS SDK, upgrade to the latest version of the driver and SDK, both of which already include support for workgroups and tags in Athena. For more information, see Connect to Amazon Athena with JDBC.

February 22, 2019

Published on *2019-02-22*

Added tag support for workgroups in Amazon Athena. A tag consists of a key and a value, both of which you define. When you tag a workgroup, you assign custom metadata to it. You can add tags to workgroups to help categorize them, using AWS tagging best practices. You can use tags to restrict access to workgroups, and to track costs. For example, create a workgroup for each cost center. Then, by adding tags to these workgroups, you can track your Athena spending for each cost center. For more information, see Using Tags for Billing in the *AWS Billing and Cost Management User Guide*.

You can work with tags by using the Athena console or the API operations. For more information, see Tag Athena resources.

In the Athena console, you can add one or more tags to each of your workgroups, and search by tags. Workgroups are an IAM-controlled resource in Athena. In IAM, you can restrict who can add, remove, or list tags on workgroups that you create. You can also use the `CreateWorkGroup` API operation that has the optional tag parameter for adding one or more tags to the workgroup. To add, remove, or list tags, use `TagResource`, `UntagResource`, and `ListTagsForResource`. For more information, see Use API and AWS CLI tag operations.

To allow users to add tags when creating workgroups, ensure that you give each user IAM permissions to both the `TagResource` and `CreateWorkGroup` API actions. For more information and examples, see Use tag-based IAM access control policies.

There are no changes to the JDBC driver when you use tags on workgroups. If you create new workgroups and use the JDBC driver or the AWS SDK, upgrade to the latest version of the driver and SDK. For information, see Connect to Amazon Athena with JDBC.

February 18, 2019

Published on *2019-02-18*

Added ability to control query costs by running queries in workgroups. For information, see Use workgroups to control query access and costs. Improved the JSON OpenX SerDe used in Athena, fixed an issue where Athena did not ignore objects transitioned to the `GLACIER` storage class, and added examples for querying Network Load Balancer logs.

Made the following changes:

- Added support for workgroups. Use workgroups to separate users, teams, applications, or workloads, and to set limits on amount of data each query or the entire workgroup can process. Because workgroups act as IAM resources, you can use resource-level permissions to control access to a specific workgroup. You can also view query-related metrics in Amazon CloudWatch, control query costs by configuring limits on the amount of data scanned, create thresholds, and trigger actions, such as Amazon SNS alarms, when these thresholds are breached. For more information, see Use workgroups to control query access and costs and Use CloudWatch and EventBridge to monitor queries and control costs.

  Workgroups are an IAM resource. For a full list of workgroup-related actions, resources, and conditions in IAM, see Actions, Resources, and Condition Keys for Amazon Athena in the *Service Authorization Reference*. Before you create new workgroups, make sure that you use workgroup IAM policies, and the AWS managed policy: AmazonAthenaFullAccess.

  You can use workgroups in the console, with workgroup API operations, or with the JDBC driver. For information about creating workgroups, see Create a workgroup. To download the JDBC driver with workgroup support, see Connect to Amazon Athena with JDBC.

  If you use workgroups with the JDBC driver, you must set the workgroup name in the connection string using the `Workgroup` configuration parameter as in the following example:

  ```
  jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
  PWD=<SECRETKEY>;S3OutputLocation=s3://amzn-s3-demo-bucket/<athena-
  output>-<AWSREGION>/;
  Workgroup=<WORKGROUPNAME>;
  ```

There are no changes in the way you run SQL statements or make JDBC API calls to the driver. The driver passes the workgroup name to Athena.

For information about differences introduced with workgroups, see Use Athena workgroup APIs and Troubleshoot workgroup errors.

- Improved the JSON OpenX SerDe used in Athena. The improvements include, but are not limited to, the following:
  - Support for the `ConvertDotsInJsonKeysToUnderscores` property. When set to `TRUE`, it allows the SerDe to replace the dots in key names with underscores. For example, if the JSON dataset contains a key with the name `"a.b"`, you can use this property to define the column name to be `"a_b"` in Athena. The default is `FALSE`. By default, Athena does not allow dots in column names.
  - Support for the `case.insensitive` property. By default, Athena requires that all keys in your JSON dataset use lowercase. Using `WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)` allows you to use case-sensitive key names in your data. The default is `TRUE`. When set to `TRUE`, the SerDe converts all uppercase columns to lowercase.

  For more information, see OpenX JSON SerDe.
- Fixed an issue where Athena returned `"access denied"` error messages, when it processed Amazon S3 objects that were archived to Glacier by Amazon S3 lifecycle policies. As a result of fixing this issue, Athena ignores objects transitioned to the `GLACIER` storage class. Athena does not support querying data from the `GLACIER` storage class.

  For more information, see the section called “Amazon S3 considerations” and Transitioning to the GLACIER Storage Class (Object Archival) in the *Amazon Simple Storage Service User Guide*.
- Added examples for querying Network Load Balancer access logs that receive information about the Transport Layer Security (TLS) requests. For more information, see the section called “Network Load Balancer”.

Athena release notes for 2018

November 20, 2018

Published on *2018-11-20*

Released the new versions of the JDBC and ODBC driver with support for federated access to Athena API with the AD FS and SAML 2.0 (Security Assertion Markup Language 2.0). For details, see the JDBC Driver Release Notes and ODBC Driver Release Notes.

With this release, federated access to Athena is supported for the Active Directory Federation Service (AD FS 3.0). Access is established through the versions of JDBC or ODBC drivers that support SAML 2.0. For information about configuring federated access to the Athena API, see the section called "Enable federated access to the Athena API".

To download the JDBC driver version 2.0.6 and its documentation, see Connect to Amazon Athena with JDBC. For information about this version, see JDBC Driver Release Notes.

To download the ODBC driver version 1.0.4 and its documentation, see Connect to Amazon Athena with ODBC. For information about this version, ODBC Driver Release Notes.

For more information about SAML 2.0 support in AWS, see About SAML 2.0 Federation in the *IAM User Guide*.

October 15, 2018

Published on *2018-10-15*

If you have upgraded to the AWS Glue Data Catalog, there are two new features that provide support for:

- Encryption of the Data Catalog metadata. If you choose to encrypt metadata in the Data Catalog, you must add specific policies to Athena. For more information, see Access to Encrypted Metadata in the AWS Glue Data Catalog.
- Fine-grained permissions to access resources in the AWS Glue Data Catalog. You can now define identity-based (IAM) policies that restrict or allow access to specific databases and tables from the Data Catalog used in Athena. For more information, see Configure access to databases and tables in the AWS Glue Data Catalog.

> **Note**
>
> Data resides in the Amazon S3 buckets, and access to it is controlled by Control access to Amazon S3 from Athena. To access data in databases and tables, continue to use access control policies to Amazon S3 buckets that store the data.

October 10, 2018

Published on *2018-10-10*

Athena supports `CREATE TABLE AS SELECT`, which creates a table from the result of a `SELECT` query statement. For details, see Creating a Table from Query Results (CTAS).

Before you create CTAS queries, it is important to learn about their behavior in the Athena documentation. It contains information about the location for saving query results in Amazon S3, the list of supported formats for storing CTAS query results, the number of partitions you can create, and supported compression formats. For more information, see Considerations and limitations for CTAS queries.

Use CTAS queries to:

- Create a table from query results in one step.
- Create CTAS queries in the Athena console, using Examples. For information about syntax, see CREATE TABLE AS.
- Transform query results into other storage formats, such as PARQUET, ORC, AVRO, JSON, and TEXTFILE. For more information, see Considerations and limitations for CTAS queries and Use columnar storage formats.

September 6, 2018

Published on *2018-09-06*

Released the new version of the ODBC driver (version 1.0.3). The new version of the ODBC driver streams results by default, instead of paging through them, allowing business intelligence tools to retrieve large data sets faster. This version also includes improvements, bug fixes, and an updated documentation for *"Using SSL with a Proxy Server"*. For details, see the Release Notes for the driver.

For downloading the ODBC driver version 1.0.3 and its documentation, see Connect to Amazon Athena with ODBC.

The streaming results feature is available with this new version of the ODBC driver. It is also available with the JDBC driver. For information about streaming results, see the ODBC Driver Installation and Configuration Guide, and search for **UseResultsetStreaming**.

The ODBC driver version 1.0.3 is a drop-in replacement for the previous version of the driver. We recommend that you migrate to the current driver.

Important

To use the ODBC driver version 1.0.3, follow these requirements:

- Keep the port 444 open to outbound traffic.
- Add the `athena:GetQueryResultsStream` policy action to the list of policies for Athena. This policy action is not exposed directly with the API and is only used with the ODBC and JDBC drivers, as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess.

August 23, 2018

Published on *2018-08-23*

Added support for these DDL-related features and fixed several bugs, as follows:

- Added support for `BINARY` and `DATE` data types for data in Parquet, and for `DATE` and `TIMESTAMP` data types for data in Avro.
- Added support for `INT` and `DOUBLE` in DDL queries. `INTEGER` is an alias to `INT`, and `DOUBLE PRECISION` is an alias to `DOUBLE`.
- Improved performance of `DROP TABLE` and `DROP DATABASE` queries.
- Removed the creation of `_$folder$` object in Amazon S3 when a data bucket is empty.
- Fixed an issue where `ALTER TABLE ADD PARTITION` threw an error when no partition value was provided.
- Fixed an issue where `DROP TABLE` ignored the database name when checking partitions after the qualified name had been specified in the statement.

For more about the data types supported in Athena, see Data types in Amazon Athena.

For information about supported data type mappings between types in Athena, the JDBC driver, and Java data types, see the *"Data Types"* section in the JDBC Driver Installation and Configuration Guide.

August 16, 2018

Published on *2018-08-16*

Released the JDBC driver version 2.0.5. The new version of the JDBC driver streams results by default, instead of paging through them, allowing business intelligence tools to retrieve large data sets faster. Compared to the previous version of the JDBC driver, there are the following performance improvements:

- Approximately 2x performance increase when fetching less than 10K rows.
- Approximately 5-6x performance increase when fetching more than 10K rows.

The streaming results feature is available only with the JDBC driver. It is not available with the ODBC driver. You cannot use it with the Athena API. For information about streaming results, see the JDBC Driver Installation and Configuration Guide, and search for **UseResultsetStreaming**.

For downloading the JDBC driver version 2.0.5 and its documentation, see Connect to Amazon Athena with JDBC.

The JDBC driver version 2.0.5 is a drop-in replacement for the previous version of the driver (2.0.2). To ensure that you can use the JDBC driver version 2.0.5, add the `athena:GetQueryResultsStream` policy action to the list of policies for Athena. This policy action is not exposed directly with the API and is only used with the JDBC driver, as part of streaming results support. For an example policy, see AWS managed policy: AWSQuicksightAthenaAccess. For more information about migrating from version 2.0.2 to version 2.0.5 of the driver, see the JDBC Driver Migration Guide.

If you are migrating from a 1.x driver to a 2.x driver, you will need to migrate your existing configurations to the new configuration. We highly recommend that you migrate to the current version of the driver. For more information, see the JDBC Driver Migration Guide.

August 7, 2018

Published on *2018-08-07*

You can now store Amazon Virtual Private Cloud flow logs directly in Amazon S3 in a GZIP format, where you can query them in Athena. For information, see Query Amazon VPC flow logs and Amazon VPC Flow Logs can now be delivered to S3.

June 5, 2018

Published on *2018-06-05*

Topics

- Support for Views
- Improvements and Updates to Error Messages
- Bug Fixes

Support for Views

Added support for views. You can now use CREATE VIEW and CREATE DIALECT VIEW, DESCRIBE VIEW, DROP VIEW, SHOW CREATE VIEW, and SHOW VIEWS in Athena. The query that defines the view runs each time you reference the view in your query. For more information, see Work with views.

Improvements and Updates to Error Messages

- Included a GSON 2.8.0 library into the CloudTrail SerDe, to solve an issue with the CloudTrail SerDe and enable parsing of JSON strings.
- Enhanced partition schema validation in Athena for Parquet, and, in some cases, for ORC, by allowing reordering of columns. This enables Athena to better deal with changes in schema evolution over time, and with tables added by the AWS Glue Crawler. For more information, see Handle schema updates.
- Added parsing support for `SHOW VIEWS`.
- Made the following improvements to most common error messages:
  - Replaced an Internal Error message with a descriptive error message when a SerDe fails to parse the column in an Athena query. Previously, Athena issued an internal error in cases of parsing errors. The new error message reads: "HIVE_BAD_DATA: Error parsing field value for field 0: java.lang.String cannot be cast to org.openx.data.jsonserde.json.JSONObject".
  - Improved error messages about insufficient permissions by adding more detail.

Bug Fixes

Fixed the following bugs:

- Fixed an issue that enables the internal translation of `REAL` to `FLOAT` data types. This improves integration with the AWS Glue crawler that returns `FLOAT` data types.
- Fixed an issue where Athena was not converting AVRO `DECIMAL` (a logical type) to a `DECIMAL` type.

- Fixed an issue where Athena did not return results for queries on Parquet data with WHERE clauses that referenced values in the TIMESTAMP data type.

May 17, 2018

Published on *2018-05-17*

Increased query concurrency quota in Athena from five to twenty. This means that you can submit and run up to twenty DDL queries and twenty SELECT queries at a time. Note that the concurrency quotas are separate for DDL and SELECT queries.

Concurrency quotas in Athena are defined as the number of queries that can be submitted to the service concurrently. You can submit up to twenty queries of the same type (DDL or SELECT) at a time. If you submit a query that exceeds the concurrent query quota, the Athena API displays an error message.

After you submit your queries to Athena, it processes the queries by assigning resources based on the overall service load and the amount of incoming requests. We continuously monitor and make adjustments to the service so that your queries process as fast as possible.

For information, see Service Quotas. This is an adjustable quota. You can use the Service Quotas console to request a quota increase for concurrent queries.

April 19, 2018

Published on *2018-04-19*

Released the new version of the JDBC driver (version 2.0.2) with support for returning the ResultSet data as an Array data type, improvements, and bug fixes. For details, see the Release Notes for the driver.

For information about downloading the new JDBC driver version 2.0.2 and its documentation, see Connect to Amazon Athena with JDBC.

The latest version of the JDBC driver is 2.0.2. If you are migrating from a 1.x driver to a 2.x driver, you will need to migrate your existing configurations to the new configuration. We highly recommend that you migrate to the current driver.

For information about the changes introduced in the new version of the driver, the version differences, and examples, see the JDBC Driver Migration Guide.

April 6, 2018

Published on *2018-04-06*

Use auto-complete to type queries in the Athena console.

March 15, 2018

Published on *2018-03-15*

Added an ability to automatically create Athena tables for CloudTrail log files directly from the CloudTrail console. For information, see Use the CloudTrail console to create an Athena table for CloudTrail logs .

February 2, 2018

Published on *2018-02-12*

Added an ability to securely offload intermediate data to disk for memory-intensive queries that use the `GROUP BY` clause. This improves the reliability of such queries, preventing "Query resource exhausted" errors.

January 19, 2018

Published on *2018-01-19*

Athena uses Presto, an open-source distributed query engine, to run queries.

With Athena, there are no versions to manage. We have transparently upgraded the underlying engine in Athena to a version based on Presto version 0.172. No action is required on your end.

With the upgrade, you can now use Presto 0.172 Functions and Operators, including Presto 0.172 Lambda Expressions in Athena.

Major updates for this release, including the community-contributed fixes, include:

- Support for ignoring headers. You can use the `skip.header.line.count` property when defining tables, to allow Athena to ignore headers. This is supported for queries that use the LazySimpleSerDe and OpenCSV SerDe, and not for Grok or Regex SerDes.
- Support for the `CHAR(n)` data type in `STRING` functions. The range for `CHAR(n)` is `[1.255]`, while the range for `VARCHAR(n)` is `[1,65535]`.

- Support for correlated subqueries.
- Support for Presto Lambda expressions and functions.
- Improved performance of the `DECIMAL` type and operators.
- Support for filtered aggregations, such as `SELECT sum(col_name) FILTER`, where `id > 0`.
- Push-down predicates for the `DECIMAL`, `TINYINT`, `SMALLINT`, and `REAL` data types.
- Support for quantified comparison predicates: `ALL`, `ANY`, and `SOME`.
- Added functions: `arrays_overlap()`, `array_except()`, `levenshtein_distance()`, `codepoint()`, `skewness()`, `kurtosis()`, and `typeof()`.
- Added a variant of the `from_unixtime()` function that takes a timezone argument.
- Added the `bitwise_and_agg()` and `bitwise_or_agg()` aggregation functions.
- Added the `xxhash64()` and `to_big_endian_64()` functions.
- Added support for escaping double quotes or backslashes using a backslash with a JSON path subscript to the `json_extract()` and `json_extract_scalar()` functions. This changes the semantics of any invocation using a backslash, as backslashes were previously treated as normal characters.

For more information about functions and operators, see DML queries, functions, and operators in this guide, and Functions and operators in the Presto documentation.

Athena does not support all of Presto's features. For more information, see Limitations.

Athena release notes for 2017

November 13, 2017

Published on *2017-11-13*

Added support for connecting Athena to the ODBC Driver. For information, see Connect to Amazon Athena with ODBC.

November 1, 2017

Published on *2017-11-01*

Added support for querying geospatial data, and for Asia Pacific (Seoul), Asia Pacific (Mumbai), and EU (London) regions. For information, see Query geospatial data and AWS Regions and Endpoints.

October 19, 2017

Published on *2017-10-19*

Added support for EU (Frankfurt). For a list of supported regions, see AWS Regions and Endpoints.

October 3, 2017

Published on *2017-10-03*

Create named Athena queries with AWS CloudFormation. For more information, see AWS::Athena::NamedQuery in the *AWS CloudFormation User Guide*.

September 25, 2017

Published on *2017-09-25*

Added support for Asia Pacific (Sydney). For a list of supported regions, see AWS Regions and Endpoints.

August 14, 2017

Published on *2017-08-14*

Added integration with the AWS Glue Data Catalog and a migration wizard for updating from the Athena managed data catalog to the AWS Glue Data Catalog. For more information, see Use AWS Glue Data Catalog to connect to your data.

August 4, 2017

Published on *2017-08-04*

Added support for Grok SerDe, which provides easier pattern matching for records in unstructured text files such as logs. For more information, see Grok SerDe. Added keyboard shortcuts to scroll through query history using the console (CTRL + ⇧/⇩ using Windows, CMD + ⇧/⇩ using Mac).

June 22, 2017

Published on *2017-06-22*

Added support for Asia Pacific (Tokyo) and Asia Pacific (Singapore). For a list of supported regions, see AWS Regions and Endpoints.

June 8, 2017

Published on *2017-06-08*

Added support for Europe (Ireland). For more information, see AWS Regions and Endpoints.

May 19, 2017

Published on *2017-05-19*

Added an Amazon Athena API and AWS CLI support for Athena; updated JDBC driver to version 1.1.0; fixed various issues.

- Amazon Athena enables application programming for Athena. For more information, see Amazon Athena API Reference. The latest AWS SDKs include support for the Athena API. For links to documentation and downloads, see the *SDKs* section in Tools for Amazon Web Services.
- The AWS CLI includes new commands for Athena. For more information, see the Amazon Athena API Reference.
- A new JDBC driver 1.1.0 is available, which supports the new Athena API as well as the latest features and bug fixes. Download the driver at https://downloads.athena.us-east-1.amazonaws.com/drivers/AthenaJDBC41-1.1.0.jar. We recommend upgrading to the latest Athena JDBC driver; however, you may still use the earlier driver version. Earlier driver versions do not support the Athena API. For more information, see Connect to Amazon Athena with JDBC.
- Actions specific to policy statements in earlier versions of Athena have been deprecated. If you upgrade to JDBC driver version 1.1.0 and have customer-managed or inline IAM policies attached to JDBC users, you must update the IAM policies. In contrast, earlier versions of the JDBC driver do not support the Athena API, so you can specify only deprecated actions in policies attached to earlier version JDBC users. For this reason, you shouldn't need to update customer-managed or inline IAM policies.
- These policy-specific actions were used in Athena before the release of the Athena API. Use these deprecated actions in policies **only** with JDBC drivers earlier than version 1.1.0. If you are upgrading the JDBC driver, replace policy statements that allow or deny deprecated actions with the appropriate API actions as listed or errors will occur:

Deprecated Policy-Specific Action	Corresponding Athena API Action
`athena:RunQuery`	`athena:StartQueryExecution`
`athena:CancelQueryExecution`	`athena:StopQueryExecution`
`athena:GetQueryExecutions`	`athena:ListQueryExecutions`

Improvements

- Increased the query string length limit to 256 KB.

Bug Fixes

- Fixed an issue that caused query results to look malformed when scrolling through results in the console.
- Fixed an issue where a `\u0000` character string in Amazon S3 data files would cause errors.
- Fixed an issue that caused requests to cancel a query made through the JDBC driver to fail.
- Fixed an issue that caused the AWS CloudTrail SerDe to fail with Amazon S3 data in US East (Ohio).
- Fixed an issue that caused `DROP TABLE` to fail on a partitioned table.

April 4, 2017

Published on *2017-04-04*

Added support for Amazon S3 data encryption and released JDBC driver update (version 1.0.1) with encryption support, improvements, and bug fixes.

Features

- Added the following encryption features:
  - Support for querying encrypted data in Amazon S3.
  - Support for encrypting Athena query results.

- A new version of the driver supports new encryption features, adds improvements, and fixes issues.
- Added the ability to add, replace, and change columns using `ALTER TABLE`. For more information, see Alter Column in the Hive documentation.
- Added support for querying LZO-compressed data.

For more information, see Encryption at rest.

Improvements

- Better JDBC query performance with page-size improvements, returning 1,000 rows instead of 100.
- Added ability to cancel a query using the JDBC driver interface.
- Added ability to specify JDBC options in the JDBC connection URL. See Connect to Amazon Athena with JDBC for the most current JDBC driver.
- Added PROXY setting in the driver, which can now be set using ClientConfiguration in the AWS SDK for Java.

Bug Fixes

Fixed the following bugs:

- Throttling errors would occur when multiple queries were issued using the JDBC driver interface.
- The JDBC driver would stop when projecting a decimal data type.
- The JDBC driver would return every data type as a string, regardless of how the data type was defined in the table. For example, selecting a column defined as an `INT` data type using `resultSet.GetObject()` would return a `STRING` data type instead of `INT`.
- The JDBC driver would verify credentials at the time a connection was made, rather than at the time a query would run.
- Queries made through the JDBC driver would fail when a schema was specified along with the URL.

March 24, 2017

Published on *2017-03-24*

Added the AWS CloudTrail SerDe, improved performance, fixed partition issues.

Features

- Added the AWS CloudTrail SerDe, which has since been superseded by the Hive JSON SerDe for reading CloudTrail logs. For information about querying CloudTrail logs, see Query AWS CloudTrail logs.

Improvements

- Improved performance when scanning a large number of partitions.
- Improved performance on `MSCK Repair Table` operation.
- Added ability to query Amazon S3 data stored in regions other than your primary Region. Standard inter-region data transfer rates for Amazon S3 apply in addition to standard Athena charges.

Bug Fixes

- Fixed a bug where a "table not found error" might occur if no partitions are loaded.
- Fixed a bug to avoid throwing an exception with `ALTER TABLE ADD PARTITION IF NOT EXISTS` queries.
- Fixed a bug in `DROP PARTITIONS`.

February 20, 2017

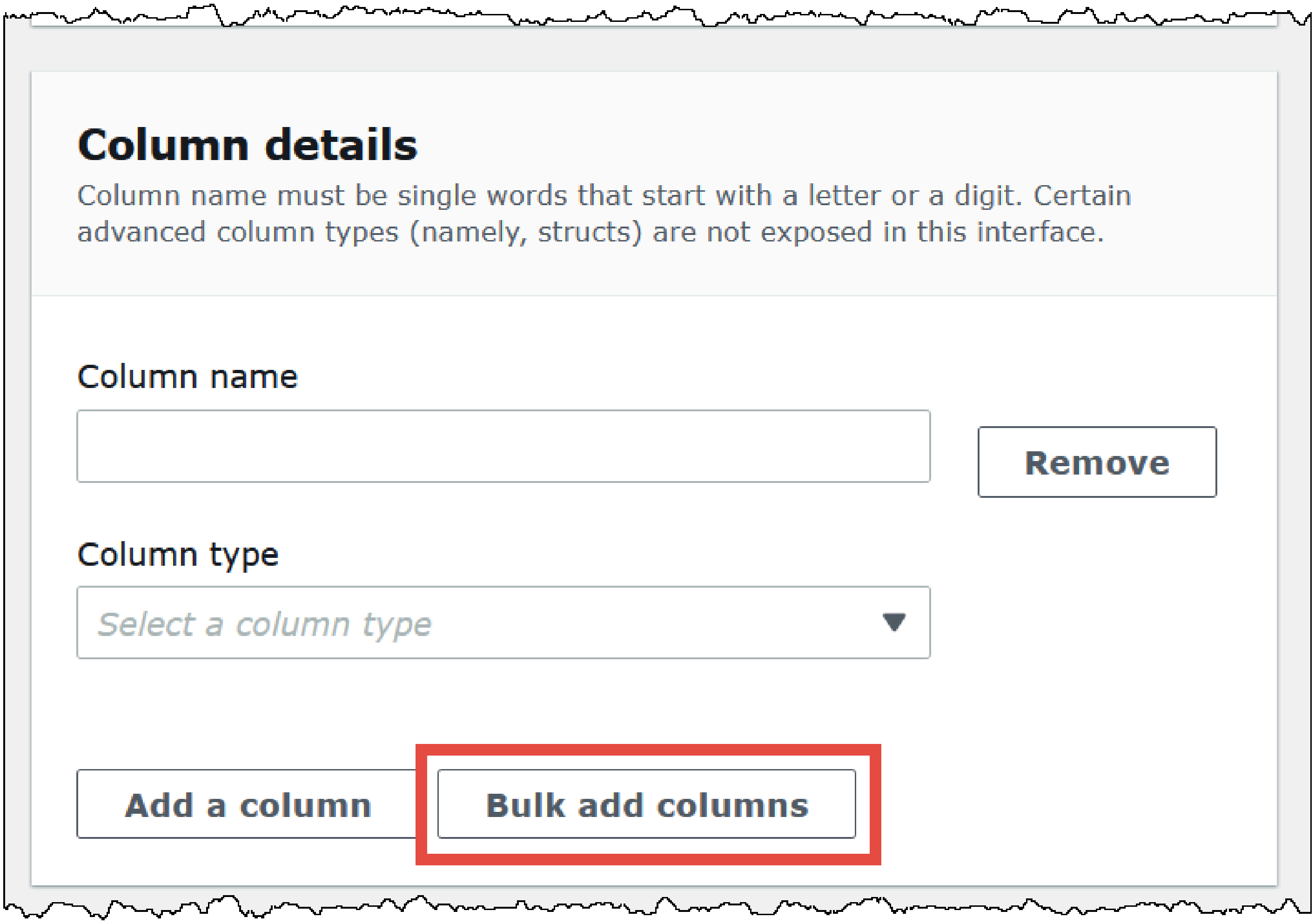
Published on *2017-02-20*

Added support for AvroSerDe and OpenCSVSerDe, US East (Ohio) Region, and bulk editing columns in the console wizard. Improved performance on large Parquet tables.

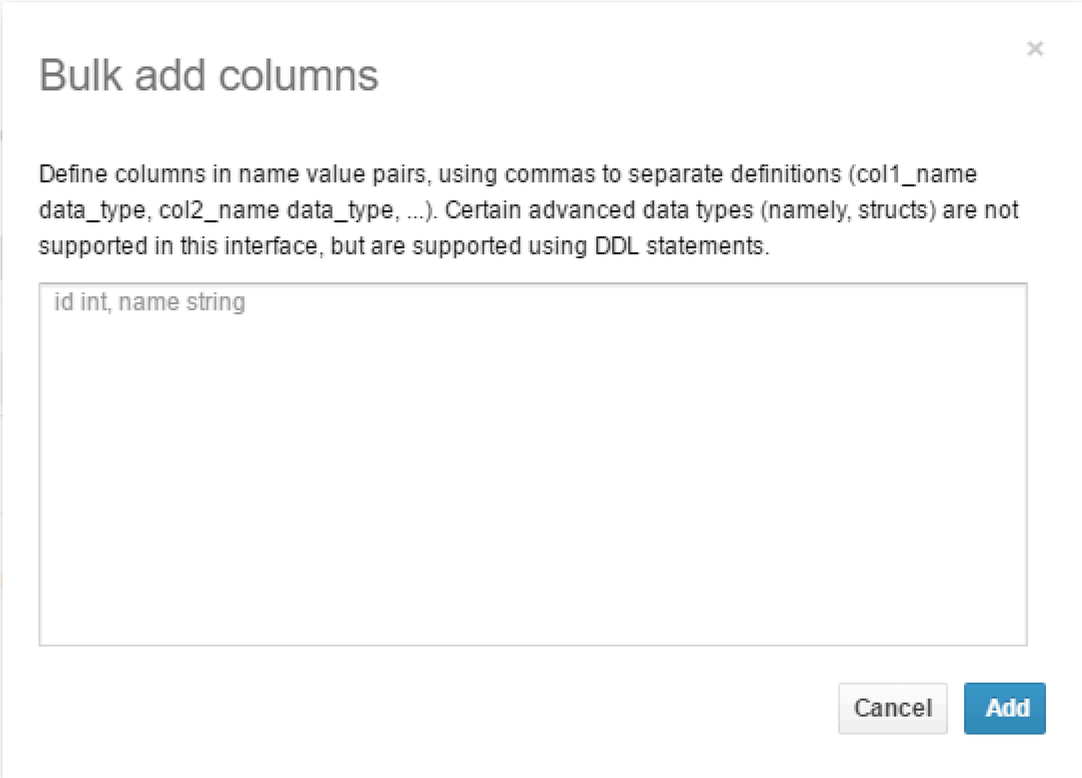
Features

- **Introduced support for new SerDes:**
  - Avro SerDe
  - Open CSV SerDe for processing CSV
- **US East (Ohio)** Region (**us-east-2**) launch. You can now run queries in this region.

- You can now use the **Create Table From S3 bucket data** form to define table schema in bulk. In the query editor, choose **Create**, **S3 bucket data**, and then choose **Bulk add columns** in the **Column details** section.



Type name value pairs in the text box and choose **Add**.



Improvements

- Improved performance on large Parquet tables.

Document history

Latest documentation update: November 20, 2024.

We update the documentation frequently to address your feedback. The following table describes important additions to the Amazon Athena documentation. Not all updates are represented.

Change	Description	Release date
Added documenta tion for Managed query results.	Athena releases managed query results feature that automatically stores, secures, and manages query result data for you at no cost. For more information, see Managed query results.	June 02, 2025
Released the new version of the JDBC driver.	Athena releases JDBC driver version 3.5.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes.	March 18, 2025
Added documenta tion for S3 Tables integration.	Amazon Athena releases capabilities to create and query table operations directly from the S3 console. For more information, see Register S3 table bucket catalogs and query Tables from Athena.	March 14, 2025
Added documenta tion for migrating to SSE-KMS.	Athena recommends that you use SSE-KMS encryption over SSE-S3 or CSE-KMS encryption methods. For more information, see Encryption recommendations.	February 25, 2025
Released the new version of the JDBC driver.	Athena releases JDBC driver version 3.4.0. For more information about this version of the driver, see Amazon Athena JDBC 3.x release notes.	February 18, 2025
Added DDL for CloudFront logs using manual partitioning and partition projectio n.	• Updated documentation to add DDL for creating a table for CloudFront logs in Athena using manual partitioning with JSON and Parquet. • Updated documentation to add DDL for creating a table for CloudFront logs in Athena using partition projection with JSON and Parquet.	February 18, 2025

Change	Description	Release date
Updated documentation for IAM Identity Center enabled Athena workgroups.	Athena now supports federated queries through Lambda and encryption of query results using KMS on TIP enabled workgroups. For more information, see Use IAM Identity Center enabled Athena workgroups.	January 22, 2025
Updated `AmazonAthenaFullAccess` and `AWSQuicksightAthenaAccess` managed policy.	The `glue:GetCatalog` , `glue:GetCatalogs` permissions were added to the AmazonAthenaFullAccess and AWSQuicksightAthenaAccess managed policy. These permissions allow users to access to SageMaker AI Lakehouse catalogs.	January 02, 2025
Added support for Asia Pacific (Malaysia) Region.	Added support for Asia Pacific (Malaysia) Region. For a list of supported regions, see AWS Regions and endpoints.	December 17, 2024
Added documentation for creating data source connections.	You can now create and manage Athena data connections entirely within the Athena console, and the properties for your connections are now centrally stored in the AWS Glue Data Catalog. For more information, see Register and use data catalogs in Athena.	December 3, 2024
Added documentation for Data Catalog views.	Use AWS Glue Data Catalog views when you want a single common view across AWS services. For more information, see Work with views.	August 9, 2024
Updated `AmazonAthenaFullAccess` managed policy.	Added `glue:GetCatalogImportStatus` to the `BaseGluePermissions` section of the AmazonAthenaFullAccess managed policy. The added action allows Athena to use the publicly documented AWS Glue API to retrieve catalog import status.	June 18, 2024

Change	Description	Release date
Updated `AmazonAthenaFullAccess` managed policy.	The `datazone:ListDomains` , `datazone:ListProjects` , and `datazone:ListAccountEnvironments` permissions were added to the AmazonAthenaFullAccess managed policy. The added actions allow Athena users to work with Amazon DataZone domains, projects, and environments. For more information, see Use Amazon DataZone in Athena.	January 3, 2024
Updated `AmazonAthenaFullAccess` managed policy.	Added `glue:StartColumnStatisticsTaskRun` , `glue:GetColumnStatisticsTaskRun` , and `glue:GetColumnStatisticsTaskRuns` permissions to the AmazonAthenaFullAccess managed policy. The added actions allow Athena to call AWS Glue to retrieve statistics for the cost-based optimizer feature. For more information, see Use the cost-based optimizer.	January 3, 2024
Added documentation for IAM Identity Center enabled Athena workgroups.	You can create Athena SQL workgroups that use IAM Identity Center authentication mode. These workgroups support using the same identity across AWS services like Amazon Athena and Amazon EMR Studio. For more information, see Use IAM Identity Center enabled Athena workgroups.	December 5, 2023
Added documentation for querying S3 Express One Zone data	You can use Athena to query data in Amazon S3 Express One Zone storage class. For more information, see Query S3 Express One Zone data.	November 28, 2023
Added documentation for Glue Data Catalog views.	You can use Glue Data Catalog views to provide a single common view across AWS services like Amazon Athena and Amazon Redshift. For more information, see Use Data Catalog views in Athena.	November 27, 2023

Change	Description	Release date
Added documentation for the cost-based optimizer feature.	You can use statistics from AWS Glue to optimize your queries in Athena SQL. For more information, see Use the cost-based optimizer.	November 17, 2023
Added documentation for the Athena JDBC 3.x driver	You can use the Athena JDBC 3.x driver to read query results directly from Amazon S3. The JDBC 3.x driver supports almost all authentication methods that the JDBC 2.x driver supports. For more information, see Athena JDBC 3.x driver.	November 16, 2023
Added documentation for using DataZone in Athena.	You can use DataZone to simplify your experience across AWS analytics services like Athena, AWS Glue, and Lake Formation. For more information, see Use Amazon DataZone in Athena.	October 4, 2023
Added documentation for capacity reservations.	You can now use capacity reservations on Amazon Athena to run SQL queries on fully-managed compute capacity. For more information, see Manage query processing capacity.	April 28, 2023
Added documentation for querying federated views.	You can now create and query views on federated data sources in Athena. For more information, see Query federated views.	April 4, 2023
Added documentation on preventing throttling in Amazon S3.	For more information, see Prevent Amazon S3 throttling.	March 24, 2023
Updated `AmazonAthenaFullAccess` managed policy.	Added `pricing:GetProducts` to the AmazonAthenaFullAccess managed policy. The added action provides access to AWS Billing and Cost Management. For more information, see GetProducts in the *AWS Billing and Cost Management API Reference*.	January 25, 2023

Change	Description	Release date
Expanded documentation for Athena compression support.	Individual topics added for Hive table compression, Iceberg table compression, and ZSTD compression levels. For more information, see Use compression in Athena.	January 20, 2023
Added documentation for Amazon Athena for Apache Spark.	You can now interactively create and run Apache Spark applications and Jupyter compatible notebooks on Amazon Athena. For more information, see Use Apache Spark in Amazon Athena.	November 30, 2022
Added documentation for the Athena IBM Db2 connector.	You can use the Amazon Athena connector for IBM Db2 to query Db2 from Athena. For more information, see Amazon Athena IBM Db2 connector	November 18, 2022
Added documentation for query result reuse.	When you re-run a query in Athena, you can now optionally choose to reuse the last stored query result. This can increase performance and reduce costs in terms of the number of bytes scanned. For more information, see Reuse query results in Athena.	November 8, 2022
Updated documentation for CloudTrail logs.	The `CREATE TABLE` DDL for querying CloudTrail logs has been updated to use the JSON SerDe instead of the CloudTrail SerDe. For more information, see Query AWS CloudTrail logs.	November 3, 2022
Added documentation for Athena engine version 3.	For more information about Athena engine version 3, see Athena engine version 3.	October 13, 2022
Added tutorial on configuring SSO for ODBC using the Okta plugin.	Configure the Amazon Athena ODBC driver and the Okta plugin for single sign-on (SSO) capability using the Okta identity provider. For more information, see Configure SSO for ODBC using the Okta plugin and Okta Identity Provider.	August 23, 2022

Change	Description	Release date
Added documentation for viewing query plans and statistics in the Athena console.	You can use the Athena query editor to see graphical representations of how your queries will be run and graphs, details, and statistics of how completed queries ran. For more information, see View execution plans for SQL queries and View statistics and execution details for completed queries.	July 21, 2022
Added documentation for querying Apache Hive views in external Hive metastores.	You can use Athena to query Apache views created in external Hive metastores. Some Hive functions are not supported or require special handling. For more information, see Work with Hive views.	April 22, 2022
Added documentation for saved queries.	You can use the saved queries feature in Athena to save, recall, edit, and rename your queries. For more information, see Use saved queries in this guide and UpdateNamedQuery in the *Amazon Athena API Reference*.	February 28, 2022
Added preview documentation for Apache Iceberg support.	Athena supports read, time travel, and write queries for Apache Iceberg tables that use the Apache Parquet format for data and the AWS Glue catalog for their metastore. For more information, see Query Apache Iceberg tables.	November 26, 2021
Added documentation for cross-account federated queries.	You can use the cross-account federated query feature to query data sources in another account. For information about setting up permissions to enable this feature, see Enable cross-account federated queries.	November 12, 2021
Added documentation for the Athena `UNLOAD` statement.	Use the `UNLOAD` statement to write query the results from a `SELECT` statement to the Apache Parquet, ORC, Apache Avro, and JSON formats. For more information, see UNLOAD.	August 5, 2021

Change	Description	Release date
Added documentation for the Athena `EXPLAIN` statement feature.	For more information, see Using EXPLAIN and EXPLAIN ANALYZE in Athena and Understand Athena EXPLAIN statement results.	April 5, 2021
Added pages on troubleshooting and performance tuning in Athena.	For more information, see Troubleshoot issues in Athena and Optimize Athena performance.	December 30, 2020
Added documentation for Athena engine versioning and Athena engine version 2.	For more information, see Athena engine versioning.	November 11, 2020
Updated federated query documentation for general availability release.	For more information, see Use Amazon Athena Federated Query and Use CalledVia context keys for Athena.	November 11, 2020
Added documentation for using the JDBC driver with Lake Formation for federated access to Athena.	For more information, see Use Lake Formation and JDBC or ODBC drivers for federated access to Athena and Tutorial: Configure federated access for Okta users to Athena using Lake Formation and JDBC.	September 25, 2020
Added documentation for the Amazon Athena OpenSearch data connector.	For more information, see Amazon Athena OpenSearch connector.	July 21, 2020

Change	Description	Release date
Added documenta tion for querying Hudi datasets.	For more information, see Query Apache Hudi datasets.	July 9, 2020
Added documenta tion on querying Apache web server logs and IIS web server logs stored in Amazon S3.	For more information, see Query Apache logs stored in Amazon S3 and Query internet information server (IIS) logs stored in Amazon S3.	July 8, 2020
Added documenta tion for the general release of the Athena Data Connector for External Hive Metastore.	For more information, see Use an external Hive metastore.	June 1, 2020
Added documenta tion for tagging data catalog resources.	For more information, see Tag Athena resources.	June 1, 2020
Added documenta tion on partition projection.	For more information, see Use partition projection with Amazon Athena.	May 21, 2020
Updated the Java code examples for Athena.	For more information, see Code samples.	May 11, 2020
Added a topic on querying Amazon GuardDuty findings.	For more information, see Query Amazon GuardDuty findings.	March 19, 2020

Change	Description	Release date
Added a topic on using CloudWatch Events to monitor Athena query state transitions.	For more information, see Monitor Athena query events with EventBridge.	March 11, 2020
Added a topic on querying AWS Global Accelerator flow logs with Athena.	For more information, see Query AWS Global Accelerator flow logs.	February 6, 2020

Change	Description	Release date
• Added documentation on using CTAS with INSERT INTO to add data from an unpartitioned source to a partitioned destination. • Added download links for the 1.1.0 preview version of the ODBC driver for Athena. • Description for `SHOW DATABASES LIKE` regex corrected. • Corrected `partitioned_by` syntax in CTA topic. • Other minor fixes.	Documentation updates include, but are not limited to, the following topics: • Use CTAS and INSERT INTO for ETL and data analysis • Connect to Amazon Athena with ODBC (The 1.1.0 preview features are now included in the 1.1.2 ODBC driver.) • SHOW DATABASES • CREATE TABLE AS	February 4, 2020

Change	Description	Release date
Added documenta tion on using CTAS with INSERT INTO to add data from a partitioned source to a partitioned destination.	For more information, see Use CTAS and INSERT INTO to work around the 100 partition limit.	January 22, 2020
Query results location informati on updated.	Athena no longer creates a 'default' query results location. For more information, see Specify a query result location.	January 20, 2020
Added topic on querying the AWS Glue Data Catalog. Updated informati on about service quotas (formerly "service limits") in Athena.	For more information, see the following topics: • Query the AWS Glue Data Catalog • Service Quotas	January 17, 2020
Corrected topic on OpenCSVSe rDe to note that the `TIMESTAMP` type should be specified in the UNIX numeric format.	For more information, see Open CSV SerDe for processing CSV.	January 15, 2020

Change	Description	Release date
Updated security topic on encryption to note that Athena does not support asymmetric keys.	Athena supports only symmetric keys for reading and writing data. For more information, see Supported Amazon S3 encryption options.	January 8, 2020
Added information about cross-account access to Amazon S3 buckets that are encrypted with a custom AWS KMS key.	For more information, see Configure cross-account access to a bucket encrypted with a custom AWS KMS key.	December 13, 2019
Added documentation for federated queries, external Hive metastores, machine learning, and user defined functions. Added new CloudWatch metrics.	For more information, see the following topics: • Use Amazon Athena Federated Query • Available data source connectors • Use an external Hive metastore • Use Machine Learning (ML) with Amazon Athena • Query with user defined functions • List of CloudWatch metrics and dimensions for Athena	November 26, 2019
Added section for new INSERT INTO command and updated query result location information for supporting data manifest files.	For more information, see INSERT INTO and Work with query results and recent queries.	September 18, 2019

Change	Description	Release date
Added section for interface VPC endpoints (PrivateLink) support. Updated JDBC drivers. Updated informati on about enriched VPC flow logs.	For more information, see Connect to Amazon Athena using an interface VPC endpoint, Query Amazon VPC flow logs, and Connect to Amazon Athena with JDBC.	September 11, 2019
Added section on integrating with AWS Lake Formation.	For more information, see Use Athena to query data registered with AWS Lake Formation.	June 26, 2019
Updated Security section for consistency with other AWS services.	For more information, see Amazon Athena security.	June 26, 2019
Added section on querying AWS WAF logs.	For more information, see Query AWS WAF logs.	May 31, 2019

Change	Description	Release date
Released the new version of the ODBC driver with support for Athena workgroup s.	To download the ODBC driver version 1.0.5 and its documentation, see Connect to Amazon Athena with ODBC. There are no changes to the ODBC driver connection string when you use tags on workgroups. To use tags, upgrade to the latest version of the ODBC driver, which is this current version. This driver version lets you use Athena API workgroup actions to create and manage workgroups, and Athena API tag actions to add, list, or remove tags on workgroups. Before you begin, make sure that you have resource-level permissions in IAM for actions on workgroups and tags.	March 5, 2019
Added tag support for workgroups in Amazon Athena.	A tag consists of a key and a value, both of which you define. When you tag a workgroup, you assign custom metadata to it. For example, create a workgroup for each cost center. Then, by adding tags to these workgroups, you can track your Athena spending for each cost center. For more information, see Using tags for billing in the *AWS Billing and Cost Management User Guide*.	February 22, 2019

Change	Description	Release date
Improved the JSON OpenX SerDe used in Athena.	The improvements include, but are not limited to, the following: • Support for the `ConvertDotsInJsonKeysToUnderscores` property. When set to `TRUE`, it allows the SerDe to replace the dots in key names with underscores. For example, if the JSON dataset contains a key with the name `"a.b"`, you can use this property to define the column name to be `"a_b"` in Athena. The default is `FALSE`. By default, Athena does not allow dots in column names. • Support for the `case.insensitive` property. By default, Athena requires that all keys in your JSON dataset use lowercase. Using `WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)` allows you to use case-sensitive key names in your data. The default is `TRUE`. When set to `TRUE`, the SerDe converts all uppercase columns to lowercase. For more information, see OpenX JSON SerDe.	February 18, 2019

Change	Description	Release date
Added support for workgroups.	Use workgroups to separate users, teams, applications, or workloads, and to set limits on amount of data each query or the entire workgroup can process. Because workgroups act as IAM resources, you can use resource-level permissions to control access to a specific workgroup. You can also view query-related metrics in Amazon CloudWatch, control query costs by configuring limits on the amount of data scanned, create thresholds, and trigger actions, such as Amazon SNS alarms, when these thresholds are breached. For more information, see Use workgroups to control query access and costs and Use CloudWatch and EventBridge to monitor queries and control costs.	February 18, 2019
Added support for analyzing logs from Network Load Balancer.	Added example Athena queries for analyzing logs from Network Load Balancer. These logs receive detailed information about the Transport Layer Security (TLS) requests sent to the Network Load Balancer. You can use these access logs to analyze traffic patterns and troubleshoot issues. For information, see the section called "Network Load Balancer".	January 24, 2019
Released the new versions of the JDBC and ODBC driver with support for federated access to Athena API with the AD FS and SAML 2.0 (Security Assertion Markup Language 2.0).	With this release of the drivers, federated access to Athena is supported for the Active Directory Federation Service (AD FS 3.0). Access is established through the versions of JDBC or ODBC drivers that support SAML 2.0. For information about configuring federated access to the Athena API, see the section called "Enable federated access to the Athena API".	November 10, 2018

Change	Description	Release date
Added support for fine-grained access control to databases and tables in Athena. Additionally, added policies in Athena that allow you to encrypt database and table metadata in the Data Catalog.	Added support for creating identity-based (IAM) policies that provide fine-grained access control to resources in the AWS Glue Data Catalog, such as databases and tables used in Athena. Additionally, you can encrypt database and table metadata in the Data Catalog, by adding specific policies to Athena. For details, see Configure access to databases and tables in the AWS Glue Data Catalog.	October 15, 2018
Added support for `CREATE TABLE AS SELECT` statements. Made other improvements in the documentation.	Added support for `CREATE TABLE AS SELECT` statements. See Create a table from query results (CTAS), Considerations and limitations for CTAS queries, and Examples of CTAS queries.	October 10, 2018
Released the ODBC driver version 1.0.3 with support for streaming results instead of fetching them in pages. Made other improvements in the documentation.	The ODBC driver version 1.0.3 supports streaming results and also includes improvements, bug fixes, and an updated documentation for *"Using SSL with a Proxy Server"*. For downloading the ODBC driver version 1.0.3 and its documentation, see Connect to Amazon Athena with ODBC.	September 6, 2018

Change	Description	Release date
Released the JDBC driver version 2.0.5 with default support for streaming results instead of fetching them in pages. Made other improvements in the documenta tion.	Released the JDBC driver 2.0.5 with default support for streaming results instead of fetching them in pages. For information, see Connect to Amazon Athena with JDBC.	August 16, 2018
Updated the documentation for querying Amazon Virtual Private Cloud flow logs, which can be stored directly in Amazon S3 in a GZIP format. Updated examples for querying ALB logs.	Updated the documentation for querying Amazon Virtual Private Cloud flow logs, which can be stored directly in Amazon S3 in a GZIP format. For informati on, see Query Amazon VPC flow logs. Updated examples for querying ALB logs. For informati on, see Query Application Load Balancer logs.	August 7, 2018
Added support for views. Added guidelines for schema manipulat ions for various data storage formats.	Added support for views. For information, see Work with views. Updated this guide with guidance on handling schema updates for various data storage formats. For informati on, see Handle schema updates.	June 5, 2018

Change	Description	Release date
Increased default query concurrency limits from five to twenty.	You can submit and run up to twenty DDL queries and twenty SELECT queries at a time. For information, see Service Quotas.	May 17, 2018
Added query tabs, and an ability to configure auto-complete in the Query Editor.	Added query tabs, and an ability to configure auto-complete in the Query Editor. For information, see Get started.	May 8, 2018
Released the JDBC driver version 2.0.2.	Released the new version of the JDBC driver (version 2.0.2). For information, see Connect to Amazon Athena with JDBC.	April 19, 2018
Added auto-complete for typing queries in the Athena console.	Added auto-complete for typing queries in the Athena console.	April 6, 2018
Added an ability to create Athena tables for CloudTrail log files directly from the CloudTrail console.	Added an ability to automatically create Athena tables for CloudTrail log files directly from the CloudTrail console. For information, see Use the CloudTrail console to create an Athena table for CloudTrail logs .	March 15, 2018
Added support for securely offloading intermediate data to disk for queries with GROUP BY.	Added an ability to securely offload intermediate data to disk for memory-intensive queries that use the GROUP BY clause. This improves the reliability of such queries, preventing "Query resource exhausted" errors. For more information, see the release note for February 2, 2018.	February 2, 2018

Change	Description	Release date
Added support for Presto version 0.172.	Upgraded the underlying engine in Amazon Athena to a version based on Presto version 0.172. For more information, see the release note for January 19, 2018.	January 19, 2018
Added support for the ODBC Driver.	Added support for connecting Athena to the ODBC Driver. For information, see Connecting to Amazon Athena with ODBC.	November 13, 2017
Added support for Asia Pacific (Seoul), Asia Pacific (Mumbai), and Europe (London) regions. Added support for querying geospatial data.	Added support for querying geospatial data, and for Asia Pacific (Seoul), Asia Pacific (Mumbai), Europe (London) regions. For information, see Querying geospatial data and AWS Regions and endpoints.	November 1, 2017
Added support for Europe (Frankfurt).	Added support for Europe (Frankfurt). For a list of supported regions, see AWS Regions and endpoints.	October 19, 2017
Added support for named Athena queries with AWS CloudFormation.	Added support for creating named Athena queries with AWS CloudFormation. For more information, see AWS::Athena::NamedQuery in the *AWS CloudFormation User Guide*.	October 3, 2017
Added support for Asia Pacific (Sydney).	Added support for Asia Pacific (Sydney). For a list of supported regions, see AWS Regions and endpoints.	September 25, 2017

Change	Description	Release date
Added a section to this guide for querying AWS service logs and different types of data, including maps, arrays, nested data, and data containing JSON.	Added examples for Query AWS service logs and for querying different types of data in Athena. For information, see Run SQL queries in Amazon Athena.	September 5, 2017
Added support for AWS Glue Data Catalog.	Added integration with the AWS Glue Data Catalog and a migration wizard for updating from the Athena managed data catalog to the AWS Glue Data Catalog. For more information, see Integration with AWS Glue and AWS Glue.	August 14, 2017
Added support for Grok SerDe.	Added support for Grok SerDe, which provides easier pattern matching for records in unstructured text files such as logs. For more information, see Grok SerDe. Added keyboard shortcuts to scroll through query history using the console.	August 4, 2017
Added support for Asia Pacific (Tokyo).	Added support for Asia Pacific (Tokyo) and Asia Pacific (Singapore). For a list of supported regions, see AWS Regions and endpoints.	June 22, 2017
Added support for Europe (Ireland).	Added support for Europe (Ireland). For more informati on, see AWS Regions and endpoints.	June 8, 2017
Added an Amazon Athena API and AWS CLI support.	Added an Amazon Athena API and AWS CLI support for Athena. Updated JDBC driver to version 1.1.0.	May 19, 2017

Change	Description	Release date
Added support for Amazon S3 data encryption.	Added support for Amazon S3 data encryption and released a JDBC driver update (version 1.0.1) with encryption support, improvements, and bug fixes. For more information, see Encryption at rest.	April 4, 2017
Added the AWS CloudTrail SerDe.	Added the AWS CloudTrail SerDe, improved performance, fixed partition issues. • The AWS CloudTrail SerDe has been superseded by the Hive JSON SerDe for reading CloudTrail logs. For information about querying CloudTrail logs, see Query AWS CloudTrail logs. • Improved performance when scanning a large number of partitions. • Improved performance on `MSCK Repair Table` operation. • Added ability to query Amazon S3 data stored in regions other than your primary region. Standard inter-region data transfer rates for Amazon S3 apply in addition to standard Athena charges.	March 24, 2017
Added support for US East (Ohio).	Added support for Avro SerDe and Open CSV SerDe for processing CSV, US East (Ohio), and bulk editing columns in the console wizard. Improved performance on large Parquet tables.	February 20, 2017
	The initial release of the *Amazon Athena User Guide*.	November, 2016