

---

# Amazon Athena

## User Guide

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

# Table of Contents

What is Amazon Athena?	1
When should I use Athena?	1
Accessing Athena	1
Understanding Tables, Databases, and the Data Catalog	2
AWS Service Integrations with Athena	3
Setting Up	6
Sign Up for AWS	6
To create an AWS account	6
Create an IAM User	6
To create a group for administrators	6
To create an IAM user for yourself, add the user to the administrators group, and create a password for the user	7
Attach Managed Policies for Using Athena	7
Getting Started	8
Prerequisites	8
Step 1: Create a Database	8
Step 2: Create a Table	9
Step 3: Query Data	11
Connecting to Other Data Sources	14
Accessing Amazon Athena	15
Using the Console	15
Using the API	15
Using the CLI	15
Connecting to Data Sources	16
Integration with AWS Glue	16
Using AWS Glue to Connect to Data Sources in Amazon S3	17
Best Practices When Using Athena with AWS Glue	20
Upgrading to the AWS Glue Data Catalog Step-by-Step	29
FAQ: Upgrading to the AWS Glue Data Catalog	32
Using a Hive Metastore	34
Overview of Features	35
Workflow	35
Considerations and Limitations	36
Connecting Athena to an Apache Hive Metastore	37
Using the AWS Serverless Application Repository	44
Using a Default Catalog	46
Using the AWS CLI with Hive Metastores	49
Reference Implementation	55
Using Amazon Athena Federated Query (Preview)	56
Considerations and Limitations	56
Deploying a Connector and Connecting to a Data Source	57
Using the AWS Serverless Application Repository	58
Athena Data Source Connectors	59
Writing Federated Queries	61
Writing a Data Source Connector	65
IAM Policies for Accessing Data Catalogs	65
Data Catalog Example Policies	66
Managing Data Sources	69
Connecting to Amazon Athena with ODBC and JDBC Drivers	72
Using Athena with the JDBC Driver	72
Connecting to Amazon Athena with ODBC	73
Creating Databases and Tables	76
Creating Databases	76
Creating Tables	78

Considerations and Limitations .....	79
Creating Tables Using AWS Glue or the Athena Console .....	80
Names for Tables, Databases, and Columns .....	84
Table names and table column names in Athena must be lowercase .....	84
Special characters .....	84
Reserved Keywords .....	85
List of Reserved Keywords in DDL Statements .....	85
List of Reserved Keywords in SQL SELECT Statements .....	85
Examples of Queries with Reserved Words .....	86
Table Location in Amazon S3 .....	86
Table Location and Partitions .....	87
Columnar Storage Formats .....	88
Converting to Columnar Formats .....	88
Overview .....	89
Before you begin .....	8
Example: Converting data to Parquet using an EMR cluster .....	91
Partitioning Data .....	92
Considerations and Limitations .....	92
Creating and Loading a Table with Partitioned Data .....	93
Preparing Partitioned and Nonpartitioned Data for Querying .....	93
Partition Projection .....	96
Pruning and Projection for Heavily Partitioned Tables .....	97
Using Partition Projection .....	97
Use Cases .....	97
Considerations and Limitations .....	98
Setting up Partition Projection .....	98
Supported Types for Partition Projection .....	103
Dynamic ID Partitioning .....	107
Amazon Kinesis Data Firehose Example .....	109
Running Queries .....	110
Query Results and Query History .....	110
Getting a Query ID .....	111
Identifying Query Output Files .....	112
Downloading Query Results Files Using the Athena Console .....	114
Specifying a Query Result Location .....	115
Viewing Query History .....	117
Working with Views .....	119
When to Use Views? .....	119
Supported Actions for Views in Athena .....	120
Considerations for Views .....	120
Limitations for Views .....	121
Working with Views in the Console .....	121
Creating Views .....	122
Examples of Views .....	123
Updating Views .....	124
Deleting Views .....	124
Creating a Table from Query Results (CTAS) .....	124
Considerations and Limitations for CTAS Queries .....	124
Running CTAS Queries in the Console .....	126
Bucketing vs Partitioning .....	129
Examples of CTAS Queries .....	130
Using CTAS and INSERT INTO for ETL .....	133
Creating a Table with More Than 100 Partitions .....	139
Handling Schema Updates .....	142
Summary: Updates and Data Formats in Athena .....	142
Index Access in ORC and Parquet .....	143
Types of Updates .....	145



Updates in Tables with Partitions .....	149
Querying Arrays .....	150
Creating Arrays .....	151
Concatenating Arrays .....	152
Converting Array Data Types .....	153
Finding Lengths .....	154
Accessing Array Elements .....	154
Flattening Nested Arrays .....	155
Creating Arrays from Subqueries .....	157
Filtering Arrays .....	158
Sorting Arrays .....	159
Using Aggregation Functions with Arrays .....	160
Converting Arrays to Strings .....	161
Using Arrays to Create Maps .....	161
Querying Arrays with Complex Types and Nested Structures .....	162
Querying Geospatial Data .....	167
What is a Geospatial Query? .....	168
Input Data Formats and Geometry Data Types .....	168
List of Supported Geospatial Functions .....	168
Examples: Geospatial Queries .....	177
Querying Hudi Datasets .....	178
Hudi Dataset Storage Types .....	179
Considerations and Limitations .....	179
Creating Hudi Tables .....	180
Querying JSON .....	182
Best Practices for Reading JSON Data .....	182
Extracting Data from JSON .....	184
Searching for Values in JSON Arrays .....	186
Obtaining Length and Size of JSON Arrays .....	187
Troubleshooting JSON Queries .....	188
Using ML with Athena (Preview) .....	189
Considerations and Limitations .....	189
ML with Athena (Preview) Syntax .....	189
Querying with UDFs (Preview) .....	190
Considerations and Limitations .....	191
UDF Query Syntax .....	191
Creating and Deploying a UDF Using Lambda .....	193
Querying AWS Service Logs .....	198
Querying Application Load Balancer Logs .....	198
Querying Classic Load Balancer Logs .....	200
Querying Amazon CloudFront Logs .....	202
Querying AWS CloudTrail Logs .....	203
Querying Amazon EMR Logs .....	208
Querying AWS Global Accelerator Flow Logs .....	211
Querying Amazon GuardDuty Findings .....	213
Querying Network Load Balancer Logs .....	214
Querying Amazon VPC Flow Logs .....	216
Querying AWS WAF Logs .....	218
Querying AWS Glue Data Catalog .....	221
Listing Databases and Searching a Specified Database .....	221
Listing Tables in a Specified Database and Searching for a Table by Name .....	222
Listing Partitions for a Specific Table .....	222
Listing or Searching Columns for a Specified Table or View .....	223
Querying Web Server Logs .....	224
Querying Apache Logs .....	225
Querying IIS Logs .....	226
Security .....	232

Data Protection .....	232
Encryption at Rest .....	233
Encryption in Transit .....	238
Key Management .....	238
Internetwork Traffic Privacy .....	239
Identity and Access Management .....	239
Managed Policies for User Access .....	240
Access through JDBC and ODBC Connections .....	243
Access to Amazon S3 .....	243
Fine-Grained Access to Databases and Tables .....	244
Access to Encrypted Metadata in the Data Catalog .....	250
Cross-account Access to S3 Buckets .....	251
Access to Workgroups and Tags .....	254
Allow Access to an Athena Data Connector for External Hive Metastore .....	254
Allow Lambda Function Access to External Hive Metastores .....	256
Allow Access to Athena Federated Query (Preview) .....	260
Allow Access to Athena UDF .....	264
Allowing Access for ML with Athena (Preview) .....	268
Enabling Federated Access to the Athena API .....	268
Logging and Monitoring .....	271
Compliance Validation .....	272
Resilience .....	272
Infrastructure Security .....	273
Connect to Amazon Athena Using an Interface VPC Endpoint .....	273
Configuration and Vulnerability Analysis .....	274
Using Athena with Lake Formation .....	275
How Lake Formation Data Access Works .....	275
Considerations and Limitations .....	277
Managing User Permissions .....	279
Applying Lake Formation Permissions to Existing Databases and Tables .....	281
Using Lake Formation and JDBC or ODBC for Federated Access .....	281
Using Workgroups to Control Query Access and Costs .....	322
Using Workgroups for Running Queries .....	322
Benefits of Using Workgroups .....	322
How Workgroups Work .....	323
Setting up Workgroups .....	324
IAM Policies for Accessing Workgroups .....	325
Workgroup Settings .....	330
Managing Workgroups .....	331
Athena Workgroup APIs .....	336
Troubleshooting Workgroups .....	336
Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events .....	338
Enabling CloudWatch Query Metrics .....	338
Monitoring Athena Queries with CloudWatch Metrics .....	339
Monitoring Athena Queries with CloudWatch Events .....	342
Setting Data Usage Control Limits .....	344
Tagging Resources .....	348
Tag Basics .....	348
Tag Restrictions .....	348
Working with Tags on Workgroups in the Console .....	349
Displaying Tags for Individual Workgroups .....	349
Adding and Deleting Tags on an Individual Workgroup .....	349
Using Tag Operations .....	350
Managing Tags Using API Operations .....	351
Managing Tags Using the AWS CLI .....	352
Tag-Based IAM Access Control Policies .....	353
Tag Policy Examples for Workgroups .....	353

Tag Policy Examples for Data Catalogs .....	355
Monitoring Logs and Troubleshooting .....	358
Logging Amazon Athena API Calls with AWS CloudTrail .....	358
Athena Information in CloudTrail .....	358
Understanding Athena Log File Entries .....	359
Troubleshooting .....	361
SerDe Reference .....	362
Using a SerDe .....	362
To Use a SerDe in Queries .....	362
Supported SerDes and Data Formats .....	363
Avro SerDe .....	364
Regex SerDe .....	366
CloudTrail SerDe .....	367
OpenCSVSerDe for Processing CSV .....	369
Grok SerDe .....	372
JSON SerDe Libraries .....	374
LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files .....	378
ORC SerDe .....	383
Parquet SerDe .....	386
Compression Formats .....	388
Notes and Resources .....	388
SQL Reference .....	390
Data Types in Athena .....	390
DML Queries, Functions, and Operators .....	391
SELECT .....	391
INSERT INTO .....	396
Presto Functions .....	399
DDL Statements .....	399
Unsupported DDL .....	400
ALTER DATABASE SET DBPROPERTIES .....	401
ALTER TABLE ADD COLUMNS .....	402
ALTER TABLE ADD PARTITION .....	402
ALTER TABLE DROP PARTITION .....	404
ALTER TABLE RENAME PARTITION .....	404
ALTER TABLE SET LOCATION .....	405
ALTER TABLE SET TBLPROPERTIES .....	405
CREATE DATABASE .....	406
CREATE TABLE .....	406
CREATE TABLE AS .....	410
CREATE VIEW .....	412
DESCRIBE TABLE .....	413
DESCRIBE VIEW .....	414
DROP DATABASE .....	414
DROP TABLE .....	414
DROP VIEW .....	415
MSCK REPAIR TABLE .....	415
SHOW COLUMNS .....	417
SHOW CREATE TABLE .....	417
SHOW CREATE VIEW .....	418
SHOW DATABASES .....	418
SHOW PARTITIONS .....	418
SHOW TABLES .....	419
SHOW TBLPROPERTIES .....	420
SHOW VIEWS .....	420
Considerations and Limitations .....	421
Code Samples, Service Quotas, and Previous JDBC Driver .....	423
Code Samples .....	423

Constants .....	423
Create a Client to Access Athena .....	424
Start Query Execution .....	424
Stop Query Execution .....	427
List Query Executions .....	428
Create a Named Query .....	429
Delete a Named Query .....	430
List Named Queries .....	432
Earlier Version JDBC Drivers .....	433
Instructions for JDBC Driver version 1.1.0 .....	434
Service Quotas .....	438
Queries .....	438
Workgroups .....	438
AWS Glue .....	439
Amazon S3 Buckets .....	439
Per Account API Call Quotas .....	439
Release Notes .....	441
July 29, 2020 .....	442
July 9, 2020 .....	442
Querying Apache Hudi Datasets .....	442
AWS CloudFormation Data Catalog Resource .....	442
June 1, 2020 .....	443
Using Apache Hive Metastore as a Metacatalog with Amazon Athena .....	443
May 21, 2020 .....	443
April 1, 2020 .....	443
March 11, 2020 .....	443
March 6, 2020 .....	443
November 26, 2019 .....	444
Federated SQL Queries .....	444
Invoking Machine Learning Models in SQL Queries .....	445
User Defined Functions (UDFs) (Preview) .....	445
Using Apache Hive Metastore as a Metacatalog with Amazon Athena (Preview) .....	446
New Query-Related Metrics .....	446
November 12, 2019 .....	446
November 8, 2019 .....	447
October 8, 2019 .....	447
September 19, 2019 .....	447
September 12, 2019 .....	447
August 16, 2019 .....	448
August 9, 2019 .....	448
June 26, 2019 .....	448
May 24, 2019 .....	448
March 05, 2019 .....	448
February 22, 2019 .....	449
February 18, 2019 .....	450
November 20, 2018 .....	451
October 15, 2018 .....	451
October 10, 2018 .....	452
September 6, 2018 .....	452
August 23, 2018 .....	452
August 16, 2018 .....	453
August 7, 2018 .....	453
June 5, 2018 .....	454
Support for Views .....	454
Improvements and Updates to Error Messages .....	454
Bug Fixes .....	454
May 17, 2018 .....	454

April 19, 2018 .....	455
April 6, 2018 .....	455
March 15, 2018 .....	455
February 2, 2018 .....	455
January 19, 2018 .....	456
November 13, 2017 .....	456
November 1, 2017 .....	457
October 19, 2017 .....	457
October 3, 2017 .....	457
September 25, 2017 .....	457
August 14, 2017 .....	457
August 4, 2017 .....	457
June 22, 2017 .....	457
June 8, 2017 .....	458
May 19, 2017 .....	458
Improvements .....	458
Bug Fixes .....	459
April 4, 2017 .....	459
Features .....	459
Improvements .....	459
Bug Fixes .....	459
March 24, 2017 .....	460
Features .....	460
Improvements .....	460
Bug Fixes .....	460
February 20, 2017 .....	460
Features .....	460
Improvements .....	462
Document History .....	463
AWS glossary .....	472

# 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.

Athena is 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? \(p. 1\)](#)
- [Accessing Athena \(p. 1\)](#)
- [Understanding Tables, Databases, and the Data Catalog \(p. 2\)](#)
- [AWS Service Integrations with Athena \(p. 3\)](#)

## When should I use 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 [Connecting to Amazon Athena with ODBC and JDBC Drivers \(p. 72\)](#).

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 AWS account and integrated with the ETL and data discovery features of AWS Glue. For more information, see [Integration with AWS Glue \(p. 16\)](#) and [What is AWS Glue](#) in the *AWS Glue Developer Guide*.

For a list of AWS services that Athena leverages or integrates with, see [the section called “AWS Service Integrations with Athena” \(p. 3\)](#).

## Accessing Athena

You can access Athena using 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 with the console, see [Getting Started \(p. 8\)](#).
- To learn how to use JDBC or ODBC drivers, see [Connecting to Amazon Athena with JDBC \(p. 72\)](#) and [Connecting to Amazon Athena with ODBC \(p. 73\)](#).
- 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 [AWS 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 Version 3 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](#).

## Understanding Tables, Databases, and the Data Catalog

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

Regardless of how the tables are created, the tables 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.

- To create a table automatically, use an AWS Glue crawler from within Athena. For more information about AWS Glue and crawlers, see [Integration with AWS Glue \(p. 16\)](#). 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.

After you create a table, you can use [SQL SELECT \(p. 391\)](#) statements to query it, including getting [specific file locations for your source data \(p. 394\)](#). Your query results are stored in Amazon S3 in [the query result location that you specify \(p. 115\)](#).

The AWS Glue Data Catalog is accessible throughout your AWS 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. In addition, AWS Glue lets you automatically discover data schema and extract, transform, and load (ETL) data.

- 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.

### Note

If you have tables in Athena created before August 14, 2017, they were created in an Athena-managed internal data catalog that exists side-by-side with the AWS Glue Data Catalog until you choose to update. For more information, see [Upgrading to the AWS Glue Data Catalog Step-by-Step \(p. 29\)](#).

When you query an existing table, under the hood, Amazon Athena uses Presto, a distributed SQL engine. We have examples with sample data within Athena to show you how to create a table and then issue a query against it using Athena. Athena also has a tutorial in the console that helps you get started creating a table based on data that is stored in Amazon S3.

- For a step-by-step tutorial on creating a table and writing queries in the Athena Query Editor, see [Getting Started \(p. 8\)](#).
- Run the Athena tutorial in the console. This launches automatically if you log in to <https://console.aws.amazon.com/athena/> for the first time. You can also choose **Tutorial** in the console to launch it.

## AWS Service Integrations with Athena

You can query data from other AWS services in Athena. Athena leverages several AWS services. For more information, see the following table.

**Note**

To see the list of supported regions for each service, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

AWS Service	Topic	Description
AWS CloudTrail	<a href="#">Querying AWS CloudTrail Logs (p. 203)</a>	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 automatically create tables for querying logs directly from the CloudTrail console, and use those tables to run queries in Athena. For more information, see <a href="#">Creating a Table for CloudTrail Logs in the CloudTrail Console (p. 205)</a> .
Amazon CloudFront	<a href="#">Querying Amazon CloudFront Logs (p. 202)</a>	Use Athena to query Amazon CloudFront.
Elastic Load Balancing	<ul style="list-style-type: none"><li>• <a href="#">Querying Application Load Balancer Logs (p. 198)</a></li><li>• <a href="#">Querying Classic Load Balancer Logs (p. 200)</a></li></ul>	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. See <a href="#">Creating the Table for ALB Logs (p. 199)</a>  Query Classic Load Balancer logs to analyze and understand traffic patterns to and from Elastic Load Balancing instances



AWS Service	Topic	Description
		and backend applications. You can see the source of traffic, latency, and bytes transferred. See <a href="#">Creating the Table for ELB Logs</a> (p. 200).
Amazon Virtual Private Cloud	<a href="#">Querying Amazon VPC Flow Logs</a> (p. 216)	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.
AWS CloudFormation	<a href="#">AWS::Athena::DataCatalog</a> in the <i>AWS CloudFormation User Guide</i> .	Specify an Athena data catalog, including a name, description, type, parameters, and tags. For more information, see <a href="#">DataCatalog</a> in the <i>Amazon Athena API Reference</i> .
	<a href="#">AWS::Athena::WorkGroup</a> in the <i>AWS CloudFormation User Guide</i> .	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 <a href="#">Using Workgroups to Control Query Access and Costs</a> (p. 322) in the <i>Amazon Athena User Guide</i> and <a href="#">CreateWorkGroup</a> in the <i>Amazon Athena API Reference</i> .
	<a href="#">AWS::Athena::NamedQuery</a> in the <i>AWS CloudFormation User Guide</i> .	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 information, see <a href="#">CreateNamedQuery</a> in the <i>Amazon Athena API Reference</i> .

AWS Service	Topic	Description
AWS Glue Data Catalog	<a href="#">Integration with AWS Glue (p. 16)</a>	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 AWS account and integrated with the ETL and data discovery features of AWS Glue. For more information, see <a href="#">Integration with AWS Glue (p. 16)</a> and <a href="#">What is AWS Glue</a> in the <i>AWS Glue Developer Guide</i> .
Amazon QuickSight	<a href="#">Connecting to Amazon Athena with ODBC and JDBC Drivers (p. 72)</a>	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 <a href="#">What is Amazon QuickSight</a> in the <i>Amazon QuickSight User Guide</i> and <a href="#">Connecting to Amazon Athena with ODBC and JDBC Drivers (p. 72)</a> .
IAM	<a href="#">Actions for Amazon Athena</a>	You can use Athena API actions in IAM permission policies. See <a href="#">Actions for Amazon Athena</a> and <a href="#">Identity and Access Management in Athena (p. 239)</a> .
AWS Systems Manager Inventory	<a href="#">Querying inventory data from multiple Regions and accounts in the AWS Systems Manager User Guide.</a>	AWS Systems Manager Inventory integrates with Amazon Athena to help you query inventory data from multiple AWS Regions and accounts.
Amazon S3 Inventory	<a href="#">Querying inventory with Athena in the Amazon Simple Storage Service Developer Guide.</a>	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 <a href="#">Amazon S3 inventory</a> in the <i>Amazon Simple Storage Service Developer Guide</i> .

# Setting Up

If you've already signed up for Amazon Web Services (AWS), you can start using Amazon Athena immediately. If you haven't signed up for AWS, or if you need assistance querying data using Athena, first complete the tasks below:

## Sign Up for AWS

When you sign up for AWS, your account is automatically signed up for all services in AWS, including Athena. You are charged only for the services that you use. When you use Athena, you use Amazon S3 to store your data. Athena has no AWS Free Tier pricing.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

### To create an AWS account

1. Open <http://aws.amazon.com/>, and then choose **Create an AWS Account**.
2. Follow the online instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account number, because you need it for the next task.

## Create an IAM User

An AWS Identity and Access Management (IAM) user is an account that you create to access services. It is a different user than your main AWS account. As a security best practice, we recommend that you use the IAM user's credentials to access AWS services. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console. If you aren't familiar with using the console, see [Working with the AWS Management Console](#).

### To create a group for administrators

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Groups**, **Create New Group**.
3. For **Group Name**, type a name for your group, such as **Administrators**, and choose **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** field to filter the list of policies.
5. Choose **Next Step**, **Create Group**. Your new group is listed under **Group Name**.

## To create an IAM user for yourself, add the user to the administrators group, and create a password for the user

1. In the navigation pane, choose **Users**, and then **Create New Users**.
2. For 1, type a user name.
3. Clear the check box next to **Generate an access key for each user** and then **Create**.
4. In the list of users, select the name (not the check box) of the user you just created. You can use the **Search** field to search for the user name.
5. Choose **Groups, Add User to Groups**.
6. Select the check box next to the administrators and choose **Add to Groups**.
7. Choose the **Security Credentials** tab. Under **Sign-In Credentials**, choose **Manage Password**.
8. Choose **Assign a custom password**. Then type a password in the **Password** and **Confirm Password** fields. When you are finished, choose **Apply**.
9. To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where `your_aws_account_id` is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

```
https://*your_account_alias*.signin.aws.amazon.com/console/
```

It is also possible the sign-in link will use your account name instead of number. To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

## Attach Managed Policies for Using Athena

Attach Athena managed policies to the IAM account you use to access Athena. There are two managed policies for Athena: `AmazonAthenaFullAccess` and `AWSQuicksightAthenaAccess`. These policies grant permissions to Athena to query Amazon S3 as well as write the results of your queries to a separate bucket on your behalf. For more information and step-by-step instructions, see [Adding IAM Identity Permissions \(Console\)](#) in the *AWS Identity and Access Management User Guide*. For information about policy contents, see [IAM Policies for User Access \(p. 240\)](#).

### 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 an identity-based policy. For more information, see [Amazon S3 Permissions \(p. 243\)](#). If the dataset or Athena query results are encrypted, you may need additional permissions. For more information, see [Configuring Encryption Options \(p. 233\)](#).

# Getting 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 is using 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

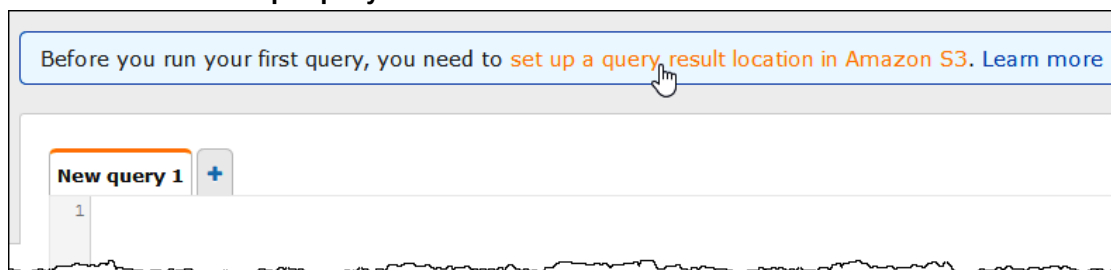
- If you have not already done so, sign up for an account in [Setting Up \(p. 6\)](#).
- Using the same AWS Region (for example, US West (Oregon)) and account that you are using for Athena, [Create a bucket in Amazon S3](#) to hold your query results from Athena.

## Step 1: Create a Database

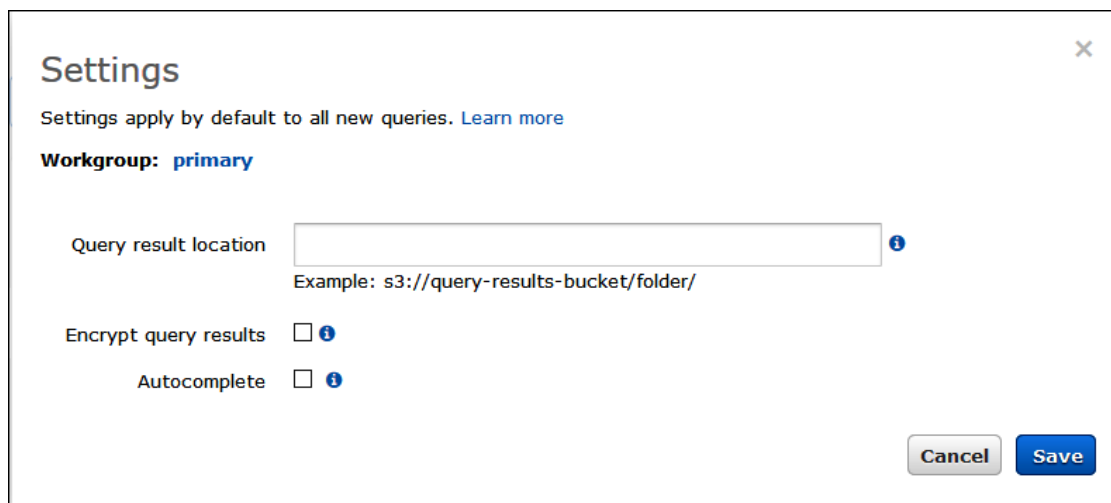
You first need to create a database in Athena.

### To create a database

1. Open the Athena console.
2. If this is your first time visiting the Athena console, you'll go to a Getting Started page. Choose **Get Started** to open the Query Editor. If it isn't your first time, the Athena Query Editor opens.
3. Choose the link to **set up a query result location in Amazon S3**.

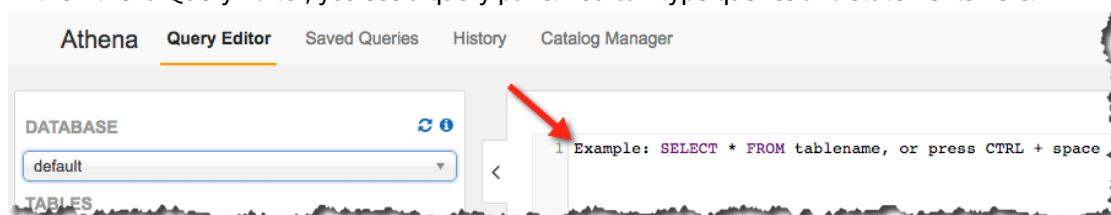


4. In the **Settings** dialog box, enter the path to the bucket that you created in Amazon S3 for your query results. Prefix the path with `s3://` and add a forward slash to the end of the path.



The screenshot shows the 'Settings' dialog box in Amazon Athena. At the top, it says 'Settings apply by default to all new queries. [Learn more](#)'. Below this, the 'Workgroup' is set to 'primary'. There are three settings: 'Query result location' with a text input field and an example 's3://query-results-bucket/folder/', 'Encrypt query results' with an unchecked checkbox, and 'Autocomplete' with an unchecked checkbox. At the bottom right are 'Cancel' and 'Save' buttons.

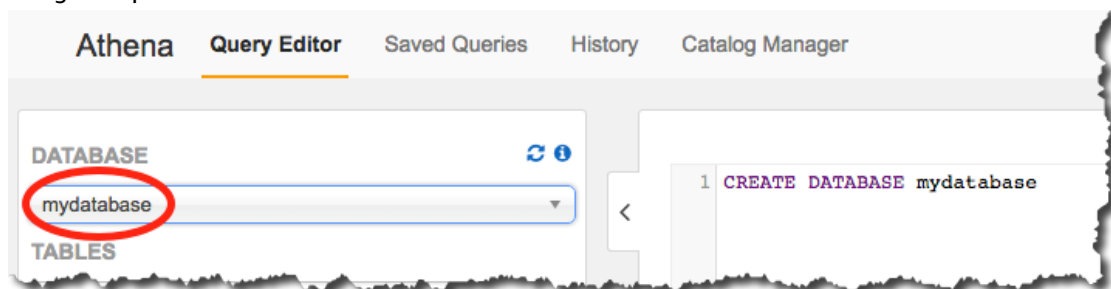
5. Click **Save**.
6. In the Athena Query Editor, you see a query pane. You can type queries and statements here.



7. To create a database named mydatabase, enter the following CREATE DATABASE statement.

```
CREATE DATABASE mydatabase
```

8. Choose **Run Query** or press **Ctrl+ENTER**.
9. Confirm that the catalog display refreshes and mydatabase appears in the **Database** list in the navigation pane on the left.

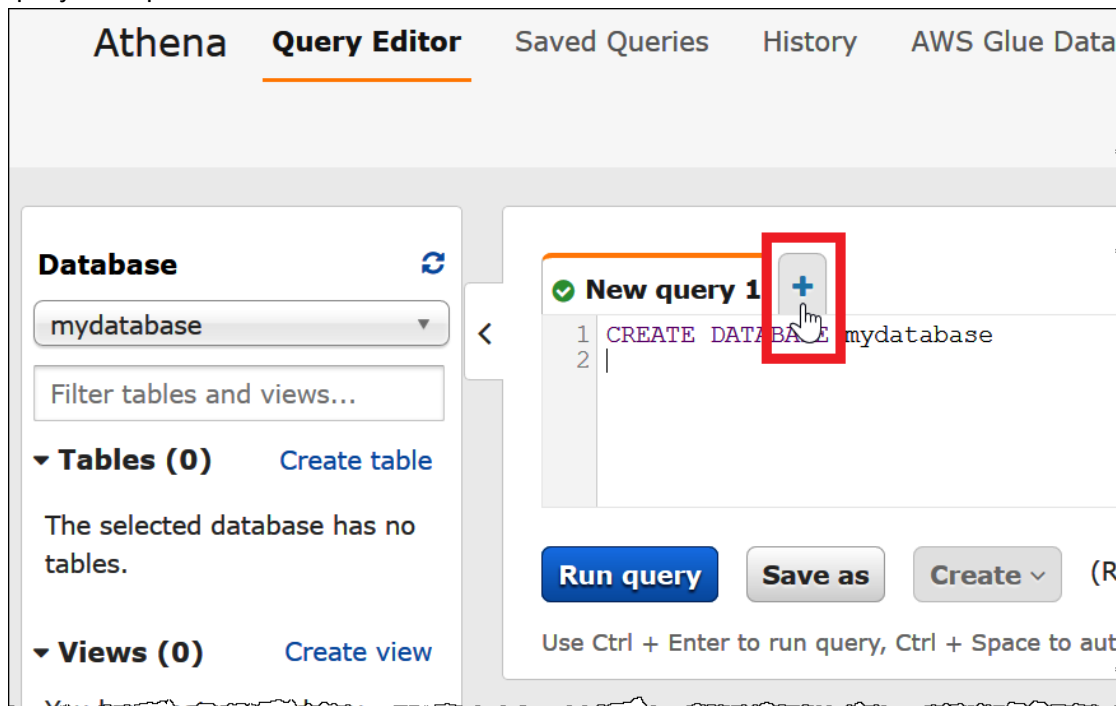


## Step 2: Create a Table

Now that you have a database, you're ready to run a statement to create a table. The table will be based on Athena sample data in the location `s3://athena-examples-aws-region/cloudfront/plaintext/`. 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.

## To create a table

1. For **Database**, choose mydatabase.
2. Choose the plus (+) sign in the Query Editor to create a tab with a new query. You can have up to ten query tabs open at once.



3. In the query pane, enter the following `CREATE TABLE` statement. In the `LOCATION` statement at the end of the query, replace `myregion` with the AWS Region that you are currently using (for example, `us-west-1`).

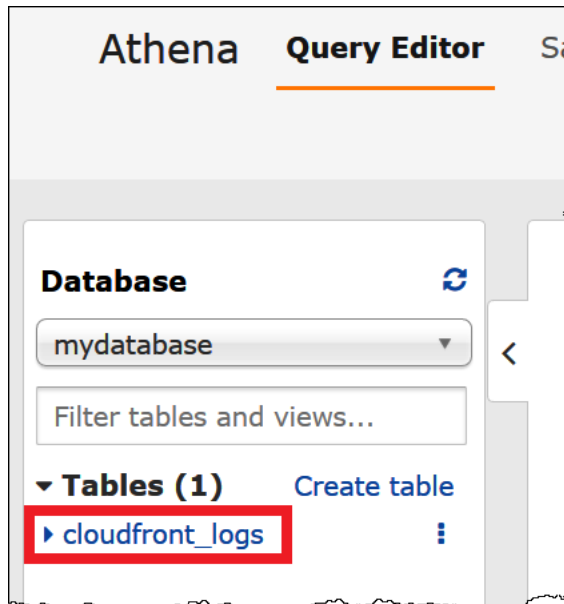
```
Create External Table if NOT EXISTS cloudfront_logs (  
    `Date` DATE,  
    Time STRING,  
    Location STRING,  
    Bytes INT,  
    RequestIP STRING,  
    Method STRING,  
    Host STRING,  
    Uri STRING,  
    Status INT,  
    Referrer STRING,  
    os STRING,  
    Browser STRING,  
    BrowserVersion STRING  
) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'  
WITH SERDEPROPERTIES (  
    "input.regex" = "^(?![#])([^ ]+)\s+([^\s]+\s)+([^\s]+\s)+([^\s]+\s)+([^\s]+\s)+\n\n+([\n\r]+\s+)([\n\r]+\s+)([^\s]+\s)+([^\s]+\s)+([^\s]+\s)+(\n|\r|[\n\r;]+).*%20(^[^\n\r\/][^\/](.*)$" ) LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

### 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/`.

4. Choose **Run Query**.

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 Amazon Athena \(p. 390\)](#).

### To run a query


1. 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 Query**.

The results look like the following:

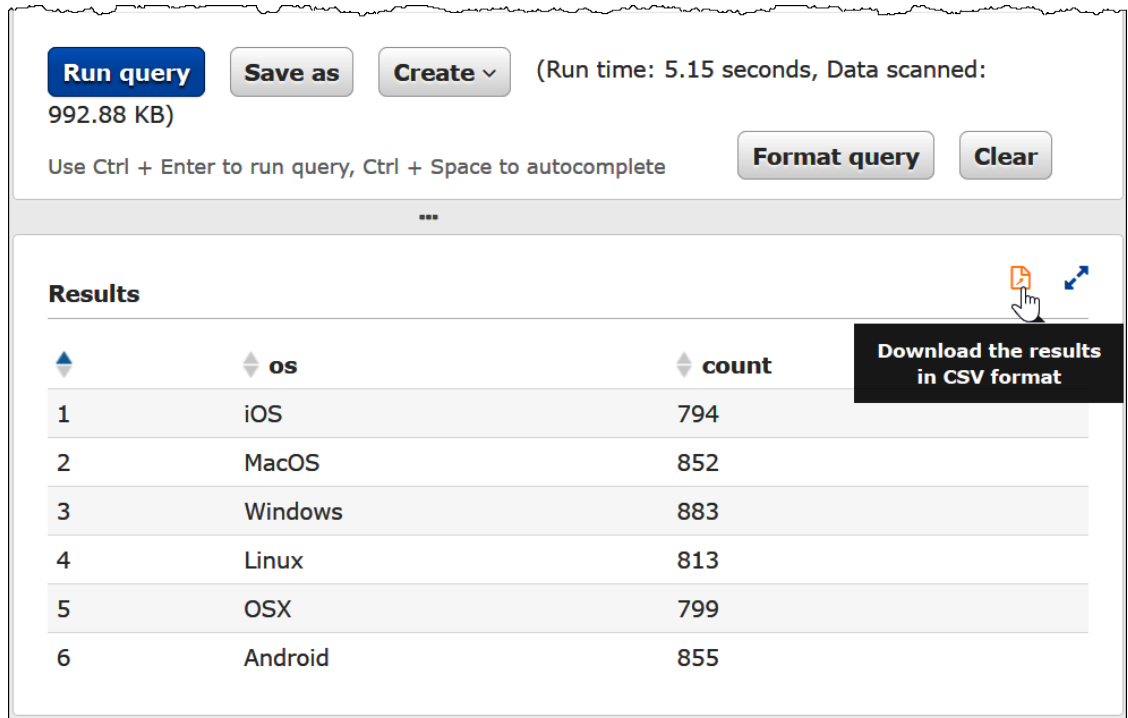




The screenshot shows a 'Results' pane with a table containing two columns: 'os' and 'count'. The table has six rows of data. The first row is highlighted in light blue.

	os	count
1	iOS	794
2	MacOS	852
3	OSX	799
4	Windows	883
5	Linux	813
6	Android	855

3. You can save the results of the query to a .csv file by choosing the download icon on the **Results** pane.



The screenshot shows the Amazon Athena console interface. At the top, there are buttons for 'Run query', 'Save as', and 'Create', along with query statistics: '(Run time: 5.15 seconds, Data scanned: 992.88 KB)'. Below these are buttons for 'Format query' and 'Clear'. The 'Results' pane is visible, showing a table with columns 'os' and 'count'. A download icon (a document with a download arrow) is located in the top right corner of the Results pane. A tooltip is displayed over the download icon, stating 'Download the results in CSV format'.

Run query Save as Create (Run time: 5.15 seconds, Data scanned: 992.88 KB)

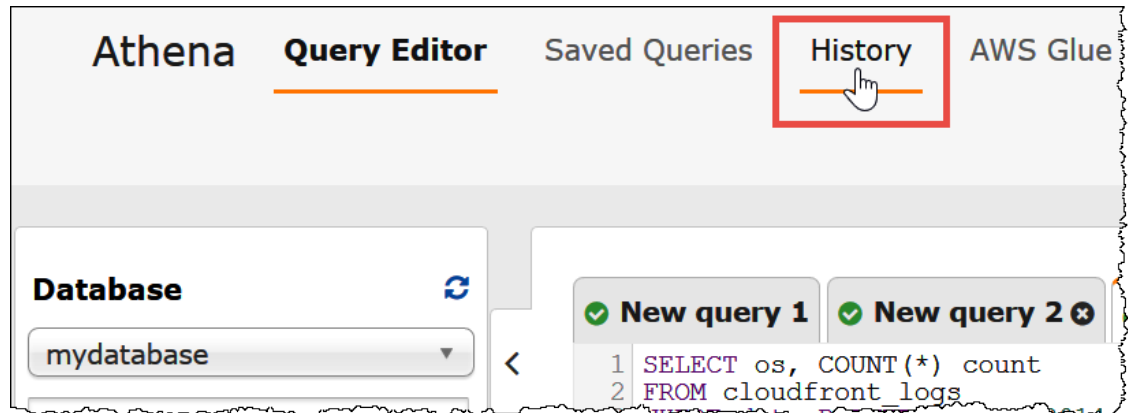
Use Ctrl + Enter to run query, Ctrl + Space to autocomplete Format query Clear

Results

	os	count
1	iOS	794
2	MacOS	852
3	Windows	883
4	Linux	813
5	OSX	799
6	Android	855

Download the results in CSV format

4. Choose the **History** tab to view your previous queries.



- Choose **Download results** to download the results of a previous query. Query history is retained for 45 days.

History						
Search for name, query, etc.						
Query submitted time	Query	Encryption type	State	Run time(s)	Data scanned	Action
2020/04/02 17:18:54 UTC-7	SELECT os, COUNT(*) count FROM cloudfront_logs WHERE date BETWEEN date '2014-07-05' AND date '2014-0...	N/A	Succeeded	5.26	992.88 KB	Download results
2020/04/02 17:05:51 UTC-7	CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs ( `Date` DATE, Time STRING, Location STRIN...	N/A	Succeeded	0.37	0 KB	Download results
2020/04/02 16:14:59 UTC-7	CREATE DATABASE mydatabase	N/A	Succeeded	0.42	0 KB	Download results

For more information, see [Working with Query Results, Output Files, and Query History](#) (p. 110).

## Connecting to Other Data Sources

This tutorial used a data source in Amazon S3 in CSV format. You can connect Athena to a variety of data sources by using AWS Glue, ODBC and JDBC drivers, external Hive metastores, and Athena data source connectors. For more information, see [Connecting to Data Sources \(p. 16\)](#).

# Accessing Amazon Athena

You can access Amazon Athena using the AWS Management Console, the Amazon Athena API, or the AWS CLI.

## Using the Console

You can use the AWS Management Console for Amazon Athena to do the following:

- Create or select a database.
- Create, view, and delete tables.
- Filter tables by starting to type their names.
- Preview tables and generate CREATE TABLE DDL for them.
- Show table properties.
- Run queries on tables, save and format queries, and view query history.
- Create up to ten queries using different query tabs in the query editor. To open a new tab, click the plus sign.
- Display query results, save, and export them.
- Access the AWS Glue Data Catalog.
- View and change settings, such as view the query result location, configure auto-complete, and encrypt query results.

In the right pane, the Query Editor displays an introductory screen that prompts you to create your first table. You can view your tables under **Tables** in the left pane.

Here's a high-level overview of the actions available for each table:

- **Preview tables** – View the query syntax in the Query Editor on the right.
- **Show properties** – Show a table's name, its location in Amazon S3, input and output formats, the serialization (SerDe) library used, and whether the table has encrypted data.
- **Delete table** – Delete a table.
- **Generate CREATE TABLE DDL** – Generate the query behind a table and view it in the query editor.

## Using the API

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 examples of using the AWS SDK for Java with Athena, see [Code Samples \(p. 423\)](#).

For more information about AWS SDK for Java documentation and downloads, see the *SDKs* section in [Tools for Amazon Web Services](#).

## Using the CLI

You can access Amazon Athena using the AWS CLI. For more information, see the [AWS CLI Reference for Athena](#).

# Connecting 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.

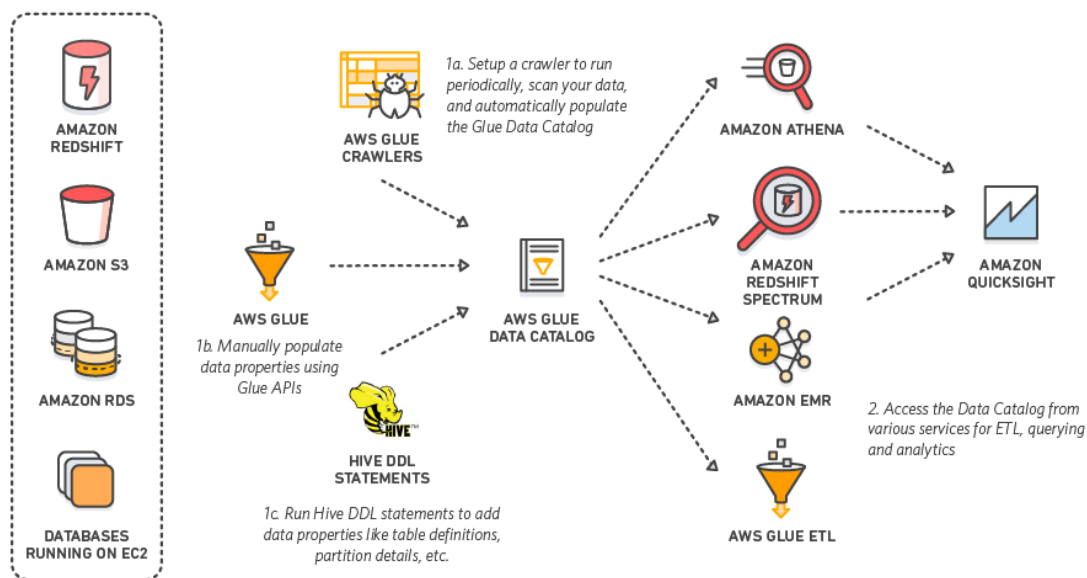
## Topics

- [Integration with AWS Glue \(p. 16\)](#)
- [Using Athena Data Connector for External Hive Metastore \(p. 34\)](#)
- [Using Amazon Athena Federated Query \(Preview\) \(p. 56\)](#)
- [IAM Policies for Accessing Data Catalogs \(p. 65\)](#)
- [Managing Data Sources \(p. 69\)](#)
- [Connecting to Amazon Athena with ODBC and JDBC Drivers \(p. 72\)](#)

## Integration with AWS Glue

[AWS Glue](#) is a fully managed ETL (extract, transform, and load) service that can categorize your data, clean it, enrich it, and move it reliably between various data stores. AWS Glue crawlers automatically infer database and table schema from your dataset, storing the associated metadata in the AWS Glue Data Catalog.

Athena natively supports querying datasets and data sources that are registered with the AWS Glue Data Catalog. When you run Data Manipulation Language (DML) queries in Athena with the Data Catalog as your source, you are using the Data Catalog schema to derive insight from the underlying dataset. When you run Data Definition Language (DDL) queries, the schema you define are defined in the AWS Glue Data Catalog. From within Athena, you can also run a AWS Glue crawler on a data source to create schema in the AWS Glue Data Catalog.



In regions where AWS Glue is supported, Athena uses the AWS Glue Data Catalog as a central location to store and retrieve table metadata throughout an AWS account. The Athena query engine requires table metadata that instructs it where to read data, how to read it, and other information necessary to process the data. The AWS Glue Data Catalog provides a unified metadata repository across a variety of data sources and data formats, integrating not only with Athena, but with Amazon S3, Amazon RDS, Amazon Redshift, Amazon Redshift Spectrum, Amazon EMR, and any application compatible with the Apache Hive metastore.

For more information about the AWS Glue Data Catalog, see [Populating the AWS Glue Data Catalog](#) in the *AWS Glue Developer Guide*. For a list of regions where AWS Glue is available, see [Regions and Endpoints](#) in the *AWS General Reference*.

Separate charges apply to AWS Glue. For more information, see [AWS Glue Pricing](#) and [Are there separate charges for AWS Glue?](#) (p. 33) For more information about the benefits of using AWS Glue with Athena, see [Why should I upgrade to the AWS Glue Data Catalog?](#) (p. 32)

### Topics

- [Using AWS Glue to Connect to Data Sources in Amazon S3](#) (p. 17)
- [Best Practices When Using Athena with AWS Glue](#) (p. 20)
- [Upgrading to the AWS Glue Data Catalog Step-by-Step](#) (p. 29)
- [FAQ: Upgrading to the AWS Glue Data Catalog](#) (p. 32)

## Using AWS Glue to Connect to Data Sources in Amazon S3

Athena can connect to your data stored in Amazon S3 using the AWS Glue Data Catalog to store metadata such as table and column names. After the connection is made, your databases, tables, and views appear in Athena's query editor.

To define schema information for AWS Glue to use, you can set up an AWS Glue crawler to retrieve the information automatically, or you can manually add a table and enter the schema information.

## Setting up a Crawler

You set up a crawler by starting in the Athena console and then using the AWS Glue console in an integrated way. When you create a crawler, you can choose data stores to crawl or point the crawler to existing catalog tables.

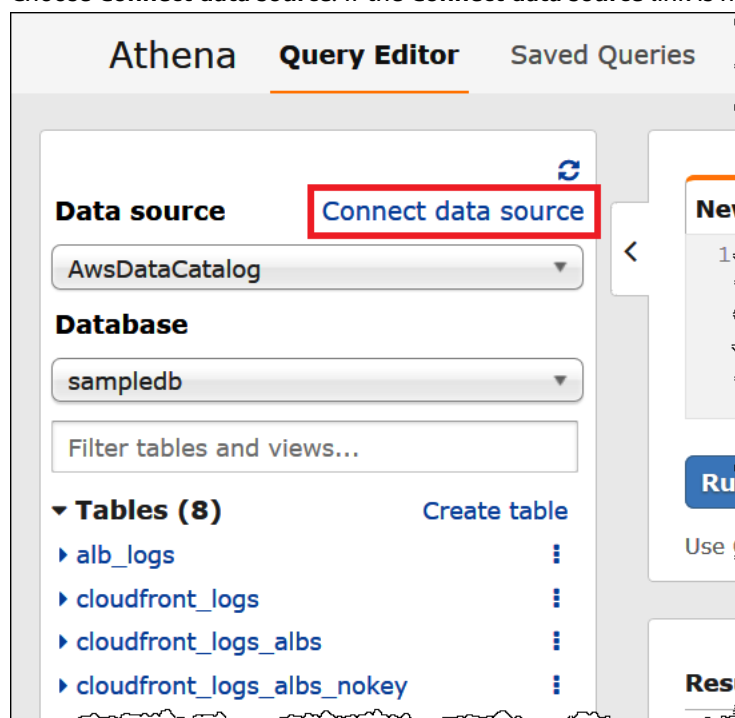
### Note

The steps for setting up a crawler depend on the options available in the Athena console. If the **Connect data source** link in **Option A** is not available, use the procedure in **Option B**.

### Option A

#### Option A: To set up a crawler in AWS Glue using the *Connect data source* link

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **Connect data source**. If the **Connect data source** link is not present, use **Option B**.



3. On the **Connect data source** page, choose **AWS Glue Data Catalog**.
4. Click **Next**.
5. On the **Connection details** page, choose **Set up crawler in AWS Glue to retrieve schema information automatically**.
6. Click **Connect to AWS AWS Glue**.
7. On the **AWS Glue console Add crawler** page, follow the steps to create a crawler.

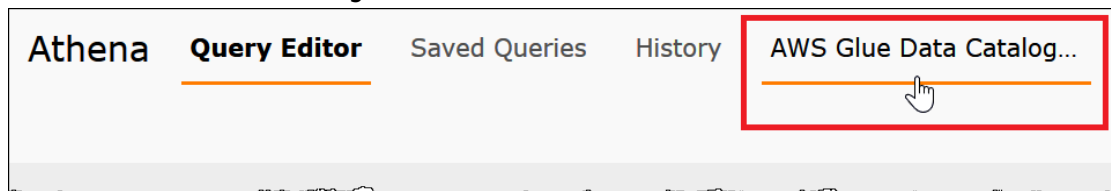
For more information, see [Populating the AWS Glue Data Catalog](#).

### Option B

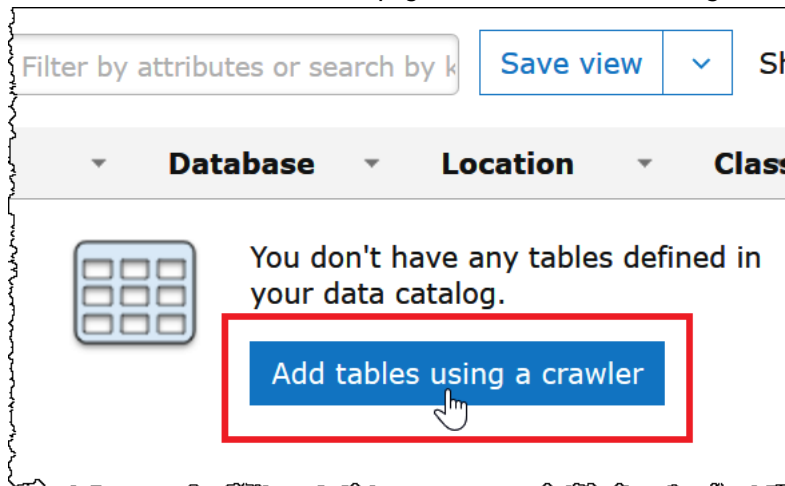
Use the following procedure to set up a AWS Glue crawler if the **Connect data source** link in **Option A** is not available in the Athena console.

**Option B: To set up a crawler in AWS Glue from the *AWS Glue Data Catalog* link**

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **AWS Glue Data Catalog**.



3. On the AWS Glue console **Tables** page, choose **Add tables using a crawler**.



4. On the **AWS Glue** console **Add crawler** page, follow the steps to create a crawler.

For more information, see [Populating the AWS Glue Data Catalog](#).

## Adding a Schema Table Manually

The following procedure shows you how to use the Athena console to add a table manually.

### To add a table and enter schema information manually

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **Connect data source**.
3. On the **Connect data source** page, choose **AWS Glue Data Catalog**.
4. Click **Next**.
5. On the **Connection details** page, choose **Add a table and enter schema information manually**.
6. Click **Continue to add table**.
7. On the **Add table** page of the Athena console, for **Database**, choose an existing database or create a new one.
8. Enter or choose a table name.
9. For **Location of Input Data Set**, specify the path in Amazon S3 to the folder that contains the dataset that you want to process.
10. Click **Next**.
11. For **Data Format**, choose a data format (**Apache Web Logs**, **CSV**, **TSV**, **Text File with Custom Delimiters**, **JSON**, **Parquet**, or **ORC**).



- For the **Apache Web Logs** option, you must also enter a regex expression in the **Regex** box.
  - For the **Text File with Custom Delimiters** option, specify a **Field terminator** (that is, a column delimiter). Optionally, you can specify a **Collection terminator** for array types or a **Map key terminator**.
12. For **Columns**, specify a column name and the column data type.
- 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**.
13. Choose **Next**.
14. (Optional) For **Partitions**, click **Add a partition** to add column names and data types.
15. Choose **Create table**. The DDL for the table that you specified appears in the **Query Editor**. The following example shows the DDL generated for a two-column table in CSV format:
- ```
CREATE EXTERNAL TABLE IF NOT EXISTS MyManualDB.MyManualTable (  
  `cola` string,  
  `colb` string  
)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'  
WITH SERDEPROPERTIES (  
  'serialization.format' = ',',  
  'field.delim' = ','  
) LOCATION 's3://bucket_name/'  
TBLPROPERTIES ('has_encrypted_data'='false');
```
16. Choose **Run query** to create the table.

## Best Practices When Using Athena with AWS Glue

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. This topic provides considerations and best practices when using either method.

Under the hood, Athena uses Presto to process DML statements and Hive to process the DDL statements that create and modify schema. With these technologies, there are a couple of conventions to follow so that Athena and AWS Glue work well together.

### In this topic

- [Database, Table, and Column Names \(p. 21\)](#)
- [Using AWS Glue Crawlers \(p. 21\)](#)
  - [Scheduling a Crawler to Keep the AWS Glue Data Catalog and Amazon S3 in Sync \(p. 21\)](#)
  - [Using Multiple Data Sources with Crawlers \(p. 22\)](#)
  - [Syncing Partition Schema to Avoid "HIVE\\_PARTITION\\_SCHEMA\\_MISMATCH" \(p. 24\)](#)
  - [Updating Table Metadata \(p. 24\)](#)
- [Working with CSV Files \(p. 25\)](#)
  - [CSV Data Enclosed in Quotes \(p. 25\)](#)
  - [CSV Files with Headers \(p. 27\)](#)
- [Working with Geospatial Data \(p. 27\)](#)
- [Using AWS Glue Jobs for ETL with Athena \(p. 27\)](#)
  - [Creating Tables Using Athena for AWS Glue ETL Jobs \(p. 28\)](#)

- [Using ETL Jobs to Optimize Query Performance](#) (p. 29)
- [Converting SMALLINT and TINYINT Datatypes to INT When Converting to ORC](#) (p. 29)
- [Automating AWS Glue Jobs for ETL](#) (p. 29)

## Database, Table, and Column Names

When you create schema in AWS Glue to query in Athena, consider the following:

- A database name cannot be longer than 252 characters.
- A table name cannot be longer than 255 characters.
- A column name cannot be longer than 128 characters.
- The only acceptable characters for database names, table names, and column names are lowercase letters, numbers, and the underscore character.

You can use the AWS Glue Catalog Manager to rename columns, but at this time table names and database names cannot be changed using the AWS Glue console. To correct database names, you need to create a new database and copy tables to it (in other words, copy the metadata to a new entity). You can follow a similar process for tables. You can use the AWS Glue SDK or AWS CLI to do this.

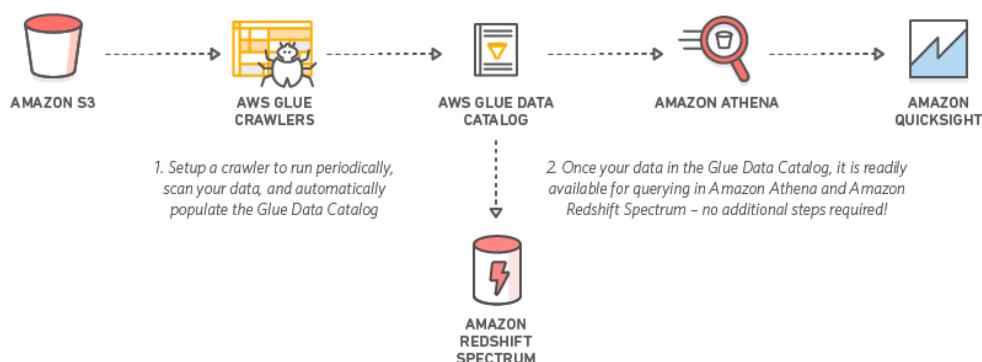
## Using AWS Glue Crawlers

AWS Glue crawlers help discover and register the schema for datasets in the AWS Glue Data Catalog. The crawlers go through your data, and inspect portions of it to determine the schema. In addition, the crawler can detect and register partitions. For more information, see [Cataloging Data with a Crawler](#) in the *AWS Glue Developer Guide*.

## Scheduling 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*.



## Using Multiple Data Sources with Crawlers

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:

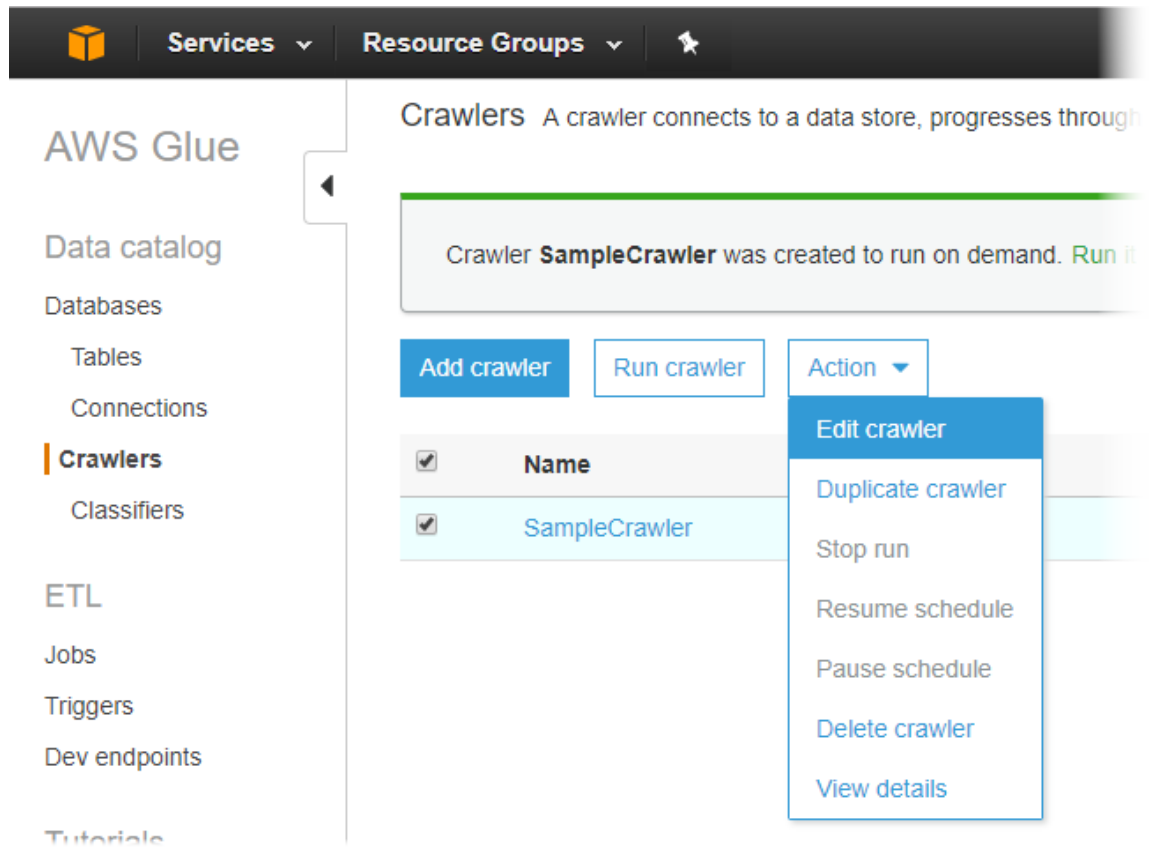
```
s3://bucket01/folder1/table1/partition1/file.txt  
s3://bucket01/folder1/table1/partition2/file.txt  
s3://bucket01/folder1/table1/partition3/file.txt  
s3://bucket01/folder1/table2/partition4/file.txt  
s3://bucket01/folder1/table2/partition5/file.txt
```

If the schema for `table1` and `table2` are similar, and a single data source is set to `s3://bucket01/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`.

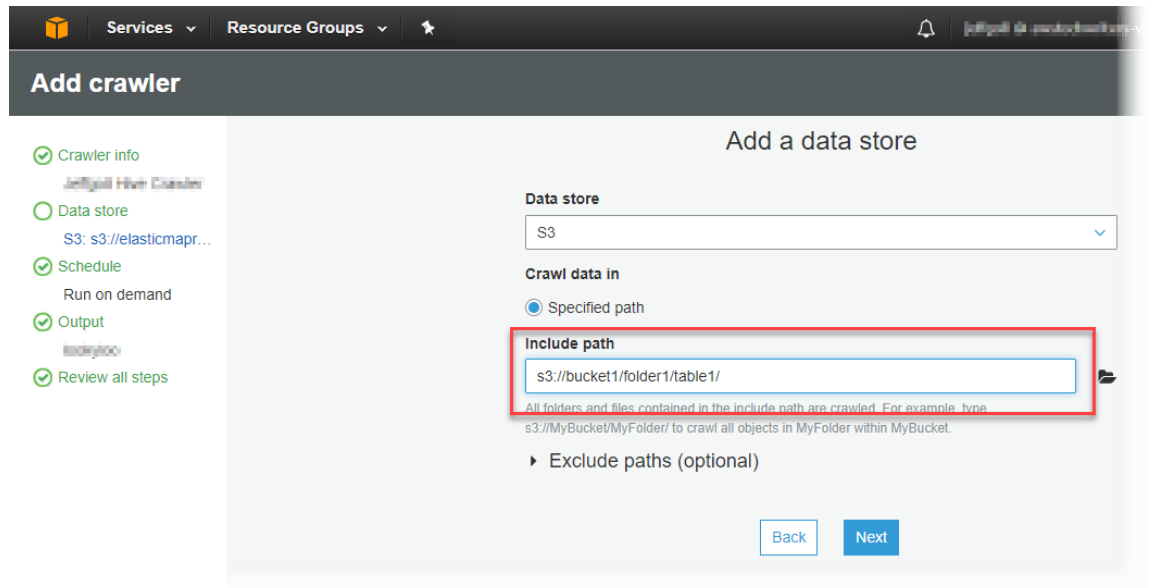
To have the AWS Glue crawler create two separate tables, set the crawler to have two data sources, `s3://bucket01/folder1/table1/` and `s3://bucket01/folder1/table2`, as shown in the following procedure.

### To add another 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. Choose **Crawlers**, select your crawler, and then choose **Action**, **Edit crawler**.



3. Under **Add information about your crawler**, choose additional settings as appropriate, and then choose **Next**.
4. Under **Add a data store**, change **Include path** to the table-level directory. For instance, given the example above, you would change it from `s3://bucket01/folder1` to `s3://bucket01/folder1/table1/`. Choose **Next**.



5. For **Add another data store**, choose **Yes, Next**.

6. For **Include path**, enter your other table-level directory (for example, `s3://bucket01/folder1/table2/`) and choose **Next**.
  - a. Repeat steps 3-5 for any additional table-level directories, and finish the crawler configuration.

The new values for **Include locations** appear under data stores as follows:

| Crawler info |                                                   |
|--------------|---------------------------------------------------|
| Name         | SampleCrawler                                     |
| Service role | arn:aws:iam::000000000000:role/AWSGlueServiceRole |

| Data stores   |                              |
|---------------|------------------------------|
| Data store    | S3                           |
| Include path  | s3://bucket1/folder1/table1/ |
| Exclude paths |                              |
| Data store    | S3                           |
| Include path  | s3://bucket1/folder1/table2/ |
| Exclude paths |                              |

## Syncing Partition Schema to Avoid "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. For more information, see [Using Multiple Data Sources with Crawlers](#) (p. 22).

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.

## Updating Table Metadata

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 [CSV Data Enclosed in quotes](#) (p. 25).

## Working with CSV Files

CSV files occasionally have quotes around the data values intended for each column, and there may be header values included in CSV files, which aren't part of the data to be analyzed. When you use AWS Glue to create schema from these files, follow the guidance in this section.

### CSV Data Enclosed in Quotes

You might have a CSV file that has data fields enclosed in double quotes like 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 `OpenCSVSerDe`, see [OpenCSVSerDe for Processing CSV \(p. 369\)](#).

#### To edit table properties in the AWS Glue console

1. In the AWS Glue console navigation pane, choose **Tables**.
2. Choose the table that you want to edit, and then choose **Edit table**.
3. In the **Edit table details** dialog box, make the following changes:
  - For **Serde 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 (`,`).

×

## Edit table details

Table name

sample\_csv\_table

Input format

org.apache.hadoop.mapred.TextInputFormat

Output format

org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat

Serde name

Serde serialization lib

org.apache.hadoop.hive.serde2.OpenCSVSerde

Serde parameters

| Key           | Value         |   |
|---------------|---------------|---|
| escapeChar    | \             | × |
| quoteChar     | "             | × |
| separatorChar | ,             | × |
| Type key...   | Type value... |   |

Description

Apply

For more information, see [Viewing and Editing Table Details](#) in the *AWS Glue Developer Guide*.

## Updating AWS Glue Table Properties Programmatically

You can use the AWS Glue [UpdateTable](#) API operation or `update-table` CLI command to modify the `SerDeInfo` block in the table definition, as in the following example JSON.

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

## 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://my_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 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://MYBUCKET/MYTABLEDATA/"}, format = "csv",
format_options = {"writeHeader": False}, transformation_ctx = "datasink2")
```

## Working with Geospatial Data

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 [Using AWS Glue Crawlers](#) (p. 21) and [Working with CSV Files](#) (p. 25). 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 [Querying Geospatial Data](#) (p. 167).

## Using AWS Glue Jobs for ETL with Athena

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 Glue](#) in the *AWS Glue Developer Guide*.



## Creating Tables Using Athena for AWS Glue ETL Jobs

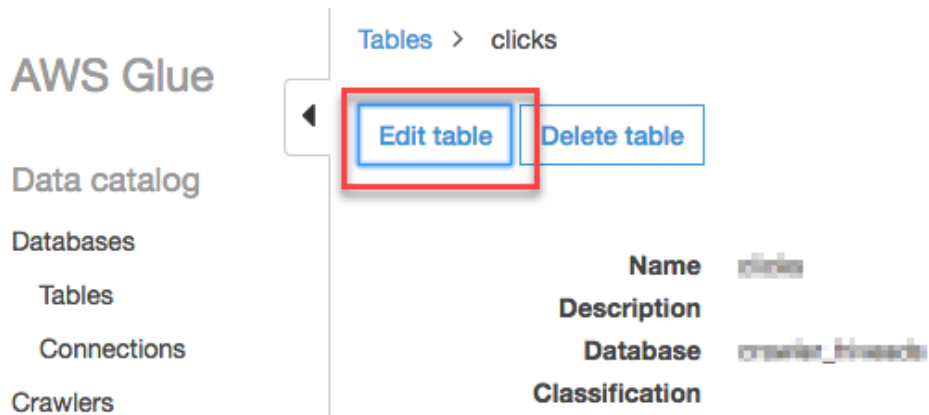
Tables that you create in Athena must have a table property added to them called a `classification`, which identifies the format of the data. This allows AWS Glue to use the tables for ETL jobs. The classification values can be `csv`, `parquet`, `orc`, `avro`, or `json`. An example `CREATE TABLE` statement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

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

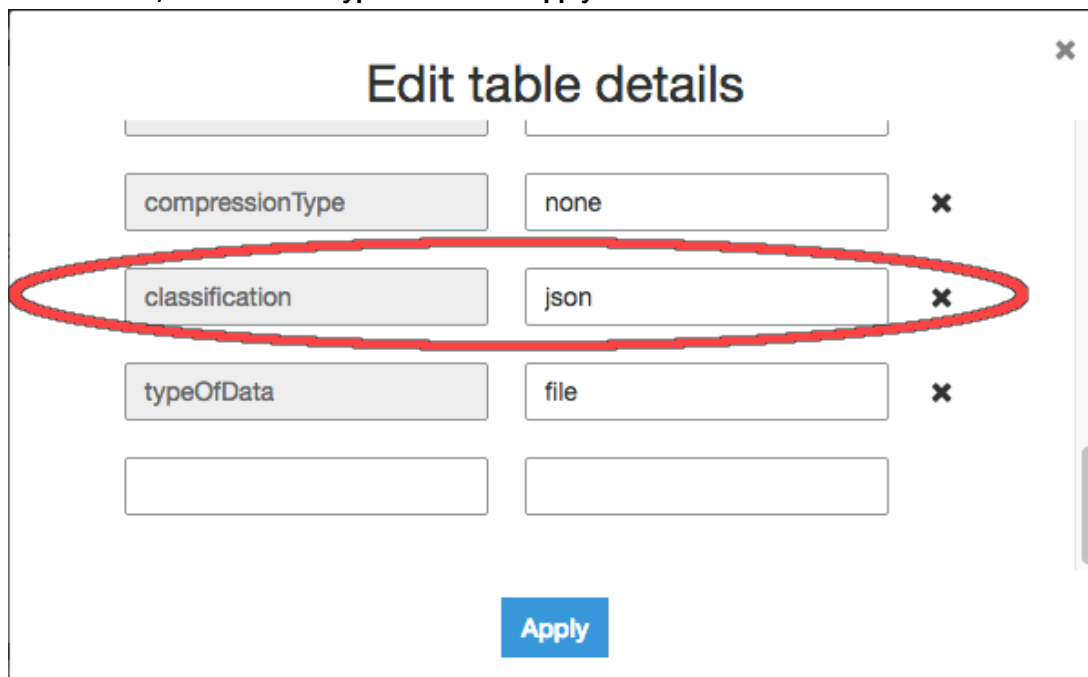
If the table property was not added when the table was created, you can add it using the AWS Glue console.

### To change the classification property using the console

#### 1. Choose **Edit Table**.



2. For Classification, select the file type and choose Apply.



The screenshot shows a modal window titled "Edit table details" with a close button (X) in the top right corner. Inside the modal, there are several input fields arranged in a grid. The first row has a label "compressionType" and a value "none", with a small "X" icon to its right. The second row has a label "classification" and a value "json", with a small "X" icon to its right. This second row is circled in red. The third row has a label "typeOfData" and a value "file", with a small "X" icon to its right. Below these fields are two empty input boxes. At the bottom center of the modal is a blue button labeled "Apply".

For more information, see [Working with Tables](#) in the *AWS Glue Developer Guide*.

## Using 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.

We recommend to use Parquet and ORC data formats. AWS Glue supports writing to both of these data formats, which can make it easier and faster for you to transform data to an optimal format for Athena. For more information about these formats and other ways to improve performance, see [Top Performance Tuning Tips for Amazon Athena](#).

## Converting SMALLINT and TINYINT Data Types to INT When Converting to ORC

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 using the wizard or writing a script for an ETL job.

## Automating 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 a suboptimal format for querying in Athena. For more information, see [Triggering AWS Glue Jobs](#) in the *AWS Glue Developer Guide*.

# Upgrading to the AWS Glue Data Catalog Step-by-Step

Currently, all regions that support Athena also support AWS Glue Data Catalog. Databases and tables are available to Athena using the AWS Glue Data Catalog and vice versa.

If you created databases and tables using Athena or Amazon Redshift Spectrum prior to a region's support for AWS Glue, you can upgrade Athena to use the AWS Glue Data Catalog.

If you are using the older Athena-managed data catalog, you see the option to upgrade at the top of the console. The metadata in the Athena-managed catalog isn't available in the AWS Glue Data Catalog or vice versa. While the catalogs exist side-by-side, creating tables or databases with the same names fails in either AWS Glue or Athena. This prevents name collisions when you do upgrade. For more information about the benefits of using the AWS Glue Data Catalog, see [FAQ: Upgrading to the AWS Glue Data Catalog \(p. 32\)](#).

A wizard in the Athena console can walk you through upgrading to the AWS Glue console. The upgrade takes just a few minutes, and you can pick up where you left off. For information about each upgrade step, see the topics in this section.

For information about working with data and tables in the AWS Glue Data Catalog, see the guidelines in [Best Practices When Using Athena with AWS Glue \(p. 20\)](#).

## Step 1 - Allow a User to Perform the Upgrade

By default, the action that allows a user to perform the upgrade is not allowed in any policy, including any managed policies. Because the AWS Glue Data Catalog is shared throughout an account, this extra failsafe prevents someone from accidentally migrating the catalog.

Before the upgrade can be performed, you need to attach a customer-managed IAM policy, with a policy statement that allows the upgrade action, to the user who performs the migration.

The following is an example policy statement.

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

## Step 2 - Update Customer-Managed/Inline Policies Associated with Athena Users

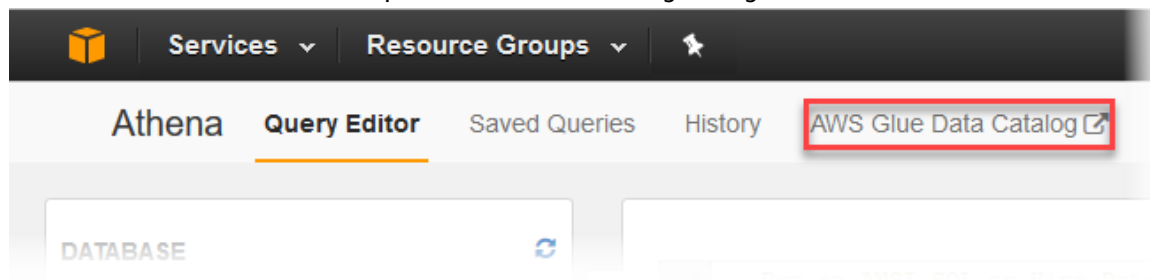
If you have customer-managed or inline IAM policies associated with Athena users, you need to update the policy or policies to allow actions that AWS Glue requires. If you use the Athena managed policy, no action is required. The AWS Glue policy actions to allow are listed in the example policy below. For the full policy statement, see [IAM Policies for User Access \(p. 240\)](#).

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateDatabase",
    "glue:DeleteDatabase",
    "glue:GetDatabase",
    "glue:GetDatabases",
    "glue:UpdateDatabase",
  ]
}
```

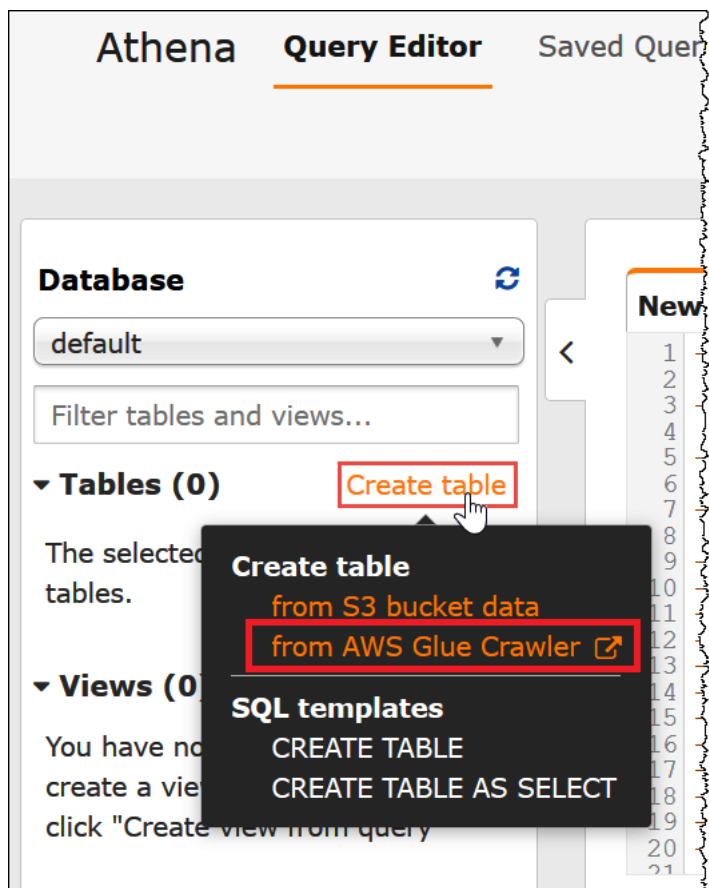
```
"glue:CreateTable",
"glue:DeleteTable",
"glue:BatchDeleteTable",
"glue:UpdateTable",
"glue:GetTable",
"glue:GetTables",
"glue:BatchCreatePartition",
"glue:CreatePartition",
"glue:DeletePartition",
"glue:BatchDeletePartition",
"glue:UpdatePartition",
"glue:GetPartition",
"glue:GetPartitions",
"glue:BatchGetPartition"
],
"Resource":[
  "*"
]
}
```

### Step 3 - Choose Upgrade in the Athena Console

After you make the required IAM policy updates, choose **Upgrade** in the Athena console. Athena moves your metadata to the AWS Glue Data Catalog. The upgrade takes only a few minutes. After you upgrade, the Athena console has a link to open the AWS Glue Catalog Manager from within Athena.



When you create a table using the console, you can create a table using an AWS Glue crawler. For more information, see [Using AWS Glue Crawlers \(p. 21\)](#).



## FAQ: Upgrading to the AWS Glue Data Catalog

If you created databases and tables using Athena in a region before AWS Glue was available in that region, metadata is stored in an Athena-managed data catalog, which only Athena and Amazon Redshift Spectrum can access. To use AWS Glue with Athena and Redshift Spectrum, you must upgrade to the AWS Glue Data Catalog.

### Why should I upgrade to the AWS Glue Data Catalog?

AWS Glue is a completely-managed extract, transform, and load (ETL) service. It has three main components:

- **An AWS Glue crawler** can automatically scan your data sources, identify data formats, and infer schema.
- **A fully managed ETL service** allows you to transform and move data to various destinations.
- **The AWS Glue Data Catalog** stores metadata information about databases and tables and points to a data store in Amazon S3 or a JDBC-compliant data store.

For more information, see [AWS Glue Concepts](#).

Upgrading to the AWS Glue Data Catalog has the following benefits.

## Unified metadata repository

The AWS Glue Data Catalog provides a unified metadata repository across a variety of data sources and data formats. It provides out-of-the-box integration with [Amazon Simple Storage Service \(Amazon S3\)](#), [Amazon Relational Database Service \(Amazon RDS\)](#), [Amazon Redshift](#), [Amazon Redshift Spectrum](#), [Athena](#), [Amazon EMR](#), and any application compatible with the Apache Hive metastore. You can create your table definitions one time and query across engines.

For more information, see [Populating the AWS Glue Data Catalog](#).

## Automatic schema and partition recognition

AWS Glue crawlers automatically crawl your data sources, identify data formats, and suggest schema and transformations. Crawlers can help automate table creation and automatic loading of partitions that you can query using Athena, Amazon EMR, and Redshift Spectrum. You can also create tables and partitions directly using the AWS Glue API, SDKs, and the AWS CLI.

For more information, see [Cataloging Tables with a Crawler](#).

## Easy-to-build pipelines

The AWS Glue ETL engine generates Python code that is entirely customizable, reusable, and portable. You can edit the code using your favorite IDE or notebook and share it with others using GitHub. After your ETL job is ready, you can schedule it to run on the fully managed, scale-out Spark infrastructure of AWS Glue. AWS Glue handles provisioning, configuration, and scaling of the resources required to run your ETL jobs, allowing you to tightly integrate ETL with your workflow.

For more information, see [Authoring AWS Glue Jobs](#) in the *AWS Glue Developer Guide*.

## Are there separate charges for AWS Glue?

Yes. With AWS Glue, you pay a monthly rate for storing and accessing the metadata stored in the AWS Glue Data Catalog, an hourly rate billed per second for AWS Glue ETL jobs and crawler runtime, and an hourly rate billed per second for each provisioned development endpoint. The AWS Glue Data Catalog allows you to store up to a million objects at no charge. If you store more than a million objects, you are charged USD\$1 for each 100,000 objects over a million. An object in the AWS Glue Data Catalog is a table, a partition, or a database. For more information, see [AWS Glue Pricing](#).

## Upgrade process FAQ

- [Who can perform the upgrade? \(p. 33\)](#)
- [My users use a managed policy with Athena and Redshift Spectrum. What steps do I need to take to upgrade? \(p. 34\)](#)
- [What happens if I don't upgrade? \(p. 34\)](#)
- [Why do I need to add AWS Glue policies to Athena users? \(p. 34\)](#)
- [What happens if I don't allow AWS Glue policies for Athena users? \(p. 34\)](#)
- [Is there risk of data loss during the upgrade? \(p. 34\)](#)
- [Is my data also moved during this upgrade? \(p. 34\)](#)

## Who can perform the upgrade?

You need to attach a customer-managed IAM policy with a policy statement that allows the upgrade action to the user who will perform the migration. This extra check prevents someone from accidentally

migrating the catalog for the entire account. For more information, see [Step 1 - Allow a User to Perform the Upgrade](#) (p. 30).

### My users use a managed policy with Athena and Redshift Spectrum. What steps do I need to take to upgrade?

The Athena managed policy has been automatically updated with new policy actions that allow Athena users to access AWS Glue. However, you still must explicitly allow the upgrade action for the user who performs the upgrade. To prevent accidental upgrade, the managed policy does not allow this action.

### What happens if I don't upgrade?

If you don't upgrade, you are not able to use AWS Glue features together with the databases and tables that you create in Athena or vice versa. You can use these services independently. During this time, Athena and AWS Glue both prevent you from creating databases or tables that have the same names in the other data catalog. This prevents name collisions when you do upgrade.

### Why do I need to add AWS Glue policies to Athena users?

Before you upgrade, Athena manages the data catalog, so Athena actions must be allowed for your users to perform queries. After you upgrade to the AWS Glue Data Catalog, AWS Glue actions must be allowed for your users. Remember, the managed policy for Athena has already been updated to allow the required AWS Glue actions, so no action is required if you use the managed policy.

### What happens if I don't allow AWS Glue policies for Athena users?

If you upgrade to the AWS Glue Data Catalog and don't update a user's customer-managed or inline IAM policies, Athena queries fail because the user won't be allowed to perform actions in AWS Glue. For the specific actions to allow, see [Step 2 - Update Customer-Managed/Inline Policies Associated with Athena Users](#) (p. 30).

### Is there risk of data loss during the upgrade?

No.

### Is my data also moved during this upgrade?

No. The migration only affects metadata.

## Using Athena Data Connector for 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

- [Overview of Features](#) (p. 35)
- [Workflow](#) (p. 35)

- [Considerations and Limitations \(p. 36\)](#)
- [Connecting Athena to an Apache Hive Metastore \(p. 37\)](#)
- [Using the AWS Serverless Application Repository to Deploy a Hive Data Source Connector \(p. 44\)](#)
- [Using a Default Catalog in External Hive Metastore Queries \(p. 46\)](#)
- [Using the AWS CLI with Hive Metastores \(p. 49\)](#)
- [Reference Implementation \(p. 55\)](#)

## 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 \(p. 56\)](#).
- **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](#).

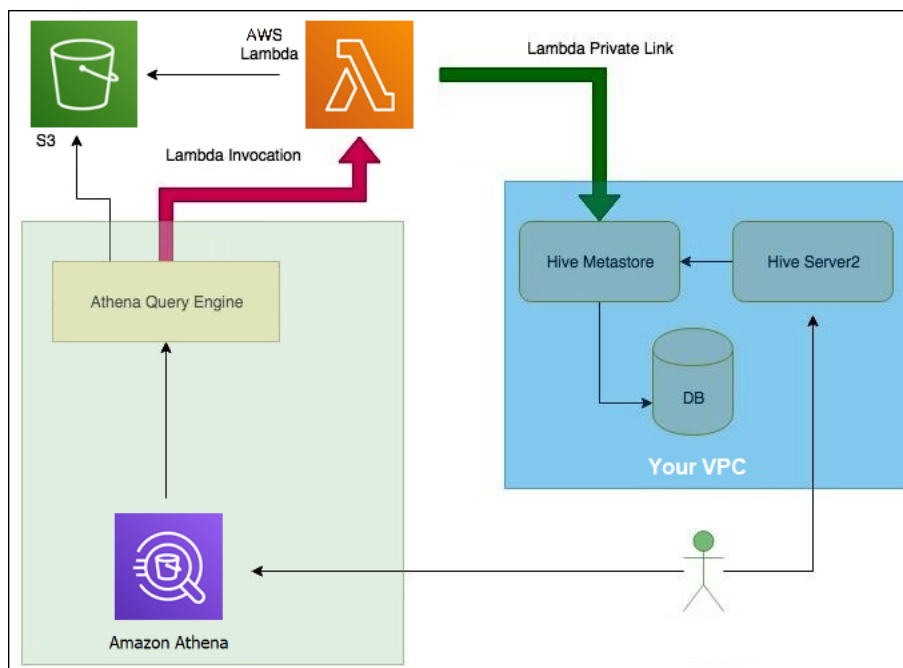
The reference implementation is available as the following two AWS SAM applications in the AWS Serverless Application Repository (SAR). You can use either of these applications in the SAR to create your own Lambda functions.

- **AthenaHiveMetastoreFunction** – Uber Lambda function .jar file.
- **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:

- DDL support for external Hive Metastore is limited to the following statements.
  - DESCRIBE TABLE
  - SHOW COLUMNS
  - SHOW TABLES
  - SHOW SCHEMAS
  - SHOW CREATE TABLE
  - SHOW TBLPROPERTIES
  - SHOW PARTITIONS
- The maximum number of registered catalogs that you can have is 1,000.
- You can use [CTAS \(p. 124\)](#) to create an AWS Glue table from a query on an external Hive metastore, but not to create a table on an external Hive metastore.

- You can use INSERT INTO to insert data into an AWS Glue table from a query on an external Hive metastore, but not to insert data into an external Hive metastore.
- Hive views are not compatible with Athena views and are not supported.
- Kerberos authentication for Hive metastore is not supported.

## 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 an Athena Data Connector for External Hive Metastore \(p. 254\)](#) and [Allow Lambda Function Access to External Hive Metastores \(p. 256\)](#).

- **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 \(p. 37\)](#) 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 [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#). For more information, see [Identity and Access Management in Athena \(p. 239\)](#).

## 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

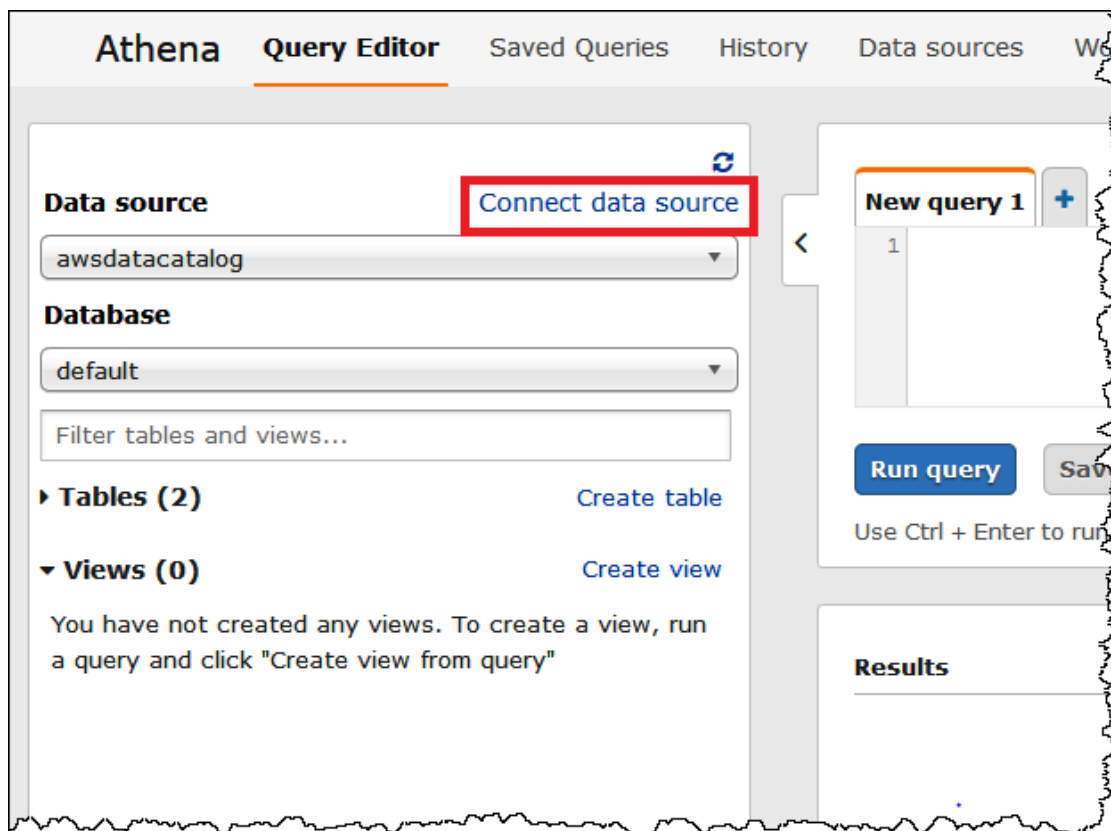
Athena does not remove the response files on Amazon S3. We recommend that you set up a retention policy to delete response files automatically.

## Connecting Athena to an Apache Hive Metastore

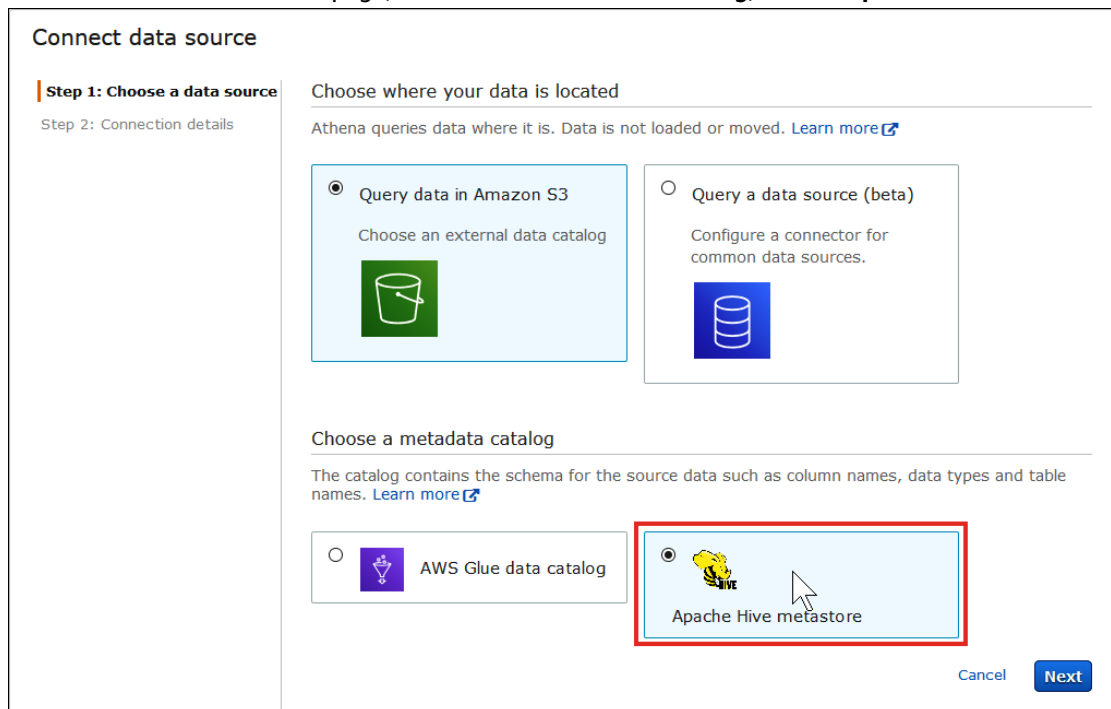
To connect Athena to an Apache Hive metastore, you must create and configure a Lambda function. For a basic implementation, you can perform all required steps starting from the Athena management console.

### To connect Athena to a Hive metastore

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **Connect data source**.



3. On the **Connect data source** page, for **Choose a metadata catalog**, choose **Apache Hive metastore**.



4. Choose **Next**.

5. On the **Connection details** page, for **Lambda function**, choose **Configure new AWS Lambda function**.

### Connect data source

[Step 1: Choose a data source](#)  
**Step 2: Connection details**

#### Connection details: Apache Hive metastore

Choose a Lambda function that is configured to connect to your data source, or create and configure a Lambda function to handle the connection. [Learn more](#)

**Lambda function** Choose or configure a new AWS Lambda function to connect to the data source.

Choose Lambda function

**Configure new AWS Lambda function**

**Catalog name** Create a unique name to specify this data source within a SQL statement.

Enter a unique name for the catalog

Use up to 127 characters and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, \_ (underscore), @ (at) and - (hyphen).

**Description**

Enter a description (optional)

Use up to 1024 characters.


[Cancel](#) [Previous](#) [Connect](#)

The **AthenaHiveMetastoreFunction** page opens in the AWS Lambda console.

Lambda > Functions > Create function > Review, configure and deploy

## AthenaHiveMetastoreFunction — version 1.0.1

Review, configure and deploy

 Copy as SAM Resource

### Application details

| Author         | Source code URL                                                                                                           | Description                                               | Report a vulnerability                                |
|----------------|---------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|-------------------------------------------------------|
| default author | <a href="https://github.com/aws-labs/aws-athena-hive-metastore">https://github.com/aws-labs/aws-athena-hive-metastore</a> | An Athena Lambda function to interact with Hive Metastore | If you believe this application poses a security risk |

### Readme file

Amazon Athena  
Hive Metastore  
Lambda Function

### Application settings

Application name  
The stack name of this application created via AWS CloudFormation

AthenaHiveMetastoreFunction

- 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 4MB.
  - HMSUri** – 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 128MB to 3008MB. 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.
- On the bottom right of the **Application details** page, select **I acknowledge that this app creates custom IAM roles**, and then choose **Deploy**.

☒ I acknowledge that this app creates custom IAM roles. [Info](#)

Cancel Previous 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.

[Lambda](#) > [Applications](#) > serverlessrepo-AthenaHiveMetastoreFunction

## serverlessrepo-AthenaHiveMetastoreFunction

Overview | Deployments | Monitoring

▼ **Getting started** Dismiss

Welcome to your new application view. From here, you can view the resources that make up your application, and monitor performance, errors, and traffic metrics. [Learn more](#)

**Resources** ↻

| Logical ID ▲                            | Physical ID      | Type ▼          | Last modified ▼ |
|-----------------------------------------|------------------|-----------------|-----------------|
| <a href="#">+ HiveMetastoreFunction</a> | MyLambdaFunction | Lambda Function | 45 seconds ago  |

- Return to the **Connection details** page of the **Data Sources** tab in the Athena console.
- Choose the **Refresh** icon next to **Choose Lambda function**. Refreshing the list of available functions causes your newly created function to appear in the list.

#### Connection details: Apache Hive metastore

Choose a Lambda function that is configured to connect to your data source, or create and configure a Lambda function to handle the connection. [Learn more](#)

**Lambda function** Choose or configure a new AWS Lambda function to connect to the data source.

Choose Lambda function



**Configure new AWS Lambda function**

**Catalog name** Create a unique name to specify this data source within a SQL statement.

Enter a unique name for the catalog

Use up to 127 characters and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, \_ (underscore), @ (at) and - (hyphen).

10. Now that your Lambda function is available, choose it.

**Lambda function** Choose or configure a new AWS Lambda function to connect to the data source.

Choose Lambda function



Choose Lambda function

Choose Lambda function

MyLambdaFunction



source within a SQL

Enter a unique name for the catalog

Use up to 127 characters and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, \_ (underscore), @ (at) and - (hyphen).

A new **Lambda function ARN** entry shows the ARN of your Lambda function.

### Connection details: Apache Hive metastore

Choose a Lambda function that is configured to connect to your data source, or create and configure a Lambda function to handle the connection. [Learn more](#)

**Lambda function**

Choose or configure a new AWS Lambda function to connect to the data source.

MyLambdaFunction

Configure new AWS Lambda function

**Lambda function ARN**

arn:aws:lambda:us-west-2::function:MyLambdaFunction

**Catalog name**

Create a unique name to specify this data source within a SQL statement.

hms-catalog-1

Use up to 127 characters and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, \_ (underscore), @ (at) and - (hyphen).

**Description**

HMS Catalog 1

Use up to 1024 characters.

Cancel

Previous

Connect

11. For **Catalog name**, enter a unique name that you will use in your SQL queries to reference the data source. The name can be up to 127 characters long and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-z, 0-9, \_ (underscore), @ (ampersand), and - (hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for custom catalog names.
12. (Optional) For **Description**, enter text that describes your data catalog.
13. Choose **Connect**. This connects Athena to your Hive metastore catalog.

The **Data sources** page shows a list of your connected catalogs, including the catalog that you just connected. All registered catalogs are visible to all users in the same AWS account.



| Data sources                                                                                                                                                                                                               |                                |                       |                                                                                                                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Data sources that Athena can connect to are listed below by their catalog names. You can connect Athena to multiple data sources and query data where it is. Athena does not load or move data. <a href="#">Learn more</a> |                                |                       |                                                                                                                                                         |
| <input type="text" value="Filter data sources"/>                                                                                                                                                                           |                                |                       |                                                                                                                                                         |
| <a href="#">Connect data source</a> <a href="#">View details</a> <a href="#">Edit</a> <a href="#">Delete</a>                                                                                                               |                                |                       |                                                                                                                                                         |
|                                                                                                                                                                                                                            | Catalog name                   | Catalog type          | Description                                                                                                                                             |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">awsdatacatalog</a> | AWS Glue data catalog | The AWS Glue Data Catalog stores metadata information about databases and tables, pointing to a data store in Amazon S3 or a JDBC-compliant data store. |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">catalog03</a>      | Hive metastore        | catalog 3                                                                                                                                               |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">ehms001</a>        | Hive metastore        | catalog for EHMS                                                                                                                                        |
| <input checked="" type="radio"/>                                                                                                                                                                                           | <a href="#">hms-catalog-1</a>  | Hive metastore        | Hive catalog                                                                                                                                            |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">ehms005</a>        | Hive metastore        | EHMS 005                                                                                                                                                |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">cmdb</a>           | Data source connector |                                                                                                                                                         |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">ddb</a>            | Data source connector |                                                                                                                                                         |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">ehms</a>           | Hive metastore        |                                                                                                                                                         |
| <input type="radio"/>                                                                                                                                                                                                      | <a href="#">ehmscatalog</a>    | Hive metastore        |                                                                                                                                                         |

14. You can now use the **Catalog name** that you specified to reference the Hive metastore in your SQL queries. 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;
```

15. To view, edit, or delete the data sources that you create, see [Managing Data Sources \(p. 69\)](#).

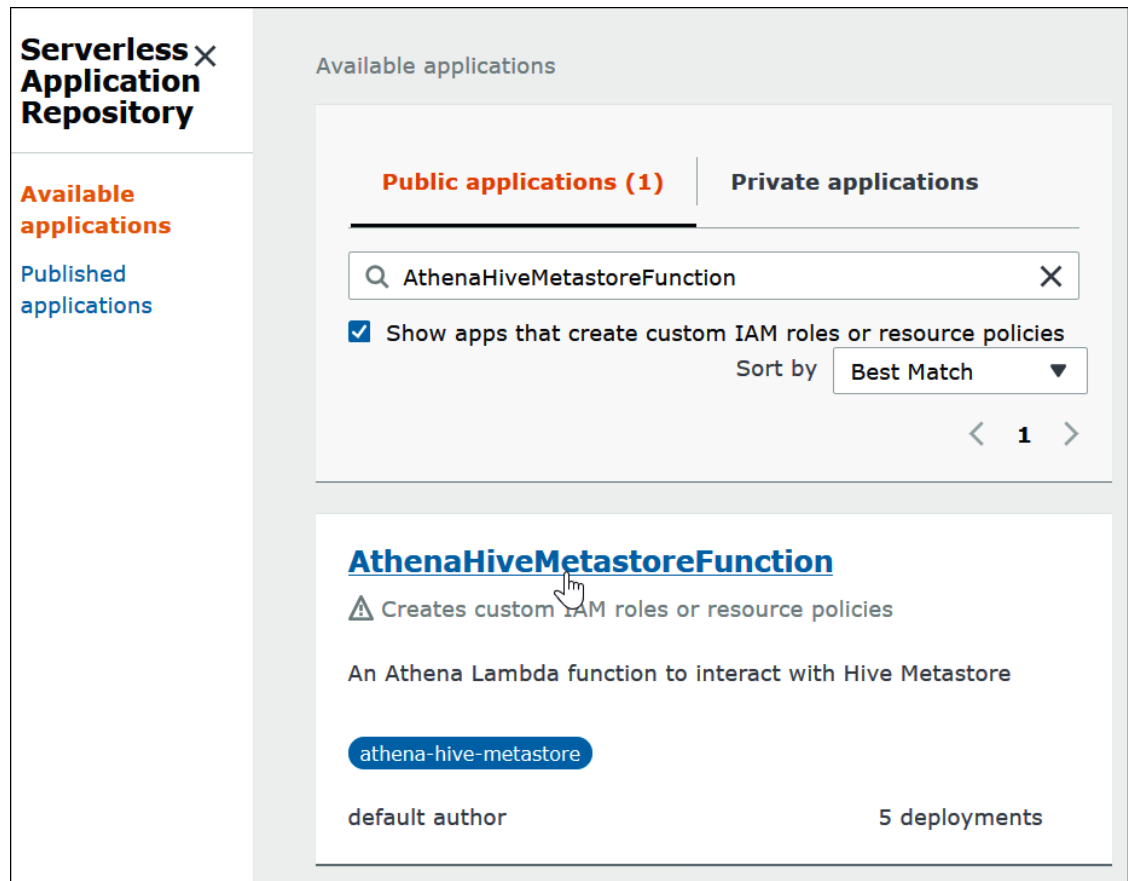
## Using the AWS Serverless Application Repository to Deploy a Hive Data Source Connector

You can also use the [AWS Serverless Application Repository](#) to deploy an Athena data source connector for Hive. Choose the connector that you want to use, provide the parameters that the connector requires, and then deploy the connector to your account.

### 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, type the name of one of the following connectors. 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.

- **AthenaHiveMetastoreFunction** – Uber Lambda function . jar file.
  - **AthenaHiveMetastoreFunctionWithLayer** – Lambda layer and thin Lambda function . jar file.
5. Choose the name of the connector.



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 4MB.
  - **HMSUri** – 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 128MB to 3008MB. 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 more information, see steps 8-15 of [Connecting Athena to an Apache Hive Metastore \(p. 37\)](#).

## Using a Default Catalog 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 catalog name before the database using the syntax `[[catalog_name].database_name].table_name`, as in the following example.

```
select * from "hms-catalog-1".hms_tpch.customer limit 10;
```

The screenshot shows the Amazon Athena query editor interface. At the top, there are tabs for 'customer-v3', 'New query 2', 'New query 3', and 'New query 4'. The active tab 'customer-v3' displays the query: `1 select * from "hms-catalog-1".hms_tpch.customer limit 10;`. Below the query editor, there are buttons for 'Run query' and 'Save as', followed by the execution details: '(Run time: 1.55 seconds, Data scanned: 3.56 MB)'. A note indicates: 'Use Ctrl + Enter to run query, Ctrl + Space to autocomplete'. Below this, there is a 'Results' section with a table showing the query results.

|   | c_custkey | c_name             | c_address                     | c_na |
|---|-----------|--------------------|-------------------------------|------|
| 1 | 1234384   | Customer#001234384 | M9A9gpYzW30QcwetlL8           | 4    |
| 2 | 1234385   | Customer#001234385 | G5EsJUSdFzBvfxzKULGJPZ6       | 14   |
| 3 | 1234386   | Customer#001234386 | oPrROIMm3sGmli                | 5    |
| 4 | 1234387   | Customer#001234387 | QHMeHmNMt,kJrY JeuW0EgE       | 7    |
| 5 | 1234388   | Customer#001234388 | okb0uAxgs0aUvSXoCooiKJ8D1HTBc | 8    |

2. When the catalog that you want to use is already selected as your data source, you can omit the catalog name from the query, as in the following example.

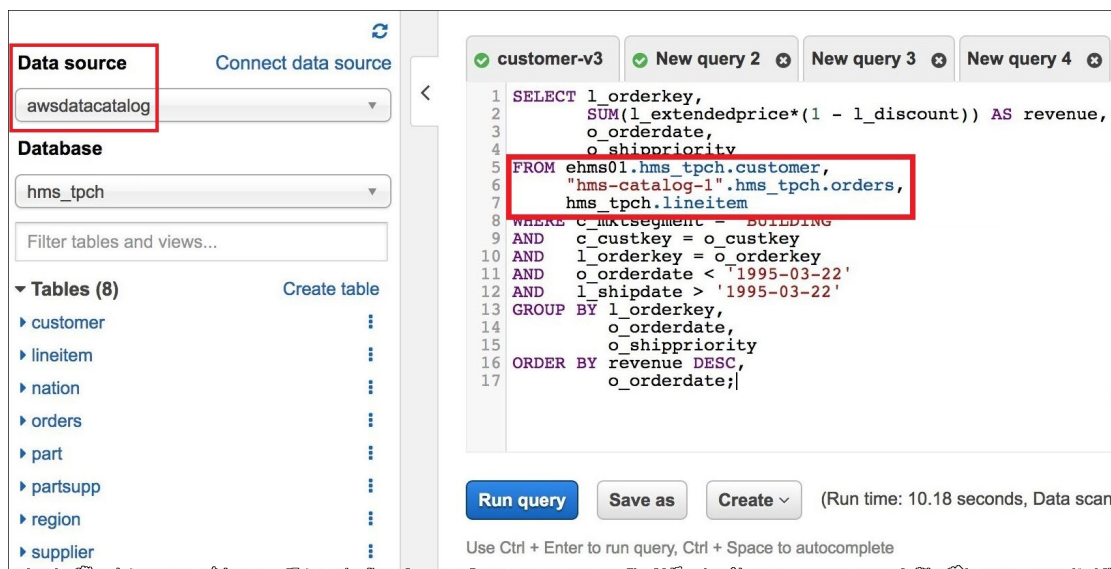
```
select * from hms_tpch.customer limit 10;
```

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Data source' dropdown is set to 'hms-catalog-1' and the 'Database' dropdown is set to 'hms\_tpch'. Below the database dropdown, there is a list of tables: customer, orders, lineitem, part, partsupp, supplier, nation, region, elb\_logs\_1, and elb\_logs\_2. The query editor on the right shows a SQL query: `select * from hms_tpch.customer limit 10;`. Below the query editor, there are buttons for 'Run query' and 'Save as', and a status bar indicating '(Run time: 1.45 seconds, Data scanned: ...)'. The results table shows 5 rows of customer data.

|   | c_custkey | c_name             | c_address            |
|---|-----------|--------------------|----------------------|
| 1 | 1234384   | Customer#001234384 | M9A9gpYzW30QcwetIL8  |
| 2 | 1234385   | Customer#001234385 | G5EsJUSdFzBvfxzKULG  |
| 3 | 1234386   | Customer#001234386 | oPrROIImm3sGmli      |
| 4 | 1234387   | Customer#001234387 | QHMeNmMt,kJrY JEuWC  |
| 5 | 1234388   | Customer#001234388 | okb0uAxs0aUvSXoCooik |

- For multiple catalogs, you can omit only the default catalog name. Specify the full name for any non-default catalogs. For example, the `FROM` statement in the following query omits the catalog name for AWS Glue catalog, but it fully qualifies the first two catalog names.

```
...
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, when you select the catalog and the database in the **Data source** panel, you can omit the catalog prefix from the query.

In the following example, the data source and database are selected in the query editor. The *show create table customer* statement succeeds when the `hms-catalog-1` prefix and the `hms_tpch` database name are omitted from the query.

The screenshot displays the Amazon Athena console interface. On the left, the 'Data source' dropdown is set to 'hms-catalog-1' and the 'Database' dropdown is set to 'hms\_tpch'. Below these, a search bar labeled 'Filter tables and views...' is present. A list of tables is shown, including 'customer', 'orders', 'lineitem', 'part', 'partsupp', 'supplier', 'nation', 'region', 'elb\_logs\_1', and 'elb\_logs\_2'. On the right, the 'New query 1' tab is active, showing a query editor with the text 'show create tabl'. Below the editor is a 'Run query' button and a 'Save as' button. The 'Results' section at the bottom right shows the SQL command 'CREATE EXTERNAL TABLE' with various column definitions.

**Data source** [Connect data source](#)

hms-catalog-1

**Database**

hms\_tpch

Filter tables and views...

▼ **Tables (10)**

- ▶ customer
- ▶ orders
- ▶ lineitem
- ▶ part
- ▶ partsupp
- ▶ supplier
- ▶ nation
- ▶ region
- ▶ elb\_logs\_1
- ▶ elb\_logs\_2

**New query 1** **New query**

1 show create tabl

**Run query** **Save as**

Use Ctrl + Enter to run query,

**Results**

```
CREATE EXTERNAL TABLE
  `c_custkey` int,
  `c_name` string,
  `c_address` string,
  `c_nationkey` int,
  `c_phone` string,
  `c_acctbal` double,
  `c_mktsegment` str
```

## Using 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://athena-data/",
      "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 sampledby --
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://athena-data/",
        "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://myjasondata/",
  "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":
"eyJzYXN0RXZhbHVhdGVkS2V5Ijp7IkhBU0hfS0VZIjp7InMiOiJ0LjYjYjNTQ1YmU4Y2I5OWE5NTg0MjFjYTYzIn0sI
}
```

## 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://mybucket/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://mybucket/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://mybucket/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://mybucket/lambda/results"
```

## Reference Implementation

Athena provides a reference implementation of its connector for external Hive metastore on GitHub.com at <https://github.com/aws-labs/aws-athena-hive-metastore>.

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-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-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, 4MB). 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.

## Building the Artifacts Yourself

Most use cases do not require you to modify the reference implementation. However, if necessary, you can modify the source code, 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-lambda-func`.

## Using Amazon Athena Federated Query (Preview)

If you have data in sources other than Amazon S3, you can use Athena Federated Query (Preview) 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 (Preview), 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 as 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.

For a list of available Athena data source connectors, see [Using Athena Data Source Connectors \(p. 59\)](#).

For information about writing your own data source connector, see [Example Athena Connector on GitHub](#).

## Considerations and Limitations

- **Available Regions** – The Athena federated query feature is available in preview in the US East (N. Virginia), Asia Pacific (Mumbai), Europe (Ireland), and US West (Oregon) Regions.

- **AmazonAthenaPreviewFunctionality workgroup** – To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups \(p. 331\)](#).
- **Views** – You cannot use views with federated data sources.

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 [Example IAM Permissions Policies to Allow Athena Federated Query \(Preview\) \(p. 260\)](#).

- **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.
- **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.

For the most up-to-date information about known issues and limitations, see [Limitations and Issues](#) in the `aws-athena-query-federation` GitHub repository.

## Deploying a Connector and Connecting to a Data Source

Preparing to create federated queries is a two-part process: deploying a Lambda function data source connector, and connecting the Lambda function to a data source. In the first part, you give the Lambda function a name that you can later choose in the Athena console. In the second part, you give the connector a name that you can reference in your SQL queries.

### Part 1: Deploying a Data Source Connector

To choose, name, and deploy a data source connector, you use the Athena and Lambda consoles in an integrated process.

#### Note

To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups \(p. 331\)](#).

#### To deploy a data source connector

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **Connect data source**.
3. On the **Connect data source** page, choose **Query a data source**.
4. For **Choose a data source**, choose the data source that you want to query with Athena, such as **Amazon CloudWatch Logs**.
5. Choose **Next**.
6. For **Lambda function**, choose **Configure new function**. The function page for the connector that you chose opens in the Lambda console. The page includes detailed information about the connector.
7. Under **Application settings**, enter the required information. At a minimum, this includes:

- **AthenaCatalogName** – A name 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.
8. Select **I acknowledge that this app creates custom IAM roles**. For more information, choose the **Info** link.
  9. Choose **Deploy**. The **Resources** section of the Lambda console shows the deployment status of the connector and informs you when the deployment is complete.

## Part 2: Connecting to a Data Source

After you deploy the data source connector to your account, you can connect it to a data source.

### To connect to a data source using a connector that you have deployed to your account

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose **Connect data source**.
3. Choose **Query a data source**.
4. Choose the data source for the connector that you just deployed, such as **Amazon CloudWatch Logs**. If you used the Athena Query Federation SDK to create your own connector and have deployed it to your account, choose **All other data sources**.
5. Choose **Next**.
6. For **Choose Lambda function**, choose the function that you named. The Lambda function's ARN is displayed.
7. For **Catalog name**, enter a unique name to use for the data source in your SQL queries, such as `cloudwatchlogs`. The name can be up to 127 characters and must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, \_(underscore), @(ampersand) and -(hyphen). The names `awsdatacatalog`, `hive`, `jmx`, and `system` are reserved by Athena and cannot be used for custom catalog names.
8. Choose **Connect**. The **Data sources** page now shows your connector in the list of catalog names. You can now use the connector in your queries.

For information about writing queries with data connectors, see [Writing Federated Queries](#) (p. 61).

## Using the AWS Serverless Application Repository to Deploy a Data Source Connector

You can also use the [AWS Serverless Application Repository](#) to deploy an Athena data source connector. You find the connector that you want to use, provide the parameters that the connector requires, and then deploy the connector to your account.

### Note

To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups](#) (p. 331).

### 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, or search for applications published with the author name **Amazon Athena Federation**. This author name is reserved for applications that the Amazon Athena team has written, tested, and validated. For a list of prebuilt Athena data connectors, see [Using Athena Data Source Connectors \(p. 59\)](#).
5. Choose the name of the connector. This 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**, **SpillBucket**, specify an Amazon S3 bucket to receive data from large response payloads. For information about the remaining configurable options, see the corresponding [Available Connectors](#) topic on GitHub.
7. At the bottom right of the **Application details** page, choose **Deploy**.

## Using Athena 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.

### Note

To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups \(p. 331\)](#).

See the following topics for more information:

- For information about deploying an Athena data source connector, see [Deploying a Connector and Connecting to a Data Source \(p. 57\)](#).
- For information about writing queries that use Athena data source connectors, see [Writing Federated Queries \(p. 61\)](#).
- For complete information about the Athena data source connectors, see [Available Connectors](#) on GitHub.

### Topics

- [Athena AWS CMDB Connector \(p. 59\)](#)
- [Amazon Athena CloudWatch Connector \(p. 60\)](#)
- [Amazon Athena CloudWatch Metrics Connector \(p. 60\)](#)
- [Amazon Athena DocumentDB Connector \(p. 60\)](#)
- [Amazon Athena DynamoDB Connector \(p. 60\)](#)
- [Amazon Athena Elasticsearch Connector \(p. 60\)](#)
- [Amazon Athena HBase Connector \(p. 60\)](#)
- [Amazon Athena Connector for JDBC-Compliant Data Sources \(p. 61\)](#)
- [Amazon Athena Redis Connector \(p. 61\)](#)
- [Amazon Athena TPC Benchmark DS \(TPC-DS\) Connector \(p. 61\)](#)

## Athena AWS CMDB Connector

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



For information about supported services, parameters, permissions, deployment, performance, and licensing, see [Amazon Athena AWS CMDB Connector](#) on GitHub.

## 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.

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.

For more information about configuration options, throttling control, table mapping schema, permissions, deployment, performance considerations, and licensing, see [Amazon Athena CloudWatch Connector](#) on GitHub.

## Amazon Athena CloudWatch Metrics Connector

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

For information about configuration options, table mapping, permissions, deployment, performance considerations, and licensing, see [Amazon Athena Cloudwatch Metrics Connector](#) on GitHub.

## Amazon Athena DocumentDB Connector

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

For information about how the connector generates schemas, configuration options, permissions, deployment, and performance considerations, see [Amazon Athena DocumentDB Connector](#) on GitHub.

## 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.

For information about configuration options, permissions, deployment, and performance considerations, see [Amazon Athena DynamoDB Connector](#) on GitHub.

## Amazon Athena Elasticsearch Connector

The Amazon Athena Elasticsearch connector enables Amazon Athena to communicate with your Elasticsearch instances so that you can use SQL to query your Elasticsearch data. This connector works with the Amazon Elasticsearch Service or any Elasticsearch-compatible endpoint that is configured with `Elasticsearch version 7.0` or higher.

For information about configuration options, databases and tables, data types, deployment, and performance considerations, see [Amazon Athena Elasticsearch Connector](#) on GitHub.

## Amazon Athena HBase Connector

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

For information about configuration options, data types, permissions, deployment, performance, and licensing, see [Amazon Athena HBase Connector](#) on GitHub.

## Amazon Athena Connector for JDBC-Compliant Data Sources

The Amazon Athena Lambda JDBC connector enables Amazon Athena to access your JDBC-compliant database. Currently supported databases include MySQL, PostgreSQL, and Amazon Redshift.

For information about supported databases, configuration parameters, supported data types, JDBC driver versions, limitations, and other information, see [Amazon Athena Lambda JDBC Connector](#) on GitHub.

## Amazon Athena Redis Connector

The Amazon Athena Redis connector enables Amazon Athena to communicate with your Redis instances so that you can query your Redis data with SQL. You can use the AWS Glue Data Catalog to map your Redis key-value pairs into virtual tables.

For information about configuration options, setting up databases and tables, data types, permissions, deployment, performance, and licensing, see [Amazon Athena Redis Connector](#) on GitHub.

## 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. The Athena TPC-DS connector generates a TPC-DS compliant database at one of four scale factors.

For information about configuration options, databases and tables, permissions, deployment, performance, and licensing, see [Amazon Athena TPC-DS Connector](#) on GitHub.

## Writing 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.

### Querying a Single Data Source

The examples in this section assume that you have configured and deployed the 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.

## Example

```
SELECT * FROM "MyCloudwatchCatalog"."/var/e-commerce-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.

## Example

```
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/e-commerce-engine/order-processor".all_log_streams
WHERE Regexp_extract(message, '(.):.*', 1) != 'INFO'
```

The following image shows a sample result.

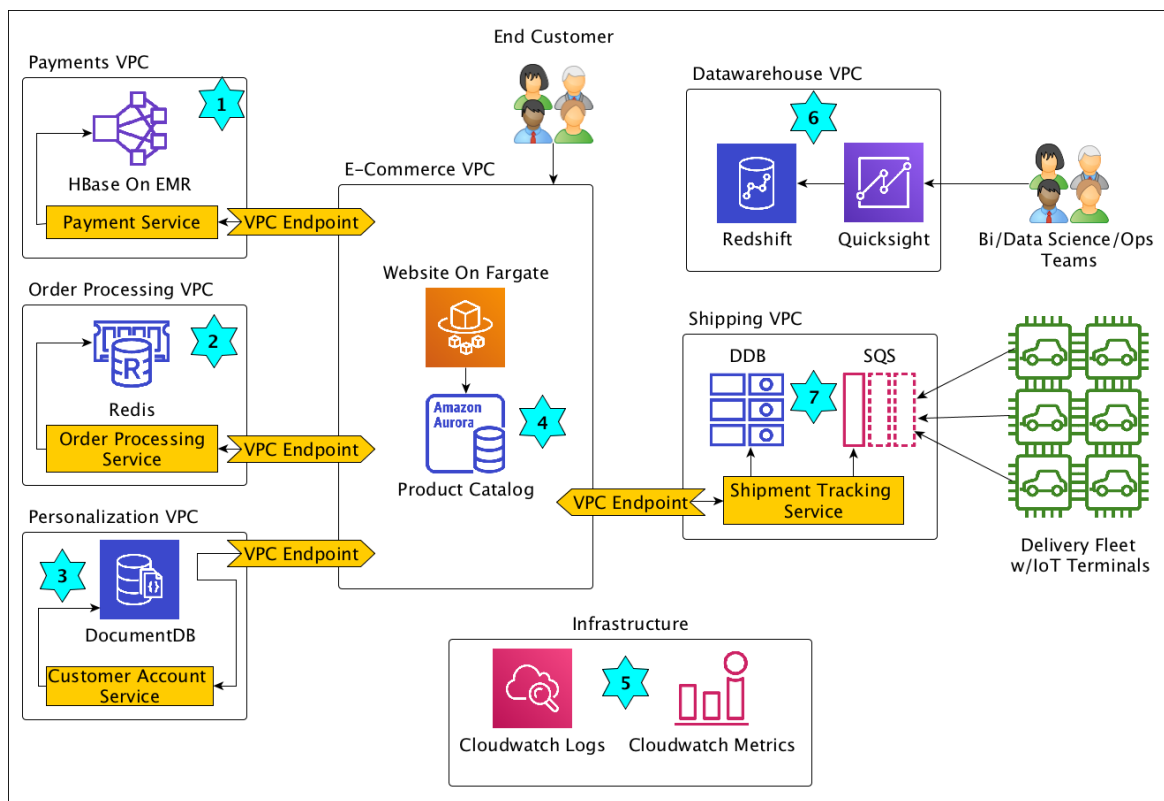
### Note

This example shows a query where the data source has been registered as a catalog with Athena. You can also reference a data source connector Lambda function using the format `lambda:MyLambdaFunctionName`.

| Results |                 |         |                                                               |           |
|---------|-----------------|---------|---------------------------------------------------------------|-----------|
|         | ec2_instance    | orderId | order_processor_log                                           | log_level |
| 1       | i-0a94127ec09dd | 0001235 | ERROR: orderId=0001235 encountered a problem during shipping. | ERROR     |
| 2       | i-0a94127ec09dd | 0002234 | WARN: orderId=0002234 encountered a problem during shipping.  | WARN      |

## Querying Multiple Data Sources

As a more complex example, imagine an ecommerce company that has an application infrastructure such as the one shown in the following diagram.



The following descriptions explain the numbered items in the diagram.

1. Payment processing in a secure VPC with transaction records stored in HBase on Amazon EMR
2. Redis to store active orders so that the processing engine can access them quickly
3. Amazon DocumentDB for customer account data such as email addresses and shipping addresses
4. A product catalog in Amazon Aurora for an ecommerce site that uses automatic scaling on Fargate
5. CloudWatch Logs to house the order processor's log events
6. A write-once-read-many data warehouse on Amazon RDS
7. DynamoDB to store shipment tracking data

Imagine that a data analyst for this ecommerce application discovers that the state of some orders is being reported erroneously. Some orders show as pending even though they were delivered, while others show as delivered but haven't shipped.

The analyst wants to know how many orders are being delayed and what the affected orders have in common across the ecommerce infrastructure. Instead of investigating the sources of information separately, the analyst federates the data sources and retrieves the necessary information in a single query. Extracting the data into a single location is not necessary.

The analyst's query uses the following Athena data connectors:

- **CloudWatch Logs** – Retrieves logs from the order processing service and uses regex matching and extraction to filter for orders with `WARN` or `ERROR` events.
- **Redis** – Retrieves the active orders from the Redis instance.
- **CMDB** – Retrieves the ID and state of the Amazon EC2 instance that ran the order processing service and logged the `WARN` or `ERROR` message.

- [DocumentDB](#) – Retrieves the customer email and address from Amazon DocumentDB for the affected orders.
- [DynamoDB](#) – Retrieves the shipping status and tracking details from the shipping table to identify possible discrepancies between reported and actual status.
- [HBase](#) – Retrieves the payment status for the affected orders from the payment processing service.

## Example

### Note

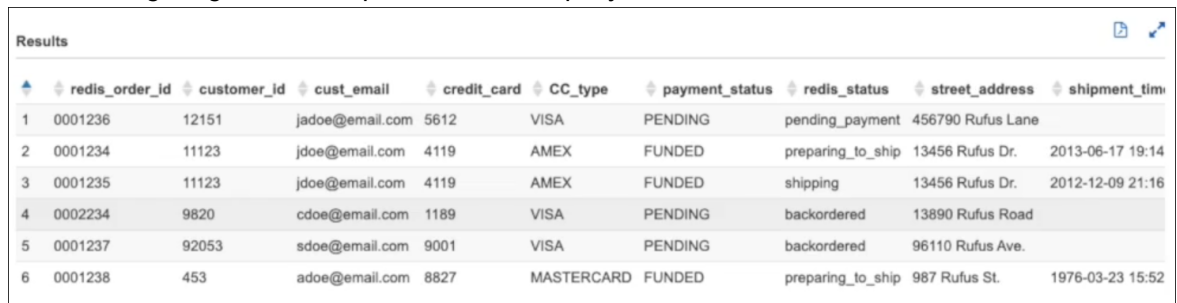
This example shows a query where the data source has been registered as a catalog with Athena. You can also reference a data source connector Lambda function using the format `lambda:MyLambdaFunctionName`.

```
--Sample query using multiple Athena data connectors.
WITH logs
  AS (SELECT log_stream,
             message
             AS
             order_processor_log,
             Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
             Regexp_extract(message, '(.):.*', 1) AS log_level
  FROM
    "MyCloudwatchCatalog"."/var/ecommerce-engine/order-processor".all_log_streams
  WHERE Regexp_extract(message, '(.):.*', 1) != 'INFO'),
active_orders
AS (SELECT *
  FROM redis.redis_db.redis_customer_orders),
order_processors
AS (SELECT instanceid,
           publicipaddress,
           state.NAME
  FROM awscmdb.ec2.ec2_instances),
customer
AS (SELECT id,
           email
  FROM docdb.customers.customer_info),
addresses
AS (SELECT id,
           is_residential,
           address.street AS street
  FROM docdb.customers.customer_addresses),
shipments
AS ( SELECT order_id,
           shipment_id,
           from_unixtime(cast(shipped_date as double)) as shipment_time,
           carrier
  FROM lambda_ddb.default.order_shipments),
payments
AS ( SELECT "summary:order_id",
           "summary:status",
           "summary:cc_id",
           "details:network"
  FROM "hbase".hbase_payments.transactions)

SELECT _key_ AS redis_order_id,
       customer_id,
       customer.email AS cust_email,
       "summary:cc_id" AS credit_card,
       "details:network" AS CC_type,
       "summary:status" AS payment_status,
       status AS redis_status,
       addresses.street AS street_address,
       shipments.shipment_time as shipment_time,
       shipments.carrier as shipment_carrier,
```

```
publicipaddress AS ec2_order_processor,  
NAME           AS ec2_state,  
log_level,  
order_processor_log  
FROM active_orders  
LEFT JOIN logs  
      ON logs.orderid = active_orders._key_  
LEFT JOIN order_processors  
      ON logs.log_stream = order_processors.instanceid  
LEFT JOIN customer  
      ON customer.id = customer_id  
LEFT JOIN addresses  
      ON addresses.id = address_id  
LEFT JOIN shipments  
      ON shipments.order_id = active_orders._key_  
LEFT JOIN payments  
      ON payments."summary:order_id" = active_orders._key_
```

The following image shows sample results of the query.



|   | redis_order_id | customer_id | cust_email      | credit_card | CC_type    | payment_status | redis_status      | street_address    | shipment_time    |
|---|----------------|-------------|-----------------|-------------|------------|----------------|-------------------|-------------------|------------------|
| 1 | 0001236        | 12151       | jadoe@email.com | 5612        | VISA       | PENDING        | pending_payment   | 456790 Rufus Lane |                  |
| 2 | 0001234        | 11123       | jdoe@email.com  | 4119        | AMEX       | FUNDED         | preparing_to_ship | 13456 Rufus Dr.   | 2013-06-17 19:14 |
| 3 | 0001235        | 11123       | jdoe@email.com  | 4119        | AMEX       | FUNDED         | shipping          | 13456 Rufus Dr.   | 2012-12-09 21:16 |
| 4 | 0002234        | 9820        | cdoe@email.com  | 1189        | VISA       | PENDING        | backordered       | 13890 Rufus Road  |                  |
| 5 | 0001237        | 92053       | sdoe@email.com  | 9001        | VISA       | PENDING        | backordered       | 96110 Rufus Ave.  |                  |
| 6 | 0001238        | 453         | adoe@email.com  | 8827        | MASTERCARD | FUNDED         | preparing_to_ship | 987 Rufus St.     | 1976-03-23 15:52 |

## Writing a Data Source Connector Using the Athena Query Federation SDK

To write your own [data source connectors](#) (p. 56), 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.

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.

### Note

To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups](#) (p. 331).

For information about how to download the SDK and detailed instructions for writing your own connector, see [Example Athena Connector](#) on GitHub.

## IAM Policies for Accessing Data Catalogs

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 \(p. 66\)](#).

### 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 *IAM User Guide*.
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 \(p. 66\)](#).
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 *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 \(p. 66\)](#).

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 [IAM Policies for Accessing Data Catalogs](#) (p. 65).

- [Example Policy for Full Access to All Data Catalogs](#) (p. 67)
- [Example Policy for Full Access to a Specified Data Catalog](#) (p. 67)
- [Example Policy for Querying a Specified Data Catalog](#) (p. 68)
- [Example Policy for Management Operations on a Specified Data Catalog](#) (p. 69)
- [Example Policy for Listing Data Catalogs](#) (p. 69)
- [Example Policy for Metadata Operations on Data Catalogs](#) (p. 69)

### 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:GetExecutionEngine",
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
        "athena:GetCatalogs",
        "athena:GetNamespaces",
        "athena:GetTables",
        "athena:GetTable"
      ],
      "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`.

```
{  
  "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:

```
{  
  "Effect": "Allow",  
  "Action": [  
    "athena:ListDataCatalogs"  
  ],  
  "Resource": "*"   
}
```

### Example Example Policy for Metadata Operations on Data Catalogs

The following policy allows metadata operations on data catalogs:

```
{  
  "Effect": "Allow",  
  "Action": [  
    "athena:GetDatabase",  
    "athena:GetTableMetadata",  
    "athena:ListDatabases",  
    "athena:ListTableMetadata"  
  ],  
  "Resource": "*"   
}
```

## Managing Data Sources

You can use the **Data Sources** page of the Athena console to view, edit, or delete the data sources that you create, including Athena data source connector, AWS Glue Data Catalog, and Hive metastore catalog types.

### To view a data source

- Do one of the following:

- Choose the catalog name of the data source.
- Select the button next to the catalog name, and then choose **View details**.

The details page includes options to **Edit** or **Delete** the data source.

### To edit a data source

1. Do one of the following:
  - Select the button next to the catalog name, and then choose **Edit**.
  - Choose the catalog name of the data source, and then choose **Edit**.

**Data sources**

Data sources that Athena can connect to are listed below by their catalog name and query data where it is. Athena does not load or move data. [Learn more](#)

*Filter data sources*

**Connect data source** **View details** **Edit** **Delete**

|                                  | <b>Catalog name</b>              | <b>Catalog type</b>   |
|----------------------------------|----------------------------------|-----------------------|
| <input type="radio"/>            | <a href="#">awsdatacatalog</a>   | AWS Glue data catalog |
| <input checked="" type="radio"/> | <a href="#">hms-catalog-s301</a> | Hive metastore        |

Athena Query Editor Saved Queries History **Data sources** Workgroup : AmazonAthe...

[Data sources](#) > hms-catalog-s301

Data source: hms-catalog-s301

**Edit** **Delete**

|                             |                                                                                          |
|-----------------------------|------------------------------------------------------------------------------------------|
| <b>Catalog name</b>         | hms-catalog-s301                                                                         |
| <b>Catalog type</b>         | Hive metastore                                                                           |
| <b>Description</b>          | test catalog                                                                             |
| <b>Lambda function name</b> | external-hms-service-new                                                                 |
| <b>Lambda function ARN</b>  | arn:aws:lambda:us-east-1: [redacted]:function:external-hms-service-new <a href="#">↗</a> |

2. On the **Edit** page for the metastore, you can choose a different Lambda function for the data source or change the description of the existing function. When you edit an AWS Glue catalog, the AWS Glue console opens the corresponding catalog for editing.

Athena Query Editor Saved Queries History **Data sources** Workgroup : AmazonAthe...

[Data sources](#) > hms-catalog-s301

Edit hms-catalog-s301

**Lambda function** external-hms-service-new [↗](#)

**Lambda function ARN** arn:aws:lambda:us-east-1: [redacted]:function:external-hms-service-new [↗](#)

**Catalog name** hms-catalog-s301

**Description** test catalog

Use up to 1024 characters

[Cancel](#) [Save](#)

3. Choose **Save**.

### To delete a data source

1. Select the button next to the data source or the name of the data source, and then choose **Delete**. You are warned that when you delete a metastore data source, its corresponding Data Catalog, tables, and views are removed from the query editor. Saved queries that used the metastore no longer run in Athena.
2. Choose **Delete**.

## Connecting 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

- [Using Athena with the JDBC Driver \(p. 72\)](#)
- [Connecting to Amazon Athena with ODBC \(p. 73\)](#)

## Using Athena with the JDBC Driver

You can use a JDBC connection to connect Athena to business intelligence tools and other applications, such as [SQL Workbench](#). To do this, download, install, and configure the Athena JDBC driver, using the following links on Amazon S3.

### Links for Downloading the JDBC Driver

The JDBC driver version 2.0.14 complies with the JDBC API 4.1 and 4.2 data standards. Before downloading the driver, check which version of Java Runtime Environment (JRE) you use. The JRE version depends on the version of the JDBC API you are using with the driver. If you are not sure, download the latest version of the driver.

Download the driver that matches your version of the JDK and the JDBC data standards:

- The [AthenaJDBC41\\_2.0.14.jar](#) is compatible with JDBC 4.1 and requires JDK 7.0 or later.
- The [AthenaJDBC42\\_2.0.14.jar](#) is compatible with JDBC 4.2 and requires JDK 8.0 or later.

### JDBC 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 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.

## Migration from Previous Version of the JDBC Driver

The current JDBC driver version 2.0.14 is a drop-in replacement of the previous version of the JDBC driver version 2.0.9, and is backwards compatible with the JDBC driver version 2.0.9, with the following step that you must perform to ensure the driver runs.

### Important

To use JDBC driver version 2.0.5 or later, attach a permissions policy to IAM principals using the JDBC driver that allows the `athena:GetQueryResultsStream` policy action. This policy action is not exposed directly with the API. It is only used with the JDBC driver as part of streaming results support. For an example policy, see [AWSQuicksightAthenaAccess Managed Policy](#) (p. 242).

Additionally, ensure that port 444, which Athena uses to stream query results, is 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.`

For more information about upgrading to versions 2.0.5 or later from version 2.0.2, see the [JDBC Driver Migration Guide](#).

For more information about the previous versions of the JDBC driver, see [Using the Previous Version of the JDBC Driver](#) (p. 433).

If you are migrating from a 1.x driver to a 2.x driver, you must migrate your existing configurations to the new configuration. We highly recommend that you migrate to driver version 2.x. For information, see the [JDBC Driver Migration Guide](#).

## Connecting to Amazon Athena with ODBC

Download the Amazon Athena ODBC driver License Agreement, ODBC drivers, and ODBC documentation using the following links.

### Amazon Athena ODBC Driver License Agreement

[License Agreement](#)

### ODBC Driver Download Links

#### Windows

| Driver Version                | Download Link                                    |
|-------------------------------|--------------------------------------------------|
| ODBC 1.1.3 for Windows 32-bit | <a href="#">Windows 32 bit ODBC Driver 1.1.3</a> |
| ODBC 1.1.3 for Windows 64-bit | <a href="#">Windows 64 bit ODBC Driver 1.1.3</a> |

#### Linux

| Driver Version              | Download Link                                  |
|-----------------------------|------------------------------------------------|
| ODBC 1.1.3 for Linux 32-bit | <a href="#">Linux 32 bit ODBC Driver 1.1.3</a> |

| Driver Version              | Download Link                                  |
|-----------------------------|------------------------------------------------|
| ODBC 1.1.3 for Linux 64-bit | <a href="#">Linux 64 bit ODBC Driver 1.1.3</a> |

## OSX

| Driver Version     | Download Link                         |
|--------------------|---------------------------------------|
| ODBC 1.1.3 for OSX | <a href="#">OSX ODBC Driver 1.1.3</a> |

## Documentation

| Driver Version               | Download Link                                                                  |
|------------------------------|--------------------------------------------------------------------------------|
| Documentation for ODBC 1.1.3 | <a href="#">ODBC Driver Installation and Configuration Guide version 1.1.3</a> |
| Release Notes for ODBC 1.1.3 | <a href="#">ODBC Driver Release Notes version 1.1.3</a>                        |

## Migration from the Previous Version of the ODBC Driver

The current ODBC driver version 1.1.3 is a drop-in replacement of the previous version of the ODBC driver version 1.0.5. It is also backward compatible with the ODBC driver version 1.0.3, if you use the following required steps to make sure that the driver runs.

### Important

To use the ODBC driver versions 1.0.3 and greater, follow these requirements:

- 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.
- Add the `athena:GetQueryResultsStream` policy action to the list of policies for Athena. This policy action is not exposed directly with the API operation, and is used only with the ODBC and JDBC drivers, as part of streaming results support. For an example policy, see [AWSQuicksightAthenaAccess Managed Policy \(p. 242\)](#).

## Previous Versions of the ODBC Driver

| Driver Version 1.0.5          | Download Link                                                                  |
|-------------------------------|--------------------------------------------------------------------------------|
| ODBC 1.0.5 for Windows 32-bit | <a href="#">Windows 32 bit ODBC Driver 1.0.5</a>                               |
| ODBC 1.0.5 for Windows 64-bit | <a href="#">Windows 64 bit ODBC Driver 1.0.5</a>                               |
| ODBC 1.0.5 for Linux 32-bit   | <a href="#">Linux 32 bit ODBC Driver 1.0.5</a>                                 |
| ODBC 1.0.5 for Linux 64-bit   | <a href="#">Linux 64 bit ODBC Driver 1.0.5</a>                                 |
| ODBC 1.0.5 for OSX            | <a href="#">OSX ODBC Driver 1.0.5</a>                                          |
| Documentation for ODBC 1.0.5  | <a href="#">ODBC Driver Installation and Configuration Guide version 1.0.5</a> |

| Driver Version 1.0.4          | Download Link                                                                  |
|-------------------------------|--------------------------------------------------------------------------------|
| ODBC 1.0.4 for Windows 32-bit | <a href="#">Windows 32 bit ODBC Driver 1.0.4</a>                               |
| ODBC 1.0.4 for Windows 64-bit | <a href="#">Windows 64 bit ODBC Driver 1.0.4</a>                               |
| ODBC 1.0.4 for Linux 32-bit   | <a href="#">Linux 32 bit ODBC Driver 1.0.4</a>                                 |
| ODBC 1.0.4 for Linux 64-bit   | <a href="#">Linux 64 bit ODBC Driver 1.0.4</a>                                 |
| ODBC 1.0.4 for OSX            | <a href="#">OSX ODBC Driver 1.0.4</a>                                          |
| Documentation for ODBC 1.0.4  | <a href="#">ODBC Driver Installation and Configuration Guide version 1.0.4</a> |

| Driver Version 1.0.3          | Download Link                                                                  |
|-------------------------------|--------------------------------------------------------------------------------|
| ODBC 1.0.3 for Windows 32-bit | <a href="#">Windows 32-bit ODBC Driver 1.0.3</a>                               |
| ODBC 1.0.3 for Windows 64-bit | <a href="#">Windows 64-bit ODBC Driver 1.0.3</a>                               |
| ODBC 1.0.3 for Linux 32-bit   | <a href="#">Linux 32-bit ODBC Driver 1.0.3</a>                                 |
| ODBC 1.0.3 for Linux 64-bit   | <a href="#">Linux 64-bit ODBC Driver 1.0.3</a>                                 |
| ODBC 1.0.3 for OSX            | <a href="#">OSX ODBC Driver 1.0</a>                                            |
| Documentation for ODBC 1.0.3  | <a href="#">ODBC Driver Installation and Configuration Guide version 1.0.3</a> |

| Driver Version 1.0.2          | Download Link                                                                  |
|-------------------------------|--------------------------------------------------------------------------------|
| ODBC 1.0.2 for Windows 32-bit | <a href="#">Windows 32-bit ODBC Driver 1.0.2</a>                               |
| ODBC 1.0.2 for Windows 64-bit | <a href="#">Windows 64-bit ODBC Driver 1.0.2</a>                               |
| ODBC 1.0.2 for Linux 32-bit   | <a href="#">Linux 32-bit ODBC Driver 1.0.2</a>                                 |
| ODBC 1.0.2 for Linux 64-bit   | <a href="#">Linux 64-bit ODBC Driver 1.0.2</a>                                 |
| ODBC 1.0 for OSX              | <a href="#">OSX ODBC Driver 1.0</a>                                            |
| Documentation for ODBC 1.0.2  | <a href="#">ODBC Driver Installation and Configuration Guide version 1.0.2</a> |



# Creating 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

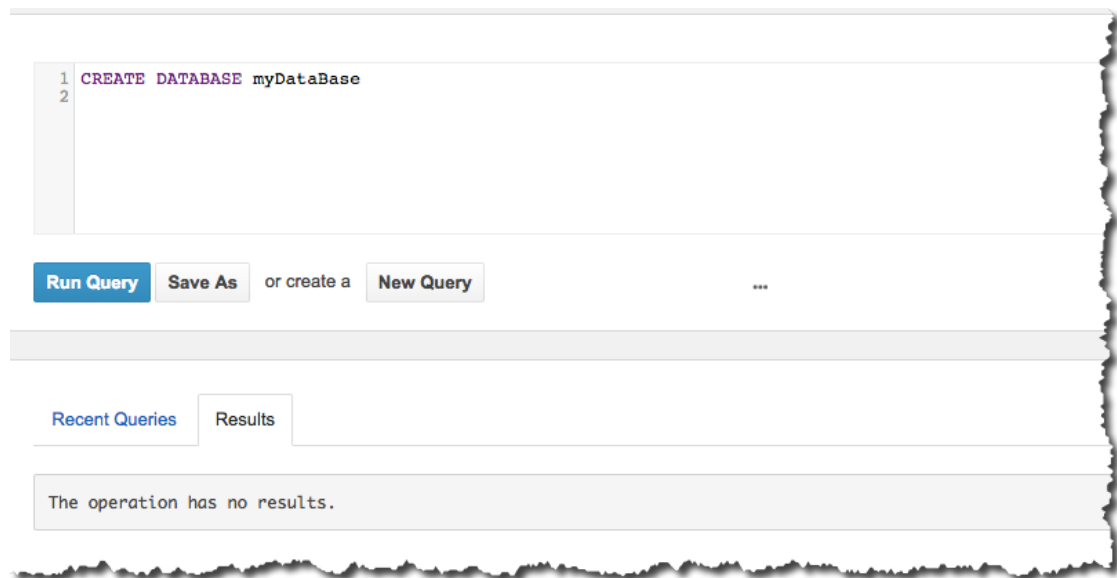
- [Creating Databases in Athena](#) (p. 76)
- [Creating Tables in Athena](#) (p. 78)
- [Names for Tables, Databases, and Columns](#) (p. 84)
- [Reserved Keywords](#) (p. 85)
- [Table Location in Amazon S3](#) (p. 86)
- [Columnar Storage Formats](#) (p. 88)
- [Converting to Columnar Formats](#) (p. 88)
- [Partitioning Data](#) (p. 92)
- [Partition Projection with Amazon Athena](#) (p. 96)

## Creating Databases in Athena

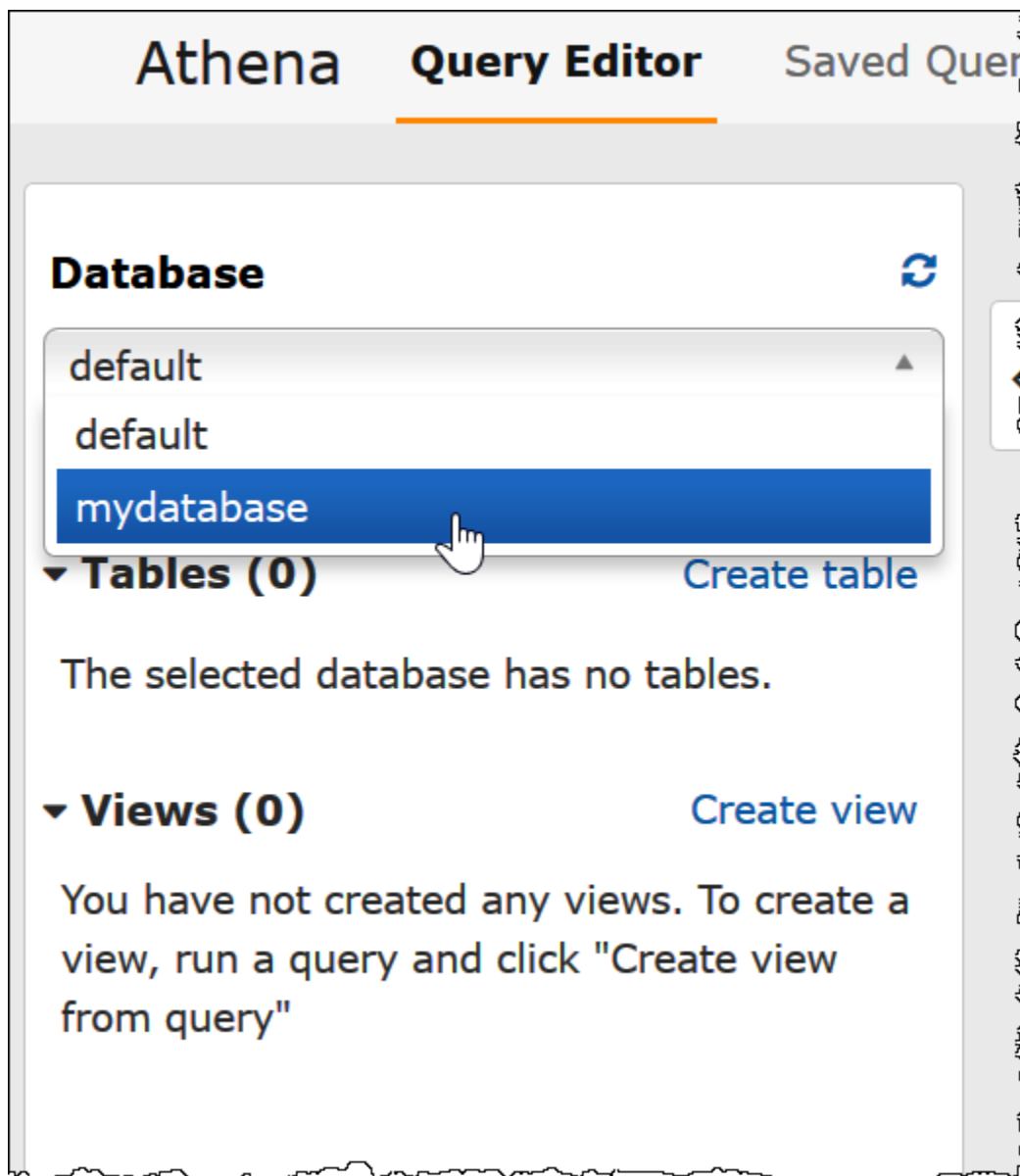
A database in Athena is a logical grouping for tables you create in it. Creating a database in the Athena console Query Editor is straightforward.

### To create a database using the Athena Query Editor

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. On the **Query Editor** tab, enter the Hive data definition language (DDL) command `CREATE DATABASE myDataBase`. Replace *myDatabase* with the name of the database that you want to create.



3. Choose **Run Query** or press **Ctrl+ENTER**.
4. To make your database the current database, select it from from the **Database** menu.



## Creating Tables in Athena

You can run DDL statements in the Athena console, using a JDBC or an ODBC driver, or using the Athena [Add table wizard \(p. 81\)](#).

When you create a new table schema in Athena, Athena stores the schema in a data catalog and uses it when you run queries.

Athena uses an approach known as *schema-on-read*, which means a schema is projected on to your data at the time you run a query. This eliminates the need for data loading or transformation.

Athena does not modify your data in Amazon S3.

Athena uses Apache Hive to define tables and create databases, which are essentially a logical namespace of tables.

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. 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.

When you query, you query the table using standard SQL and the data is read at that time. You can find guidance for how to create databases and tables using [Apache Hive documentation](#), but the following provides guidance specifically for Athena.

The maximum query string length is 256 KB.

Hive supports multiple data formats through the use of serializer-deserializer (SerDe) libraries. You can also define complex schemas using regular expressions. For a list of supported SerDe libraries, see [Supported SerDes and Data Formats \(p. 363\)](#).

## Considerations and Limitations

Following are some important limitations and considerations for tables in Athena.

### Requirements for Tables in Athena and Data in Amazon S3

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 the appropriate permissions to work with data in the Amazon S3 location. For more information, see [Access to Amazon S3 \(p. 243\)](#).
- 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 [Creating a Workgroup \(p. 332\)](#).
- Athena does not support querying the data in the `GLACIER` storage class. It ignores objects transitioned to the `GLACIER` storage class based on an Amazon S3 lifecycle policy.

For more information, 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 Developer 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](#).

## Functions Supported

The functions supported in Athena queries are those found within Presto. For more information, see [Presto 0.172 Functions and Operators](#) in the Presto documentation.

## Transactional Data Transformations Are 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 \(p. 400\)](#).

## Operations That Change Table States 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.

## All Tables Are EXTERNAL

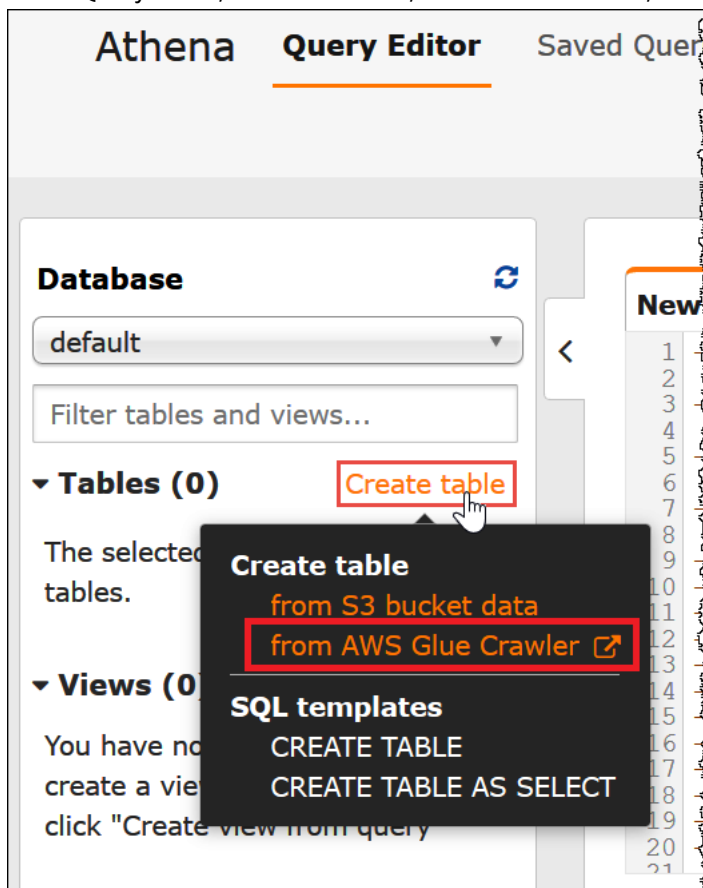
If you use `CREATE TABLE` without the `EXTERNAL` keyword, Athena issues an error; only tables with the `EXTERNAL` keyword can be created. We recommend that you always use the `EXTERNAL` keyword. When you drop a table in Athena, only the table metadata is removed; the data remains in Amazon S3.

## Creating Tables Using AWS Glue or the Athena Console

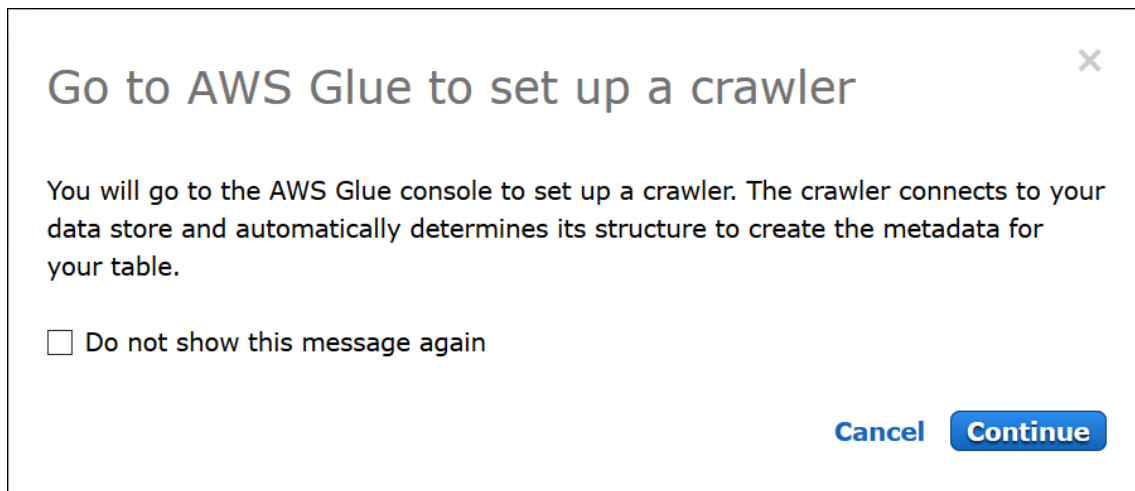
You can create tables in Athena by using AWS Glue, the add table wizard, or by running a DDL statement in the Athena Query Editor.

### To create a table using the AWS Glue Data Catalog

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. In the Query Editor, under **Database**, choose **Create table**, and then choose **from AWS Glue crawler**.



3. In the **Go to AWS Glue to set up a crawler** dialog box, choose **Continue**.

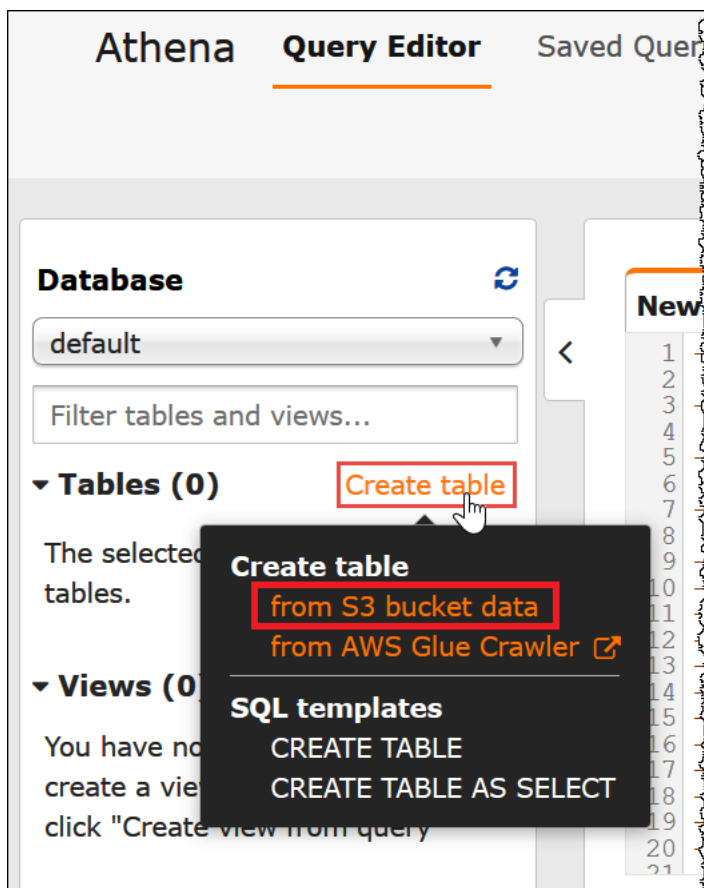


4. Follow the steps in the AWS Glue console to add a crawler.

For more information, see [Using AWS Glue Crawlers \(p. 21\)](#).

## To create a table using the Athena add table wizard

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Under the database display in the Query Editor, choose **Create table**, and then choose **from S3 bucket data**.



3. in the **Add table** wizard, follow the steps to create your table.

**Databases** > **Add table**

**Step 1: Name & Location**   Step 2: Data Format   Step 3: Columns   Step 4: Partitions

Database

Choose an existing database or create a new one by selecting "Create new database".

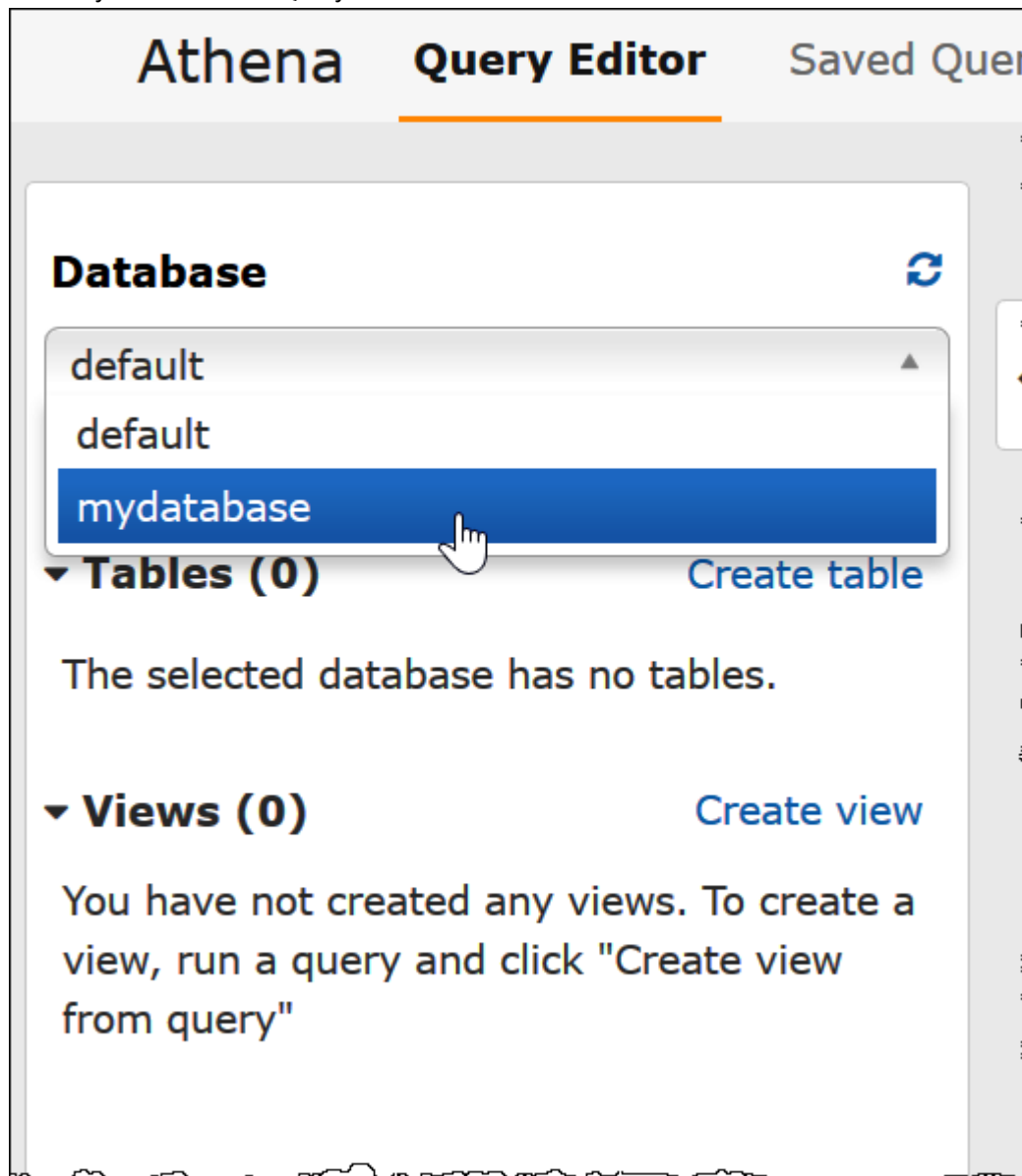
Name of the new database

Table Name

Name of the new table. Table names must be globally unique. Table names tend to correspond to the directory where the data will be stored.

## To create a table using Hive DDL

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. Enter a statement like the following, and then choose **Run Query**, or press **Ctrl+ENTER**.

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



After the table is created, you can run queries against your data.

## 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 \(p. 119\)](#), enclose them in double quotes (").

For more information, see [Reserved Keywords \(p. 85\)](#).

## Reserved Keywords

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 \(p. 119\)](#), enclose them in double quotes (").

- [List of Reserved Keywords in DDL Statements \(p. 85\)](#)
- [List of Reserved Keywords in SQL `SELECT` Statements \(p. 85\)](#)
- [Examples of Queries with Reserved Keywords \(p. 86\)](#)

## List of Reserved Keywords 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, 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
```

## List of Reserved Keywords 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_DATE, CURRENT_PATH,
```

```
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, LAST, LEFT, LIKE, LOCALTIME, LOCALTIMESTAMP, NATURAL,  
NORMALIZE, NOT, NULL, ON, OR, ORDER, OUTER, PREPARE,  
RECURSIVE, RIGHT, ROLLUP, SELECT, TABLE, THEN, TRUE,  
UNESCAPE, 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://test_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;
```

## Table Location in Amazon S3

When you run a `CREATE TABLE` query in Athena, you register your table with the AWS Glue Data Catalog. (If you are using Athena's older internal catalog, we highly recommend that you [upgrade \(p. 29\)](#) to the AWS Glue Data Catalog.)

To specify the path to your data in Amazon S3, use the `LOCATION` property, as shown in the following example:

```
CREATE EXTERNAL TABLE `test_table`(  
...  
)  
ROW FORMAT ...  
STORED AS INPUTFORMAT ...  
OUTPUTFORMAT ...  
LOCATION s3://bucketname/folder/
```

- For information about naming buckets, see [Bucket Restrictions and Limitations](#) in the *Amazon Simple Storage Service Developer Guide*.
- For information about using folders in Amazon S3, see [Using Folders](#) in the *Amazon Simple Storage Service Console User Guide*.

The `LOCATION` in Amazon S3 specifies *all* of the files representing your table.

### Important

Athena reads *all* data stored in `s3://bucketname/folder/`. 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 you want Athena to read. If you are leveraging partitioning, to ensure Athena scans data within a partition, your `WHERE` filter must include the partition. For more information, see [Table Location and Partitions](#) (p. 87).

When you specify the `LOCATION` in the `CREATE TABLE` statement, use the following guidelines:

- Use a trailing slash.

#### Use:

```
s3://bucketname/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.amazonaws.com` to the Amazon S3 bucket path.
  - Do not specify an Amazon S3 [access point](#) in the `LOCATION` clause. The table location can only be specified as a URI.
  - Do not use empty folders like `//` in the path, as follows: `s3://bucketname/folder//folder/`. While this is a valid Amazon S3 path, Athena does not allow it and changes it to `s3://bucketname/folder/folder/`, removing the extra `/`.

#### Do not use:

```
s3://path_to_bucket  
s3://path_to_bucket/*  
s3://path_to_bucket/mySpecialFile.dat  
s3://bucketname/prefix/filename.csv  
s3://test-bucket.s3.amazonaws.com  
S3://bucket/prefix/prefix/  
arn:aws:s3:::bucketname/prefix  
s3://arn:aws:s3:<region>:<account_id>:accesspoint/<accesspointname>  
https://<accesspointname>-<number>.s3-accesspoint.<region>.amazonaws.com
```

## Table Location and Partitions

Your source data may be grouped into Amazon S3 folders called *partitions* based on a set of columns. For example, these columns may represent the year, month, and day the particular record was created.

When you create a table, you can choose to make it partitioned. When Athena runs a SQL query against a non-partitioned table, it uses the `LOCATION` property from the table definition as the base path to list and then scan all available files. However, before a partitioned table can be queried, you must update the AWS Glue Data Catalog with partition information. This information represents the schema of files within the particular partition and the `LOCATION` of files in Amazon S3 for the partition.

- To learn how the AWS Glue crawler adds partitions, see [How Does a Crawler Determine When to Create Partitions?](#) in the *AWS Glue Developer Guide*.
- To learn how to configure the crawler so that it creates tables for data in existing partitions, see [Using Multiple Data Sources with Crawlers](#) (p. 22).
- You can also create partitions in a table directly in Athena. For more information, see [Partitioning Data](#) (p. 92).

When Athena runs a query on a partitioned table, it checks to see if any partitioned columns are used in the `WHERE` clause of the query. If partitioned columns are used, Athena requests the AWS Glue Data Catalog to return the partition specification matching the specified partition columns. The partition specification includes the `LOCATION` property that tells Athena which Amazon S3 prefix to use when reading data. In this case, *only* data stored in this prefix is scanned. If you do not use partitioned columns in the `WHERE` clause, Athena scans all the files that belong to the table's partitions.

For examples of using partitioning with Athena to improve query performance and reduce query costs, see [Top Performance Tuning Tips for Amazon Athena](#).

## 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 \(CTAS\)](#) (p. 124) queries in Athena and specify a data storage format as Parquet or ORC, or use the AWS Glue Crawler.

## Converting to Columnar Formats

Your Amazon Athena query performance improves if you convert your data into open source columnar formats, such as [Apache Parquet](#) or [ORC](#).

### Note

Use the [CREATE TABLE AS \(CTAS\)](#) (p. 131) queries to perform the conversion to columnar formats, such as Parquet and ORC, in one step.

You can do this to existing Amazon S3 data sources by creating a cluster in Amazon EMR and converting it using Hive. The following example using the AWS CLI shows you how to do this with a script and data stored in Amazon S3.

## Overview

The process for converting to columnar formats using an EMR cluster is as follows:

1. Create an EMR cluster with Hive installed.
2. In the step section of the cluster create statement, specify a script stored in Amazon S3, which points to your input data and creates output data in the columnar format in an Amazon S3 location. In this example, the cluster auto-terminates.

### Note

The script is based on Amazon EMR version 4.7 and needs to be updated to the current version. For information about versions, see [Amazon EMR Release Guide](#).

The full script is located on Amazon S3 at:

```
s3://athena-examples-myregion/conversion/write-parquet-to-s3.q
```

Here's an example script beginning with the CREATE TABLE snippet:

```
ADD JAR /usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.0.0-amzn-5.jar;
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'
with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId, referrer,
  userAgent, userCookie, ip' )
LOCATION 's3://MyRegion.elasticmapreduce/samples/hive-ads/tables/impressions/' ;
```

### Note

Replace *MyRegion* in the LOCATION clause with the region where you are running queries. For example, if your console is in us-west-1, s3://us-west-1.elasticmapreduce/samples/hive-ads/tables/.

This creates the table in Hive on the cluster which uses samples located in the Amazon EMR samples bucket.

3. On Amazon EMR release 4.7.0, include the ADD JAR line to find the appropriate JsonSerDe. The prettified sample data looks like the following:

```
{
  "number": "977680",
  "referrer": "fastcompany.com",
```

```
"processId": "1823",
"adId": "TRktxshQXAHWo261jAHubijAoNlAqA",
"browserCookie": "mvlrdwrmeF",
"userCookie": "emFlrLGrm5fA2xLFT5npwbPuG7kf6X",
"requestEndTime": "1239714001000",
"impressionId": "1I5G20RmOuG2rt7fFGFgsaWk9Xpkfb",
"userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR
2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; InfoPa",
"timers": {
  "modelLookup": "0.3292",
  "requestTime": "0.6398"
},
"threadId": "99",
"ip": "67.189.155.225",
"modelId": "bxxiuxduad",
"hostname": "ec2-0-51-75-39.amazon.com",
"sessionId": "J9NOccA3dDMFlixCuSotl9QBbjs6aS",
"requestBeginTime": "1239714000000"
}
```

4. In Hive, load the data from the partitions, so the script runs the following:

```
MSCK REPAIR TABLE impressions;
```

The script then creates a table that stores your data in a Parquet-formatted file on Amazon S3:

```
CREATE EXTERNAL TABLE parquet_hive (
  requestBeginTime string,
  adId string,
  impressionId string,
  referrer string,
  userAgent string,
  userCookie string,
  ip string
) STORED AS PARQUET
LOCATION 's3://myBucket/myParquet/';
```

The data are inserted from the *impressions* table into *parquet\_hive*:

```
INSERT OVERWRITE TABLE parquet_hive
SELECT
  requestbetime,
  adid,
  impressionid,
  referrer,
  useragent,
  usercookie,
  ip FROM impressions WHERE dt='2009-04-14-04-05';
```

The script stores the above *impressions* table columns from the date, 2009-04-14-04-05, into s3://myBucket/myParquet/ in a Parquet-formatted file.

5. After your EMR cluster is terminated, create your table in Athena, which uses the data in the format produced by the cluster.

## Before you begin

- You need to create EMR clusters. For more information about Amazon EMR, see the [Amazon EMR Management Guide](#).

- Follow the instructions found in [Setting Up \(p. 6\)](#).

## Example: Converting data to Parquet using an EMR cluster

1. Use the AWS CLI to create a cluster. If you need to install the AWS CLI, see [Installing the AWS Command Line Interface](#) in the AWS Command Line Interface User Guide.
2. You need roles to use Amazon EMR, so if you haven't used Amazon EMR before, create the default roles using the following command:

```
aws emr create-default-roles
```

3. Create an Amazon EMR cluster using the emr-4.7.0 release to convert the data using the following AWS CLI **emr create-cluster** command:

```
export REGION=us-west-1
export SAMPLEURI=s3://{REGION}.elasticmapreduce/samples/hive-ads/tables/impressions/
export S3BUCKET=myBucketName

aws emr create-cluster
--applications Name=Hadoop Name=Hive Name=HCatalog \
--ec2-attributes KeyName=myKey,InstanceProfile=EMR_EC2_DefaultRole,SubnetId=subnet-
mySubnetId \
--service-role EMR_DefaultRole \
--release-label emr-4.7.0 \
--instance-type m4.large \
--instance-count 1 \
--steps Type=HIVE,Name="Convert to Parquet",
ActionOnFailure=TERMINATE_CLUSTER,
Args=[-f,
"s3://athena-examples/conversion/write-parquet-to-s3.q",-hiveconf,
INPUT="${SAMPLEURI}",-hiveconf,
OUTPUT="s3://{S3BUCKET}/myParquet",-hiveconf,
REGION=${REGION}
] \
--region ${REGION} \
--auto-terminate
```

For more information, see [Create and Use IAM Roles for Amazon EMR](#) in the Amazon EMR Management Guide.

A successful request gives you a cluster ID.

4. Monitor the progress of your cluster using the AWS Management Console, or using the cluster ID with the *list-steps* subcommand in the AWS CLI:

```
aws emr list-steps --cluster-id myClusterID
```

Look for the script step status. If it is COMPLETED, then the conversion is done and you are ready to query the data.

5. Create the same table that you created on the EMR cluster.

You can use the same statement as above. Log into Athena and enter the statement in the **Query Editor** window:

```
CREATE EXTERNAL TABLE parquet_hive (
  requestBeginTime string,
```



```
adId string,
impressionId string,
referrer string,
userAgent string,
userCookie string,
ip string
)
STORED AS PARQUET
LOCATION 's3://myBucket/myParquet/';
```

Choose **Run Query**.

6. Run the following query to show that you can query this data:

```
SELECT * FROM parquet_hive LIMIT 10;
```

Alternatively, you can select the view (eye) icon next to the table's name in **Catalog**:



The results should show output similar to this:

|    | requestbeginTime | adid                           | impressionid                   | referrer           | useragent                                                                      |
|----|------------------|--------------------------------|--------------------------------|--------------------|--------------------------------------------------------------------------------|
| 1  | 1239682352000    | sn07U0dSU2BUek2lkJ1EKGXmhxDwhs | 5EM6xQDRXPRRvwMx4wPCWIE03930q6 | cartoonnetwork.com | Mozilla/5.1; (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.1) Gecko/20090715    |
| 2  | 1239682686000    | XaiowOqorg8rcCpUrgPr0IH091r27  | TAa4P6gEnLsweSViaABw6BmEL4InF1 | cartoonnetwork.com | Echoping/6.0.2                                                                 |
| 3  | 1239682753000    | c2sNCqusvvn7RPqCMpr0h7FVvnuDw  | ON6doUquwLE4a1pnVLhJlHmJbuHk   | cartoonnetwork.com | Echoping/6.0.2                                                                 |
| 4  | 1239682506000    | 4Xkt3ErCHRw1sN1XmMHg9ndJTipo   | 87GLC447C8BJ7sqCudcCgHgtMTg5A  | cartoonnetwork.com | Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6; en-us) AppleWebKit/525.27    |
| 5  | 1239682573000    | nAAuKDKRp26pWULS1wbBbbVEvrmHjS | cqodkEKNQ91QpDvHJ6esitkaTveia  | cartoonnetwork.com | Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6; en-us) AppleWebKit/525.27    |
| 6  | 1239682387000    | Muvf2gHNwxS5RpNnxTQgPEHfmrqQAJ | SEgg89XEIRmgWNIHRkdP0pLhvpVELx | cartoonnetwork.com | Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4 |
| 7  | 1239682595000    | cooJJd6RLuqOQ6Hpxg3jVUXRXof4   | 5f0frAsugNDI65euRaxHM18qCuXRR7 | cartoonnetwork.com | Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4 |
| 8  | 1239682537000    | IO9e9TUFqSTS0hKaetDX8xgaN7VfF  | Hu62Kiuu9ejeSIWkFrJPDtrjqKQGGM | cartoonnetwork.com | Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)     |
| 9  | 1239682667000    | QPORoongM5oDxkvnmbNEgAIF1wOWar | tpETvVW6fP5STPgF7FckLhCClns1   | cartoonnetwork.com | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET     |
| 10 | 1239682347000    | 2RWcfpDa1nXleuXwKJhaWnoqDrbSm  | MhTBNASQpJ3dJU6JWRGlg8whjFvqP  | corriere.it        | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.507     |

## Partitioning Data

By partitioning your data, you can restrict the amount of data scanned by each query, thus improving performance and reducing cost. Athena leverages Hive for [partitioning](#) data. 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 loaded one time per day, may partition by a data source identifier and date.

## 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. For more information, see [Table Location and Partitions \(p. 87\)](#).
- 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://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.

## Creating and Loading a Table with Partitioned Data

To create a table that uses partitions, you must define it during the [CREATE TABLE \(p. 406\)](#) statement. Use `PARTITIONED BY` to define the keys by which to partition data, as in the following example. `LOCATION` 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://bucket/folder/'
```

After you create the table, you load the data in the partitions for querying. For Hive-compatible data, you run [MSCK REPAIR TABLE \(p. 415\)](#). For non-Hive compatible data, you use [ALTER TABLE ADD PARTITION \(p. 402\)](#) to add the partitions manually.

## Preparing Partitioned and Nonpartitioned Data for Querying

The following sections discuss two scenarios:

1. Data is already partitioned, stored on Amazon S3, and you need to access the data on Athena.
2. Data is not partitioned.

### Scenario 1: Data already partitioned and stored on S3 in Hive format

#### Storing Partitioned Data

Partitions are stored in separate folders in Amazon S3. For example, here is the partial listing for sample ad impressions:

```
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.

## Creating a Table

To make a table out of 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'  
    with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId, referrer,  
        userAgent, userCookie, ip' )  
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 [Supported SerDes and Data Formats \(p. 363\)](#).

After you run the preceding statement in Athena, choose **New Query** and run the following command:

```
MSCK REPAIR TABLE impressions
```

Athena loads the data in the partitions.

## Query the Data

Now, 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 you data similar to the following:

```
2009-04-12-13-20    ap3HcVKAwfXtgIPu6WpuUfAfL0DQEc  
2009-04-12-13-20    17uchtodoS9kdeQP1x0XThK15IuRsV  
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    hPEPtBwk45msmwWTxPVVolkVu4v11b  
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

A layout like the following does not, however, work for automatically adding partition data with MSCK REPAIR TABLE:

```
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_$folder$

```

In this case, you would have to use `ALTER TABLE ADD PARTITION` 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. 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/'

```

## Additional Resources

- You can use CTAS and INSERT INTO to partition a dataset. For more information, see [Using CTAS and INSERT INTO for ETL and Data Analysis \(p. 133\)](#).
- You can automate adding partitions by using the [JDBC driver \(p. 72\)](#).

# 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, partition values and locations are calculated from configuration rather than read from a repository like 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.

## Pruning and Projection for Heavily Partitioned Tables

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.

## Using 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](#) (p. 34). 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.

## 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. For an example, see [Amazon Kinesis Data Firehose Example](#) (p. 109).

## Customizing the Partition Path Template

By default, Athena builds partition locations using the form `s3://<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 [Specifying Custom S3 Storage Locations](#) (p. 103).

## 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.
- 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 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.
- Views in Athena do not use projection configuration properties.

### Topics

- [Setting up Partition Projection](#) (p. 98)
- [Supported Types for Partition Projection](#) (p. 103)
- [Dynamic ID Partitioning](#) (p. 107)
- [Amazon Kinesis Data Firehose Example](#) (p. 109)

## Setting 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.

This section shows how to set these table properties for AWS Glue. To set them, you can use the AWS Glue console, Athena [CREATE TABLE](#) (p. 406) queries, or [AWS Glue API](#) operations. The following procedure shows how to set the properties in the AWS Glue console.

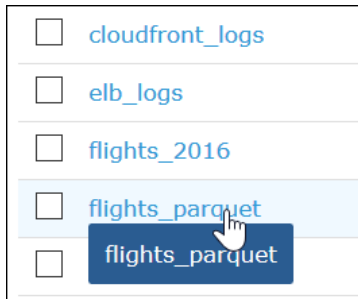
### To configure and enable partition projection using the AWS Glue console

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

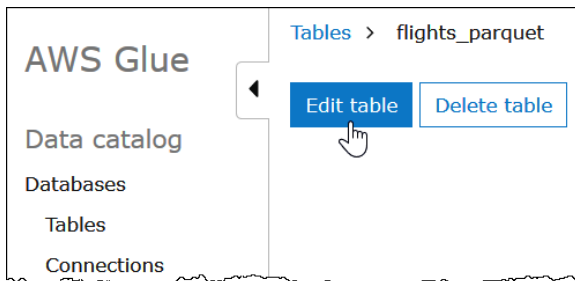
2. Choose the **Tables** tab.

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 **Edit table**.



5. In the **Edit table details** dialog box, 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 \(p. 103\)](#).
6. Following the guidance in [Supported Types for Partition Projection \(p. 103\)](#), 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 2000 through 2016.



**Edit table details**

**Table name**  
flights\_parquet

**Input format**  
org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat

**Output format**

**Table properties**

| Key                   | Value      |   |
|-----------------------|------------|---|
| last_modified_time    | 1582588443 | × |
| EXTERNAL              | TRUE       | × |
| last_modified_by      | hadoop     | × |
| projection.year.type  | integer    | × |
| projection.year.range | 2000,2016  | × |
|                       |            | × |

7. Add a key-value pair to enable partition projection. For **Key**, enter `projection.enabled`, and for its **Value**, enter `true`.

| Key                   | Value      |   |
|-----------------------|------------|---|
| last_modified_time    | 1582588443 | ✕ |
| EXTERNAL              | TRUE       | ✕ |
| last_modified_by      | hadoop     | ✕ |
| projection.year.type  | integer    | ✕ |
| projection.year.range | 2000,2016  | ✕ |
| projection.enabled    | true       | ✕ |
|                       |            |   |

Apply

**Note**

You can disable partition projection on this table at any time by setting `projection.enabled` to `false`.

8. When you are finished, choose **Apply**.
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 2000 to 2016.

✓ New query 1

New query 4 ✕

New query 5 ✕

New query 6 ✕

1

Select distinct year from flights\_parquet

2

order by year ASC

Run query

Save as

Create ▾

(Run time: 4.09 seconds, D

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

|    | year |
|----|------|
| 1  | 2000 |
| 2  | 2001 |
| 3  | 2002 |
| 4  | 2003 |
| 5  | 2004 |
| 6  | 2005 |
| 7  | 2006 |
| 8  | 2007 |
| 9  | 2008 |
| 10 | 2009 |
| 11 | 2010 |
| 12 | 2011 |
| 13 | 2012 |
| 14 | 2013 |
| 15 | 2014 |
| 16 | 2015 |
| 17 | 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).`

## Specifying 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.

### To specify a custom partition location template

- 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:
  - For **Key**, enter `storage.location.template`.
  - For **Value**, specify a location that includes a placeholder for every partition column.

The following example template values assume a table with partition columns `a`, `b`, and `c`.

```
s3://bucket/table_root/a=${a}/${b}/some_static_subdirectory/${c}/
```

```
s3://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://bucket/table_root/a=${a}/${b}/some_static_subdirectory/
```

- 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                                                         |
|------------------------------------------------|-------------------|---------------------------------------------------------------------|
| <code>projection.<i>columnName</i>.type</code> | <code>enum</code> | Required. The projection type to use for column <i>columnName</i> . |

| Property Name                             | Example Values                            | Description                                                                                                                                               |
|-------------------------------------------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                           |                                           | The value must be <code>enum</code> (case insensitive) to signal the use of the enum type. Leading and trailing white space is allowed.                   |
| <code>projection.columnName.values</code> | <code>A, B, C, D, E, F, G, Unknown</code> | Required. A comma-separated list of enumerated partition values for column <code>columnName</code> . 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 1MB 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                                                                                                                                                                                                                                                                              |
|---------------------------------------------|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>projection.columnName.type</code>     | <code>integer</code>                                                      | Required. The projection type to use for column <code>columnName</code> . The value must be <code>integer</code> (case insensitive) to signal the use of the integer type. Leading and trailing white space is allowed.                                                                  |
| <code>projection.columnName.range</code>    | <code>0, 10</code><br><code>-1, 8675309</code><br><code>0001, 9999</code> | Required. A two-element comma-separated list that provides the minimum and maximum range values to be returned by queries on the column <code>columnName</code> . These values are inclusive, can be negative, and can have leading zeroes. Leading and trailing white space is allowed. |
| <code>projection.columnName.interval</code> | <code>1</code><br><code>5</code>                                          | Optional. A positive integer that specifies the interval between successive partition values for the column <code>columnName</code> . For example, a range value of "1,3" with an interval value of "1"                                                                                  |

| Property Name                         | Example Values | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                       |                | 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.                                                                                                                                                                                                                                                                    |
| projection. <i>columnName</i> .digits | 5              | Optional. A positive integer that specifies the number of digits to include in the partition value's final representation for column <i>columnName</i> . 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. <i>columnName</i> .type  | date                                                                              | Required. The projection type to use for column <i>columnName</i> . The value must be date (case insensitive) to signal the use of the date type. Leading and trailing white space is allowed.                                                                                                                                                                                                                                                                                                                                                                                                       |
| projection. <i>columnName</i> .range | 201701,201812<br>01-01-2010,12-31-2018<br>NOW-3YEARS,NOW<br>201801,NOW<br>+1MONTH | Required. A two-element, comma-separated list which provides the minimum and maximum range values for the column <i>columnName</i> . These values are inclusive and can use any format compatible with the Java <code>java.time.*</code> date types. Both the minimum and maximum values must use the same format. The format specified in the <code>.format</code> property must be the format used for these values.<br><br>This column can also contain relative date strings, formatted in this regular expression pattern:<br><br><code>\s*NOW\s*(([\+ -])\s*([0-9]+)\s*(YEARS? MONTHS? </code> |

| Property Name                                | Example Values                                                        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                              |                                                                       | <p>WEEKS?   DAYS?   HOURS?   MINUTES?   SECONDS?)\s*)?</p> <p>White spaces are allowed, but in date literals are considered part of the date strings themselves.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| projection. <i>columnName</i> .format        | <pre>yyyyMM dd-MM-yyyy dd-MM-yyyy- HH-mm-ss</pre>                     | <p>Required. A date format string based on the Java date format <a href="#">DateTimeFormatter</a>. Can be any supported Java.time.* type.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| projection. <i>columnName</i> .interval      | <pre>5</pre>                                                          | <p>A positive integer that specifies the interval between successive partition values for column <i>columnName</i>. 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.</p> <p>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.</p> |
| projection. <i>columnName</i> .interval.unit | <pre>YEARS MONTHS WEEKS DAYS HOURS MINUTES SECONDS MILLISECONDS</pre> | <p>A time unit word that represents the serialized form of a <a href="#">ChronoUnit</a>. Possible values are YEARS, MONTHS, WEEKS, DAYS, HOURS, MINUTES, SECONDS, or MILLISECONDS. These values are case insensitive.</p> <p>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.</p>                                                                                                                                                                                                                                                        |

## 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 on an injected column fail if a filter expression on the column allows multiple values.

| Property Name                       | Value    | Description                                                                                                                                                                   |
|-------------------------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| projection. <i>columnName</i> .type | injected | Required. The projection type to use for the column <i>columnName</i> . The value specified must be injected (case insensitive). Leading and trailing white space is allowed. |

For more information, see [Injection \(p. 107\)](#).

## Dynamic ID Partitioning

You might have tables partitioned on a unique identifier column that has the following characteristics:

- Adds new values frequently, perhaps automatically.
- Cannot be easily generated. They might be user names or device IDs of varying composition or length, not sequential integers within a defined range.

For such partitioning schemes, the `enum` projection type would be impractical for the following reasons:

- You would have to modify the table properties each time a value is added to the table.
- A single table property would have millions of characters or more.
- Projection requires that all partition columns be configured for projection. This requirement could not be avoided for only one column.

To overcome these limitations, you can use `injection` or `bucketing`.

## Injection

If your query pattern on a dynamic ID dataset always specifies a single value for the high cardinality partition column, you can use value injection. Injection avoids the need to project the full partition space.

Imagine that you want to partition an IoT dataset on a UUID field that has extremely high cardinality like `device_id`. The field has the following characteristics:

- An extremely high number (potentially billions) of values.
- Because its values are random strings, it is not projectable using other projection methods.
- The extremely large number of partitions cannot be stored in commonly used metastores.

However, if all of your queries include a `WHERE` clause that filters for only a single `device_id`, you can use the following approach in your `CREATE TABLE` statement.

```
...
PARTITIONED BY
(
    device_id STRING
)
LOCATION "s3://bucket/prefix/"
TBLPROPERTIES
(
    "projection.enabled" = "true",
    "projection.device_id.type" = "injected",
    "storage.location.template" = "s3://bucket/prefix/${device_id}"
)
```

A `SELECT` query on a table like this looks like the following:



```
SELECT
  col1,
  col2,...,
  device_id
FROM
  table
WHERE
  device_id = "b6319dc2-48c1-4cd5-a0a3-a1969f7b48f7"
  AND (
    col1 > 0
    or col2 < 10
  )
```

In the example, Athena projects only a single partition for any given query. This avoids the need to store and act upon millions or billions of virtual partitions only to find one partition and read from it.

## Bucketing

In the bucketing technique, you use a fixed set of bucket values rather than the entire set of identifiers for your partitioning. If you can map an identifier to a bucket, you can use this mapping in your queries. You still benefit as when you partition on the identifiers themselves.

Bucketing has the following advantages over injection:

- You can specify more than one value at a time for a field in the `WHERE` clause.
- You can continue to use your partitions with more traditional metastores.

Using the scenario in the previous example and assuming 1 million buckets, identified by an integer, the `CREATE TABLE` statement becomes the following.

```
...
PARTITIONED BY
(
  BUCKET_ID BIGINT
)
LOCATION "s3://bucket/prefix/"
TBLPROPERTIES
(
  "projection.enabled" = "true",
  "projection.bucket_id.type" = "integer",
  "projection.bucket_id.range" = "1,1000000"
)
```

A corresponding `SELECT` query uses a mapping function in the `WHERE` clause, as in the following example.

```
SELECT
  col1,
  col2,...,
  identifier
FROM
  table
WHERE
  bucket_id = map_identifier_to_bucket("ID-IN-QUESTION")
  AND identifier = "ID-IN-QUESTION"
```

Replace the *map\_identifier\_to\_bucket* function in the example with any scalar expression that maps an identifier to an integer. For example, the expression could be a simple hash or modulus. The

function enforces a constant upper bound on the number of partitions that can ever be projected on the specified dimension. When paired with a file format that supports predicate pushdown such as Apache Parquet or ORC, the bucket technique provides good performance.

For information on writing your own user-defined function like the scalar bucketing function in the preceding example, see [Querying with User Defined Functions \(Preview\)](#) (p. 190).

## Amazon Kinesis Data Firehose Example

Kinesis Data Firehose stores data in Amazon S3 in the following path format:

```
s3://bucket/folder/yyyy/MM/dd/HH/file.extension
```

Normally, to use Athena to query Kinesis Data Firehose data without using partition projection, you create a table for Kinesis Data Firehose logs in Athena. Then you must add partitions to your table in the AWS Glue Data Catalog every hour when Kinesis Data Firehose creates a partition.

By using partition projection, you can use a one-time configuration to inform Athena where the partitions reside. The following `CREATE TABLE` example assumes a start date of 2018-01-01 at midnight. Note the use of `NOW` for the upper boundary of the date range, which allows new data to automatically become queryable at the appropriate UTC time.

```
CREATE EXTERNAL TABLE my_table
(
  ...
)
...
PARTITIONED BY
(
  datehour STRING
)
LOCATION "s3://bucket/prefix/"
TBLPROPERTIES
(
  "projection.enabled" = "true",
  "projection.datehour.type" = "date",
  "projection.datehour.range" = "2018/01/01/00,NOW",
  "projection.datehour.format" = "yyyy/MM/dd/HH",
  "projection.datehour.interval" = "1",
  "projection.datehour.interval.unit" = "HOURS",
  "storage.location.template" = "s3://bucket/prefix/${datehour}"
)
```

With this table you can run queries like the following, without having to manually add partitions:

```
SELECT *
FROM my_table
WHERE datehour >= '2018/02/03/00'
AND datehour < '2018/02/03/04'
```

# Running SQL Queries Using 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 that you connect to using Athena query federation (preview), such as Hive metastores and Amazon DocumentDB instances. For more information about working with data sources, see [Connecting to Data Sources \(p. 16\)](#). 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. When you run a query, Athena saves the results of a query in a query result location that you specify. This allows you to view query history and to download and view query results sets.

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.

## Topics

- [Working with Query Results, Output Files, and Query History \(p. 110\)](#)
- [Working with Views \(p. 119\)](#)
- [Creating a Table from Query Results \(CTAS\) \(p. 124\)](#)
- [Handling Schema Updates \(p. 142\)](#)
- [Querying Arrays \(p. 150\)](#)
- [Querying Geospatial Data \(p. 167\)](#)
- [Using Athena to Query Apache Hudi Datasets \(p. 178\)](#)
- [Querying JSON \(p. 182\)](#)
- [Using Machine Learning \(ML\) with Amazon Athena \(Preview\) \(p. 189\)](#)
- [Querying with User Defined Functions \(Preview\) \(p. 190\)](#)
- [Querying AWS Service Logs \(p. 198\)](#)
- [Querying AWS Glue Data Catalog \(p. 221\)](#)
- [Querying Web Server Logs Stored in Amazon S3 \(p. 224\)](#)

For considerations and limitations, see [Considerations and Limitations for SQL Queries in Amazon Athena \(p. 421\)](#).

## Working with Query Results, Output Files, and Query History

Amazon Athena automatically stores query results and metadata information 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.

Output files are saved automatically for every query that runs regardless of whether the query itself was saved or not. To access and view query output files, 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

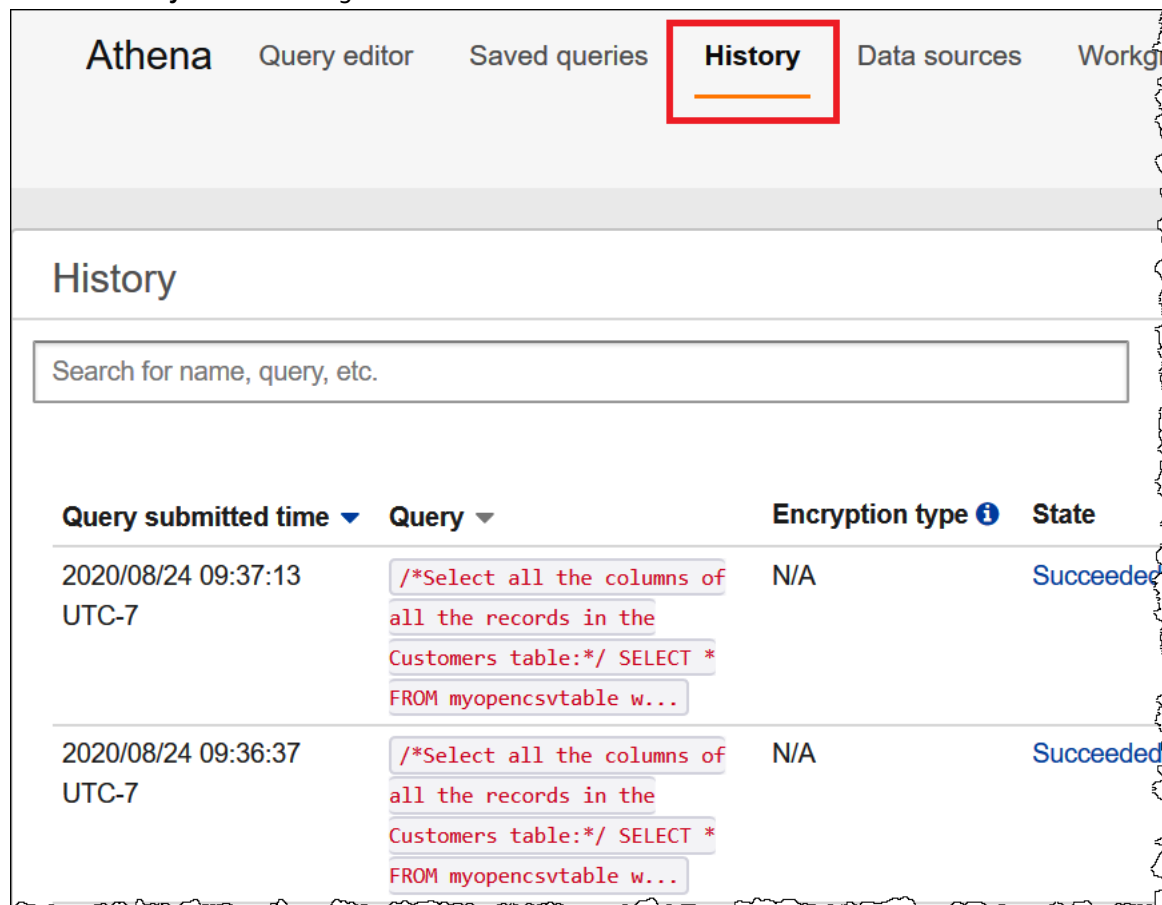
IAM principals with permission to the Amazon S3 `GetObject` action for the query result location are able to retrieve query results from Amazon S3 even if permission to the Athena `GetQueryResults` action is denied.

## Getting a Query ID

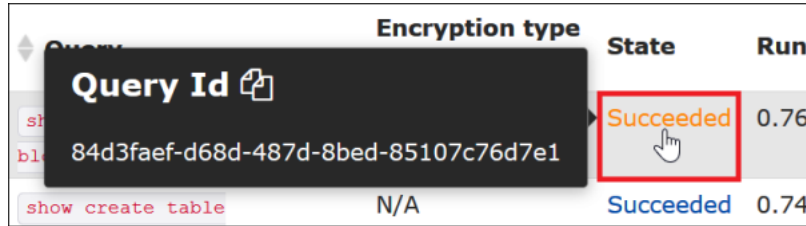
Each query that runs is known as a *query execution*. The query execution has a unique identifier known as the query ID or query execution ID. To work with query result files, and to quickly find query result files, you need the query ID. We refer to the query ID in this topic as *QueryID*.

To use the Athena console to get the *QueryID* of a query that ran

1. Choose **History** from the navigation bar.

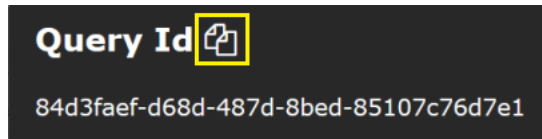


2. From the list of queries, choose the query status under **State**—for example, **Succeeded**. The query ID shows in a pointer tip.



| Query Id                             | Encryption type | State     | Run  |
|--------------------------------------|-----------------|-----------|------|
| 84d3faef-d68d-487d-8bed-85107c76d7e1 |                 | Succeeded | 0.76 |
| show create table                    | N/A             | Succeeded | 0.74 |

- To copy the ID to the clipboard, choose the icon next to **Query ID**.



## Identifying 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  | <i>QueryID</i> .csv<br><i>QueryID</i> .txt                   | DML query results files are saved in comma-separated values (CSV) format. They contain the tabular result of each query without headers. Currently, this output format cannot be changed.<br><br>DDL query results are saved as plain text files.<br><br>You can download results files from the console from the <b>Results</b> pane when using the console or from the query <b>History</b> . For more information, see <a href="#">Downloading Query Results Files Using the Athena Console</a> (p. 114). |
| Query metadata files | <i>QueryID</i> .csv.metadata<br><i>QueryID</i> .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 <code>GetQueryResults</code> action. Although these files can be deleted, we do not recommend it because                                                                                                                                                                             |

| File type           | File naming patterns         | Description                                                                                                                                                                                                                                                                                                                              |
|---------------------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                     |                              | important information about the query is lost.                                                                                                                                                                                                                                                                                           |
| Data manifest files | <i>QueryID</i> -manifest.csv | Data manifest files are generated to track files that Athena creates in Amazon S3 data source locations when an <a href="#">INSERT INTO</a> (p. 396) query runs. If a query fails, the manifest also tracks files that the query intended to write. The manifest is useful for identifying orphaned files resulting from a failed query. |

Query output files are stored in sub-folders 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. See [the section called "Specifying a Query Result Location"](#) (p. 115) below.
- The following sub-folders are created only for queries run from the console whose results path has not been overridden 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.

### To identify the query output location and query result files using the AWS CLI

- Use the `aws athena get-query-execution` command as shown 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://aws-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://aws-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"
}
```

## Downloading 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, or using the query **History**.

### To download the query results file of the most recent query

1. Enter your query in the query editor and then choose **Run query**.

When the query finishes running, the **Results** pane shows the query results.

2. To download the query results file, choose the file icon in the query results pane. Depending on your browser and browser configuration, you may need to confirm the download.

The screenshot shows the Amazon Athena console interface. At the top, there's a query editor with a tab labeled 'New query 1' and a plus icon to add more. The query text is: `1 SELECT * FROM myopencsvtable`  
`2`  
Below the editor are buttons for 'Run query' (in blue), 'Save as', and 'Create'. To the right of these buttons, it says '(Run time: 1.13 seconds, Data scanned: 0.06 KB)'. Below the buttons, a note says 'Use Ctrl + Enter to run query, Ctrl + Space to autocomplete'. To the right of this note are 'Format query' and 'Clear' buttons.  
Below the query editor is the 'Results' pane. It has a table with 4 columns: 'col1', 'col2', 'col3', and 'col4'. The first row of data shows 'a1', 'a2', 'a3', and 'a4'. The second row shows '1', '2', 'abc', and 'def'. In the top right corner of the results pane, there is a file icon (a document with a download arrow) and a tooltip that says 'Download the results in CSV format'.

### To download a query results file for an earlier query

1. Choose **History**.
2. Page through the list of queries until you find the query, and then, under **Action** for the query, choose **Download results**.

## Specifying 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.

## Specifying 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. If no query results location is specified, the query fails with an error.

Previously, 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 AWS 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.

### To specify a client-side setting query result location using the Athena console

1. From the navigation bar, choose **Settings**.
2. Enter a **Query result location**. The location you enter is used for subsequent queries unless you change it later.



**Settings**

Settings apply by default to all new queries. [Learn more](#)

**Workgroup:** **primary**

Query result location  ⓘ  
Example: s3://query-results-bucket/folder/

Encrypt query results ☐ ⓘ

Autocomplete ☐ ⓘ

If you are a member of a workgroup that specifies a query result location and overrides client-side settings, the option to change the query result location is unavailable, as the following image shows:

**Settings**

Individual query settings are overridden by workgroup settings. [Learn more](#)

**Workgroup:** **TeamA**

Query result location  ⓘ  
Example: s3://query-results-bucket/folder/

Encrypt query results ☐ ⓘ

Autocomplete ☐ ⓘ

## Specifying 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. Choose **Workgroup:****CurrentWorkgroupName** in the navigation bar.
2. Do one of the following:
  - If editing an existing workgroup, select it from the list, choose **View details**, and then choose **Edit Workgroup**.
  - If creating a new workgroup, choose **Create workgroup**.
3. For **Query result location**, choose the **Select** folder.

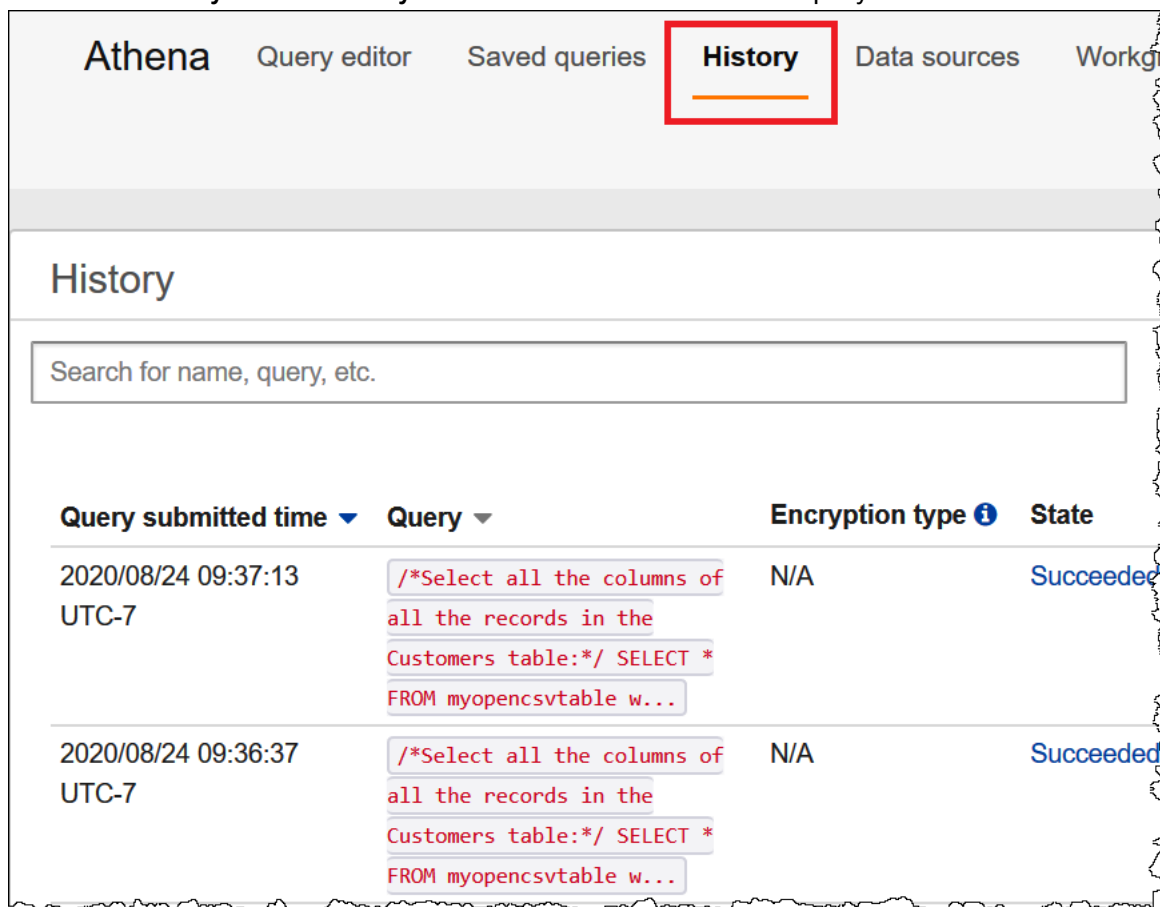
4. From the list of S3 locations, choose the blue arrow successively until the bucket and folder you want to use appears in the top line. Choose **Select**.
5. Under **Settings**, do one of the following:
  - Select **Override client-side settings** to save query files in the location that you specified above for all queries that members of this workgroup run.
  - Clear **Override client-side settings** to save query files in the location that you specified above have the query location that you specified above only when workgroup members run queries using the Athena API, ODBC driver, or JDBC driver without specifying an output location in Amazon S3.
6. If editing a workgroup, choose **Save**. If creating a workgroup, choose **Create workgroup**.

## Viewing Query History

You can use the Athena console to see the queries that succeeded and failed, download query result files for the queries that succeeded, and view error details for the queries that failed. Athena keeps a query history for 45 days.

### To view query history in the Athena console

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose the **History** tab. The **History** tab shows information about each query that ran.



The screenshot shows the Athena console interface. At the top, there are tabs: Athena, Query editor, Saved queries, **History** (highlighted with a red box), Data sources, and Workgroups. Below the tabs, the 'History' section is displayed. It includes a search bar with the placeholder text 'Search for name, query, etc.'. Below the search bar is a table with the following columns: Query submitted time, Query, Encryption type, and State. The table contains two rows of query history.

| Query submitted time ▼    | Query ▼                                                                                                 | Encryption type ⓘ | State     |
|---------------------------|---------------------------------------------------------------------------------------------------------|-------------------|-----------|
| 2020/08/24 09:37:13 UTC-7 | /*Select all the columns of all the records in the Customers table:*/ SELECT * FROM myopencsvtable w... | N/A               | Succeeded |
| 2020/08/24 09:36:37 UTC-7 | /*Select all the columns of all the records in the Customers table:*/ SELECT * FROM myopencsvtable w... | N/A               | Succeeded |

3. Do one of the following:

- To see a query statement in the Query Editor, choose the text of the query in the **Query** column. Longer query statements are abbreviated.

| Query submitted time      | Query                             |
|---------------------------|-----------------------------------|
| 2020/01/28 15:37:40 UTC-8 | Select * from elb_logs limit 10   |
| 2020/01/28 15:34:55 UTC-8 | Select * from new_parquet         |
| 2020/02/05 09:02:37 UTC-8 | show create table cloudfront_logs |

- To see a query ID, choose its **State** (**Succeeded**, **Failed**, or **Cancelled**). The query ID shows in a pointer tip.

| Query                                | Encryption type | State     | Run  |
|--------------------------------------|-----------------|-----------|------|
| 84d3faef-d68d-487d-8bed-85107c76d7e1 | N/A             | Succeeded | 0.76 |
| show create table                    | N/A             | Succeeded | 0.74 |

- To download the results of a successful query into a .csv file, choose **Download results**.

|             |     |           |       |         |                  |
|-------------|-----|-----------|-------|---------|------------------|
| elb_logs    | N/A | Succeeded | 1.76  | 3.38 MB | Download results |
| new_parquet | N/A | Cancelled | 36.36 | 0 KB    |                  |

- To see the details for a query that failed, choose **Error details** for the query.

|        |      |      |               |
|--------|------|------|---------------|
| Failed | 0.81 | 0 KB | Error details |
|--------|------|------|---------------|

**Your query has the following errors:**FAILED: ParseException line 2:0 cannot recognize input near '\_idem' 'string' ',' in column specification

This query ran against the 'sampleddb' database, unless qualified by the query. Please post the error message on our [forum](#) or [contact customer support](#) with query id.

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.

## Working with Views

A view in Amazon Athena is a logical, 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. For more information, see [CREATE VIEW](#) (p. 412).

### Topics

- [When to Use Views?](#) (p. 119)
- [Supported Actions for Views in Athena](#) (p. 120)
- [Considerations for Views](#) (p. 120)
- [Limitations for Views](#) (p. 121)
- [Working with Views in the Console](#) (p. 121)
- [Creating Views](#) (p. 122)
- [Examples of Views](#) (p. 123)
- [Updating Views](#) (p. 124)
- [Deleting Views](#) (p. 124)

## When to Use Views?

You may want to create 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 multiple tables in 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 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.
- *Experiment with optimization techniques and 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 the underlying table and column names, and minimize maintenance problems* if those names change. In that case, you recreate the view using the new names. All queries that use the view rather than the underlying tables keep running with no changes.

## Supported Actions for Views in Athena

Athena supports the following actions for views. You can run these commands in the Query Editor.

| Statement                                 | Description                                                                                                                                                                                                                                                                      |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">CREATE VIEW (p. 412)</a>      | Creates a new view from a specified <code>SELECT</code> query. For more information, see <a href="#">Creating Views (p. 122)</a> .<br><br>The optional <code>OR REPLACE</code> clause lets you update the existing view by replacing it.                                         |
| <a href="#">DESCRIBE VIEW (p. 414)</a>    | Shows the list of columns for the named view. This allows you to examine the attributes of a complex view.                                                                                                                                                                       |
| <a href="#">DROP VIEW (p. 415)</a>        | Deletes an existing view. The optional <code>IF EXISTS</code> clause suppresses the error if the view does not exist. For more information, see <a href="#">Deleting Views (p. 124)</a> .                                                                                        |
| <a href="#">SHOW CREATE VIEW (p. 418)</a> | Shows the SQL statement that creates the specified view.                                                                                                                                                                                                                         |
| <a href="#">SHOW VIEWS (p. 420)</a>       | Lists the views in the specified database, or in the current database if you omit the database name. Use the optional <code>LIKE</code> 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. |
| <a href="#">SHOW COLUMNS (p. 417)</a>     | Lists the columns in the schema for a view.                                                                                                                                                                                                                                      |

## Considerations for Views

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, if you have migrated to using it, or with Presto running on the Amazon EMR cluster connected to the same catalog. You cannot preview or add to Athena views that were created in other ways.
- If you are creating views through the AWS GlueData Catalog, you must include the `PartitionKeys` parameter and set its value to an empty list, as follows: `"PartitionKeys": [ ]`. Otherwise, your view query will fail in Athena. The following example shows a view created from the Data Catalog with `"PartitionKeys": [ ]`:

```
aws glue create-table
--database-name mydb
--table-input '{
  "Name": "test",
  "TableType": "EXTERNAL_TABLE",
  "Owner": "hadoop",
  "StorageDescriptor": {
    "Columns": [ {
      "Name": "a", "Type": "string" }, { "Name": "b", "Type": "string" } ],
    "Location": "s3://xxxxx/Oct2018/25Oct2018/",
    "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
    "OutputFormat": "org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat",
    "SerdeInfo": { "SerializationLibrary": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
    "Parameters": { "separatorChar": "|", "serialization.format": "1" } }, "PartitionKeys":
  [ ] }'
```

- 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 \(p. 244\)](#) 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 for Views

- Athena view names cannot contain special characters, other than underscore (\_). For more information, see [Names for Tables, Databases, and Columns \(p. 84\)](#).
- 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 [Reserved Keywords \(p. 85\)](#).
- You cannot use views with federated data sources, external Hive metastores, or UDFs.
- 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 [Access to Amazon S3 \(p. 243\)](#).

## Working with Views in the Console

In the Athena console, you can:

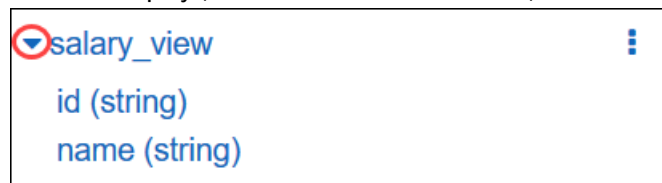
- Locate all views in the left pane, where tables are listed. Athena runs a [SHOW VIEWS \(p. 420\)](#) operation to present this list to you.
- Filter views.
- Preview a view, show its properties, edit it, or delete it.

### To list the view actions in the console

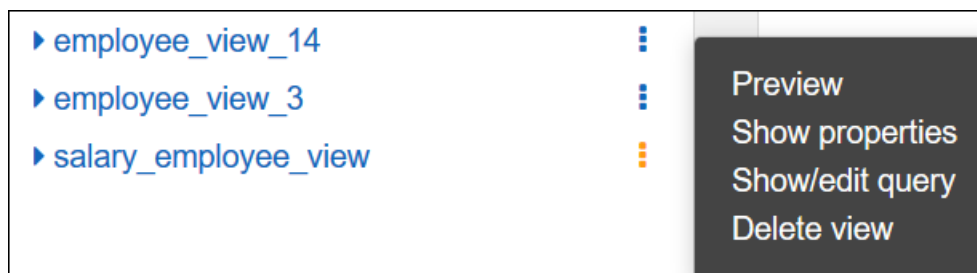
A view shows up in the console only if you have already created it.

1. In the Athena console, choose **Views**, choose a view, then expand it.

The view displays, with the columns it contains, as shown in the following example:



2. In the list of views, choose a view, and open the context (right-click) menu. The actions menu icon (:) is highlighted for the view that you chose, and the list of actions opens, as shown in the following example:



3. Choose an option. For example, **Show properties** shows the view name, the name of the database in which the table for the view is created in Athena, and the time stamp when it was created:

| View properties |                           | ✕ |  |
|-----------------|---------------------------|---|--|
| Name            | Value                     |   |  |
| View name       | employee_view             |   |  |
| Database Name   |                           |   |  |
| Create Time     | 2018/03/07 19:08:33 UTC-5 |   |  |

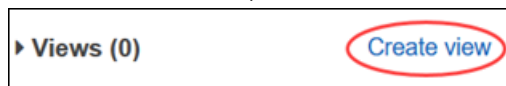
## Creating Views

You can create a view from any `SELECT` query.

### To create a view in the console

Before you create a view, choose a database and then choose a table. Run a `SELECT` query on a table and then create a view from it.

1. In the Athena console, choose **Create view**.



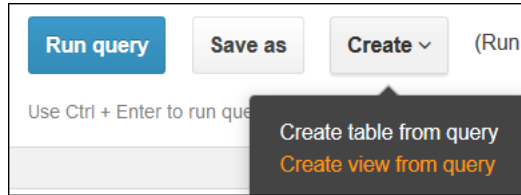
In the Query Editor, a sample view query displays.

2. Edit the sample view query. Specify the table name and add other syntax. For more information, see [CREATE VIEW \(p. 412\)](#) and [Examples of Views \(p. 123\)](#).

View names cannot contain special characters, other than underscore (`_`). See [Names for Tables, Databases, and Columns \(p. 84\)](#). Avoid using [Reserved Keywords \(p. 85\)](#) for naming views.

3. Run the view query, debug it if needed, and save it.

Alternatively, create a query in the Query Editor, and then use **Create view from query**.



If you run a view that is not valid, Athena displays an error message.

If you delete a table from which the view was created, when you 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 Views

To show the syntax of the view query, use [SHOW CREATE VIEW \(p. 418\)](#).

### 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;
```



## Updating Views

After you create a view, it appears in the **Views** list in the left pane.

To edit the view, choose it, choose the context (right-click) menu, and then choose **Show/edit query**. You can also edit the view in the Query Editor. For more information, see [CREATE VIEW \(p. 412\)](#).

## Deleting Views

To delete a view, choose it, choose the context (right-click) menu, and then choose **Delete view**. For more information, see [DROP VIEW \(p. 415\)](#).

## Creating 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 \(p. 410\)](#).

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 into other storage formats, such as Parquet and ORC. This improves query performance and reduces query costs in Athena. For information, see [Columnar Storage Formats \(p. 88\)](#).
- Create copies of existing tables that contain only the data you need.

### Topics

- [Considerations and Limitations for CTAS Queries \(p. 124\)](#)
- [Running CTAS Queries in the Console \(p. 126\)](#)
- [Bucketing vs Partitioning \(p. 129\)](#)
- [Examples of CTAS Queries \(p. 130\)](#)
- [Using CTAS and INSERT INTO for ETL and Data Analysis \(p. 133\)](#)
- [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions \(p. 139\)](#)

## Considerations and Limitations for CTAS Queries

The following table describes what you need to know about CTAS queries in Athena:

| Item              | What You Need to Know                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CTAS query syntax | <p>The CTAS query syntax differs from the syntax of <code>CREATE [EXTERNAL] TABLE</code> used for creating tables. See <a href="#">CREATE TABLE AS (p. 410)</a>.</p> <p><b>Note</b></p> <p>Table, database, or column names for CTAS queries should not contain quotes or backticks. To ensure this, check that your table, database, or column names do not represent <a href="#">reserved words (p. 85)</a>, and do not contain special characters (which require enclosing them in quotes or backticks). For more information, see <a href="#">Names for Tables, Databases, and Columns (p. 84)</a>.</p> |

| Item                              | What You Need to Know                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CTAS queries vs views             | CTAS queries write new data to a specified location in Amazon S3, whereas views do not write any data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Location of CTAS query results    | <p>If your workgroup <a href="#">overrides the client-side setting (p. 330)</a> for query results location, Athena creates your table in the location <code>s3://&lt;workgroup-query-results-location&gt;/tables/&lt;query-id&gt;/</code>. To see the query results location specified for the workgroup, <a href="#">view the workgroup's details (p. 334)</a>.</p> <p>If your workgroup does not override the query results location, you can use the syntax <code>WITH (external_location = 's3://&lt;location&gt;')</code> in your CTAS query to specify where your CTAS query results are stored.</p> <p><b>Note</b><br/>The <code>external_location</code> 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.</p> <p>If you omit the <code>external_location</code> syntax and are not using the workgroup setting, Athena uses your <a href="#">client-side setting (p. 115)</a> for the query results location and creates your table in the location <code>s3://&lt;client-query-results-location&gt;/&lt;Unsaved-or-query-name&gt;/&lt;year&gt;/&lt;month&gt;/&lt;date&gt;/tables/&lt;query-id&gt;/</code>.</p> |
| Locating Orphaned Files           | If a CTAS or <code>INSERT INTO</code> statement fails, it is possible that orphaned data are left in the data location. Because Athena does not delete any data (even 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 <a href="#">Identifying Query Output Files (p. 112)</a> and <a href="#">DataManifestLocation</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Formats for storing query results | The results of CTAS queries are stored in Parquet by default if you don't specify a data storage format. You can store CTAS results in <code>PARQUET</code> , <code>ORC</code> , <code>AVRO</code> , <code>JSON</code> , and <code>TEXTFILE</code> . Multi-character delimiters are not supported for the CTAS <code>TEXTFILE</code> format. CTAS queries do not require specifying a SerDe to interpret format transformations. See <a href="#">Example: Writing Query Results to a Different Format (p. 131)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Compression formats               | GZIP compression is used for CTAS query results by default. For Parquet and ORC, you can also specify <code>SNAPPY</code> . See <a href="#">Example: Specifying Data Storage and Compression Formats (p. 131)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Partition and Bucket Limits       | <p>You can partition and bucket the results data of a CTAS query. For more information, see <a href="#">Bucketing vs Partitioning (p. 129)</a>. Athena supports writing to 100 unique partition and bucket combinations. For example, if no buckets are defined in the destination table, you can specify a maximum of 100 partitions. If you specify five buckets, 20 partitions (each with five buckets) are allowed. If you exceed this count, an error occurs.</p> <p>Include partitioning and bucketing predicates at the end of the <code>WITH</code> clause that specifies properties of the destination table. For more information, see <a href="#">Example: Creating Bucketed and Partitioned Tables (p. 133)</a> and <a href="#">Bucketing vs Partitioning (p. 129)</a>.</p> <p>For information about working around the 100-partition limitation, see <a href="#">Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions (p. 139)</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                        |

| Item       | What You Need to Know                                                                                                                                                                                            |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Encryption | You can encrypt CTAS query results in Amazon S3, similar to the way you encrypt other query results in Athena. For more information, see <a href="#">Encrypting Query Results Stored in Amazon S3 (p. 235)</a> . |
| Data types | Column data types for a CTAS query are the same as specified for the original query.                                                                                                                             |

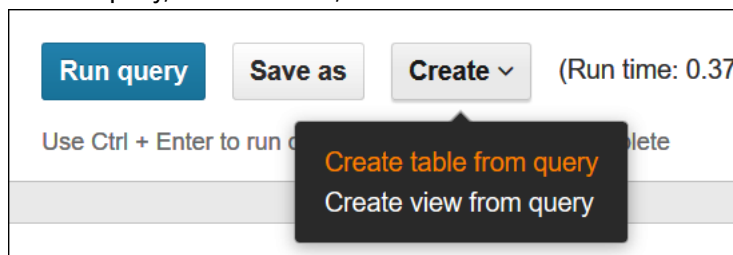
## Running CTAS Queries in the Console

In the Athena console, you can:

- [Create a CTAS query from another query \(p. 126\)](#)
- [Create a CTAS query from scratch \(p. 126\)](#)

### To create a CTAS query from another query

1. Run the query, choose **Create**, and then choose **Create table from query**.



2. In the **Create a new table on the results of a query** form, complete the fields as follows:
  - a. For **Database**, select the database in which your query ran.
  - b. For **Table name**, specify the name for your new table. Use only lowercase and underscores, such as `my_select_query_parquet`.
  - c. For **Description**, optionally add a comment to describe your query.
  - d. For **Output location**, optionally specify the location in Amazon S3, such as `s3://my_athena_results/mybucket/`. If you don't specify a location and your workgroup does not [Override Client-Side Settings \(p. 330\)](#), the following predefined location is used: `s3://aws-athena-query-results-<account>-<region>/<query-name-or-unsaved>/<year/month/date>/<query-id>/`.
  - e. For **Output data format**, select from the list of supported formats. Parquet is used if you don't specify a format. See [Columnar Storage Formats \(p. 88\)](#).

Create a new table on the results of a query

Data will be written to the default S3 location in the format specified. You can also customize the S3 location. See limitations [here](#).

Database

default

Table name\*

my\_ctas\_table

Must be lowercase and only use underscore special characters

Description

My table in Parquet

Output location

s3://my-bucket/my-folder/

Default location: s3://aws-athena-query-results--us-west-2/<query-name-or-unsaved>/2018/9/2/<query-id>/

Output data format

Parquet

Columnar (recommended)

Parquet

ORC

Storage by row

Avro

CSV

JSON

TSV

Cancel

Next

|   |                             |          |  |  |  |
|---|-----------------------------|----------|--|--|--|
| 5 | 2015-01-03T08:00:03.470121Z | elb_demo |  |  |  |
| 6 | 2015-01-03T08:00:04.159502Z | elb_demo |  |  |  |
| 7 | 2015-01-03T08:00:04.778187Z | elb_demo |  |  |  |
| 8 | 2015-01-03T08:00:06.178798Z | elb_demo |  |  |  |

|   |      |          |
|---|------|----------|
| 6 | 8888 | 0.001625 |
|   | 80   | 6.94E-4  |
|   | 8888 | 0.001639 |
|   | 443  | 5.33E-4  |

f. Choose **Next** to review your query and revise it as needed. For query syntax, see [CREATE TABLE AS \(p. 410\)](#). The preview window opens, as shown in the following example:

---

127

## Create a new table on the results of a query

Review your query and revise as needed.

Running the following query will create a table `das` in database `my_db` on the results of the query. The query will write data in `Parquet` at `s3://aws-athena-query-results-XXXXXXXXXX-us-west-2/<query-name-or-unsaved>/2018/9/4/<query-id>/`.

```
CREATE TABLE my_db.das
WITH (
  format='PARQUET'
) AS
SELECT * FROM "my_db"."json_test" limit 10;
```

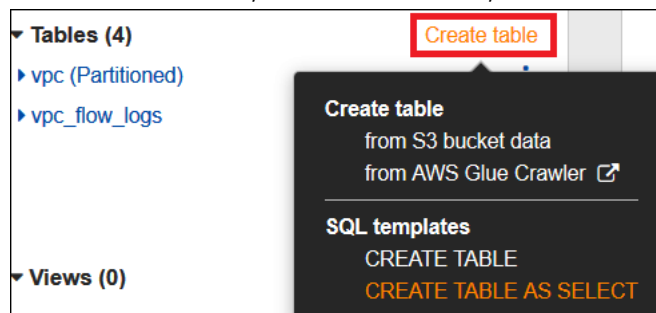
Cancel Previous Create

- g. Choose **Create**.
3. Choose **Run query**.

### To create a CTAS query from scratch

Use the `CREATE TABLE AS SELECT` template to create a CTAS query from scratch.

1. In the Athena console, choose **Create table**, and then choose **CREATE TABLE AS SELECT**.



2. In the Query Editor, edit the query as needed. For query syntax, see [CREATE TABLE AS](#) (p. 410).
3. Choose **Run query**.
4. Optionally, choose **Save as** to save the query.

See also [Examples of CTAS Queries](#) (p. 130).

## Bucketing vs Partitioning

You can specify partitioning and bucketing, for storing data from CTAS query results in Amazon S3. For information about CTAS queries, see [CREATE TABLE AS SELECT \(CTAS\)](#) (p. 124).

This section discusses partitioning and bucketing as they apply to CTAS queries only. For general guidelines about using partitioning in CREATE TABLE queries, see [Top Performance Tuning Tips for Amazon Athena](#).

Use the following tips to decide whether to partition and/or to configure bucketing, and to select columns in your CTAS queries by which to do so:

- *Partitioning CTAS query results* works well when the number of partitions you plan to have is limited. When you run a CTAS query, Athena writes the results to a specified location in Amazon S3. If you specify partitions, it creates them and stores each partition in a separate partition folder in the same location. The maximum number of partitions you can configure with CTAS query results in one query is 100. However, you can work around this limitation. For more information, see [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions](#) (p. 139).

Having partitions in Amazon S3 helps with Athena query performance, because this helps you run targeted queries for only specific partitions. Athena then scans only those partitions, saving you query costs and query time. For information about partitioning syntax, search for `partitioned_by` in [CREATE TABLE AS](#) (p. 410).

Partition data by those columns that have similar characteristics, such as records from the same department, and that can have a limited number of possible values, such as a limited number of distinct departments in an organization. This characteristic is known as *data cardinality*. For example, if you partition by the column `department`, and this column has a limited number of distinct values, partitioning by `department` works well and decreases query latency.

- *Bucketing CTAS query results* works well when you bucket data by the column that has high cardinality and evenly distributed values.

For example, columns storing `timestamp` data could potentially have a very large number of distinct values, and their data is evenly distributed across the data set. This means that a column storing `timestamp` type data will most likely have values and won't have nulls. This also means that data from such a column can be put in many buckets, where each bucket will have roughly the same amount of data stored in Amazon S3.

To choose the column by which to bucket the CTAS query results, use the column that has a high number of values (high cardinality) and whose data can be split for storage into many buckets that will have roughly the same amount of data. Columns that are sparsely populated with values are not good candidates for bucketing. This is because you will end up with buckets that have less data and other buckets that have a lot of data. By comparison, columns that you predict will almost always have values, such as `timestamp` type values, are good candidates for bucketing. This is because their data has high cardinality and can be stored in roughly equal chunks.

For more information about bucketing syntax, search for `bucketed_by` in [CREATE TABLE AS](#) (p. 410).

To conclude, you can partition and use bucketing for storing results of the same CTAS query. These techniques for writing data do not exclude each other. Typically, the columns you use for bucketing differ from those you use for partitioning.

For example, if your dataset has columns `department`, `sales_quarter`, and `ts` (for storing `timestamp` type data), you can partition your CTAS query results by `department` and `sales_quarter`.

These columns have relatively low cardinality of values: a limited number of departments and sales quarters. Also, for partitions, it does not matter if some records in your dataset have null or no values assigned for these columns. What matters is that data with the same characteristics, such as data from the same department, will be in one partition that you can query in Athena.

At the same time, because all of your data has `timestamp` type values stored in a `ts` column, you can configure bucketing for the same query results by the column `ts`. This column has high cardinality. You can store its data in more than one bucket in Amazon S3. Consider an opposite scenario: if you don't create buckets for timestamp type data and run a query for particular date or time values, then you would have to scan a very large amount of data stored in a single location in Amazon S3. Instead, if you configure buckets for storing your date- and time-related results, you can only scan and query buckets that have your value and avoid long-running queries that scan a large amount of data.

## Examples of CTAS Queries

Use the following examples to create CTAS queries. For information about the CTAS syntax, see [CREATE TABLE AS \(p. 410\)](#).

In this section:

- [Example: Duplicating a Table by Selecting All Columns \(p. 130\)](#)
- [Example: Selecting Specific Columns From One or More Tables \(p. 130\)](#)
- [Example: Creating an Empty Copy of an Existing Table \(p. 131\)](#)
- [Example: Specifying Data Storage and Compression Formats \(p. 131\)](#)
- [Example: Writing Query Results to a Different Format \(p. 131\)](#)
- [Example: Creating Unpartitioned Tables \(p. 131\)](#)
- [Example: Creating Partitioned Tables \(p. 132\)](#)
- [Example: Creating Bucketed and Partitioned Tables \(p. 133\)](#)

### 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

The following example uses a CTAS query to create a new table with Parquet data from a source table in a different format. You can specify `PARQUET`, `ORC`, `AVRO`, `JSON`, and `TEXTFILE` in a similar way.

This example also specifies compression as `SNAPPY`. If omitted, `GZIP` is used. `GZIP` and `SNAPPY` are the supported compression formats for CTAS query results stored in Parquet and ORC.

```
CREATE TABLE new_table
WITH (
    format = 'Parquet',
    parquet_compression = 'SNAPPY')
AS SELECT *
FROM old_table;
```

The following example is similar, but it stores the CTAS query results in ORC and uses the `orc_compression` parameter to specify the compression format. If you omit the compression format, Athena uses `GZIP` by default.

```
CREATE TABLE new_table
WITH (format = 'ORC',
    orc_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://my_athena_results/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://my_athena_results/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://my_athena_results/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://my_athena_results/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 \(p. 411\)](#). For more information about choosing the columns for partitioning, see [Bucketing vs Partitioning \(p. 129\)](#).

#### 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://my_athena_results/ctas_csv_partitioned/',  
partitioned_by = ARRAY['key1'])  
AS SELECT name1, address1, comment1, key1  
FROM table1;
```

```
CREATE TABLE ctas_json_partitioned  
WITH (  
    format = 'JSON',  
    external_location = 's3://my_athena_results/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 \(p. 411\)](#).

For information about choosing the columns for bucketing, see [Bucketing vs Partitioning \(p. 129\)](#).

```
CREATE TABLE ctas_avro_bucketed  
WITH (  
    format = 'AVRO',  
    external_location = 's3://my_athena_results/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;
```

## Using CTAS and INSERT INTO for ETL and Data Analysis

You can use Create Table as Select ([CTAS \(p. 124\)](#)) and [INSERT INTO \(p. 396\)](#) 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 \(p. 391\)](#) 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 \(p. 134\)](#)
- [Step 2: Use CTAS to Partition, Convert, and Compress the Data \(p. 135\)](#)
- [Step 3: Use INSERT INTO to Add Data \(p. 136\)](#)
- [Step 4: Measure Performance and Cost Differences \(p. 137\)](#)

## 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 query editor, run the [CREATE DATABASE \(p. 406\)](#) command to create a database. To avoid Amazon S3 cross-Region data transfer charges, run this and the other queries in this topic in the us-east-1 Region.

```
CREATE DATABASE blogdb
```

2. Run the following statement to [create a table \(p. 406\)](#).

```
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 \(p. 124\)](#) 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://your-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

- To show the partitions created, run the following AWS CLI command. Be sure to include the final forward slash (/).

```
aws s3 ls s3://your-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://your-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 \(p. 396\)](#) 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://your-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://your-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 [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions \(p. 139\)](#).

## 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.

## Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions

You can create up to 100 partitions per query with a `CREATE TABLE AS SELECT` (CTAS (p. 124)) query. Similarly, you can add a maximum of 100 partitions to a destination table with an `INSERT INTO` statement. To work around these limitations, 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://<my-tpch-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://<my-tpch-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_linenum,
       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 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.

## Handling 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**

- [Summary: Updates and Data Formats in Athena \(p. 142\)](#)
- [Index Access in ORC and Parquet \(p. 143\)](#)
- [Types of Updates \(p. 145\)](#)
- [Updates in Tables with Partitions \(p. 149\)](#)

## Summary: Updates and Data Formats in Athena

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 [Index Access in ORC and Parquet \(p. 143\)](#).

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 | JSON | AVRO | PARQUET Read by Name (default) | PARQUET Read by Index | ORC: Read by Index (default) | ORC: Read by Name |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|------|------|--------------------------------|-----------------------|------------------------------|-------------------|
| <a href="#">Rename columns (p. 147)</a>                                             | 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                 |
| <a href="#">Add columns at the beginning or in the middle of the table (p. 146)</a> | 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                 |
| <a href="#">Add columns at the end of the table (p. 146)</a>                        | Store your data in CSV or TSV, JSON, AVRO, ORC, or Parquet.                                                                                                                           | Y                                      | Y    | Y    | Y                              | Y                     | Y                            | Y                 |
| <a href="#">Remove columns (p. 146)</a>                                             | Store your data in JSON, AVRO, or Parquet and ORC, if they are read by name. Do not use CSV and TSV.                                                                                  | N                                      | Y    | Y    | Y                              | N                     | N                            | Y                 |
| <a href="#">Reorder columns (p. 148)</a>                                            | Store your data in AVRO, JSON or ORC and Parquet if they are read by name.                                                                                                            | N                                      | Y    | Y    | Y                              | N                     | N                            | Y                 |
| <a href="#">Change a column's data type (p. 148)</a>                                | 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                 |

## Index Access in ORC and 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 \(p. 144\)](#).
- Athena reads *Parquet by name by default*, as defined in `SERDEPROPERTIES` (`'parquet.column.index.access'='false'`). For more information, see [PARQUET: Read by Name \(p. 145\)](#).

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**

When `orc.column.index.access` is set to `false`, Athena becomes case sensitive. This can prevent Athena from reading data if you are using Spark, which requires lower case, and have column names that use uppercase. The workaround is to rename the columns to lower case.

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://schema_updates/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`.

## Types of Updates

Here are the types of updates that a table's schema can have. We review each type of schema update and specify which data formats allow you to have them in Athena.

- [Adding Columns at the Beginning or Middle of the Table \(p. 146\)](#)
- [Adding Columns at the End of the Table \(p. 146\)](#)
- [Removing Columns \(p. 146\)](#)
- [Renaming Columns \(p. 147\)](#)
- [Reordering Columns \(p. 148\)](#)
- [Changing a Column's Data Type \(p. 148\)](#)

Depending on how you expect your schemas to evolve, to continue using Athena queries, choose a compatible data format.

Let's 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://schema_updates/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://schema_updates/orders_csv/';
```

In the following sections, we review how updates to these tables affect Athena queries.

## Adding 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 [Index Access in ORC and Parquet \(p. 143\)](#).

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 shows adding a column to a JSON table in the middle of the table:

```
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://schema_updates/orders_json/';
```

## Adding 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.

## Removing 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 [Index Access in ORC and Parquet \(p. 143\)](#).

- 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://schema_updates/orders_parquet/';
```

## Renaming 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 [Index Access in ORC and Parquet \(p. 143\)](#).

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://schema_updates/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://schema_updates/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;
```

## Reordering 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 [Index Access in ORC and Parquet \(p. 143\)](#).

The following example illustrates reordering of columns:

```
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://schema_updates/orders_parquet/';
```

## Changing a Column's Data Type

You change column types because a column's data type can no longer hold the amount of information, for example, when an ID column exceeds the size of an `INT` data type and has to change to a `BIGINT` data type.

Changing a column's data type has these limitations:

- Only certain data types can be converted to other data types. See the table in this section for data types that can change.
- For data in Parquet and ORC, you cannot change a column's data type 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 [Avoiding Schema Mismatch Errors for Tables with Partitions \(p. 150\)](#).

### Important

We strongly suggest that you test and verify your queries before performing data type translations. If Athena cannot convert the data type from the original data type to the target data type, the `CREATE TABLE` query may fail.

The following table lists data types that you can change:

### Compatible 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                               |

In the following example of the `orders_json` table, change the data type for the column ``o_shippriority`` to `BIGINT`:

```
CREATE EXTERNAL TABLE orders_json (  
  `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://schema_updates/orders_json';
```

The following query runs successfully, similar to the original `SELECT` query, before the data type change:

```
Select * from orders_json  
LIMIT 10;
```

## Updates in 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 \(p. 142\)](#).

## Avoiding 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 \(p. 404\)](#) and [ALTER TABLE ADD PARTITION \(p. 402\)](#).

## Querying Arrays

Amazon Athena lets you create arrays, concatenate them, convert them to different data types, and then filter, flatten, and sort them.

### Topics

- [Creating Arrays \(p. 151\)](#)
- [Concatenating Strings and Arrays \(p. 152\)](#)
- [Converting Array Data Types \(p. 153\)](#)
- [Finding Lengths \(p. 154\)](#)
- [Accessing Array Elements \(p. 154\)](#)
- [Flattening Nested Arrays \(p. 155\)](#)
- [Creating Arrays from Subqueries \(p. 157\)](#)
- [Filtering Arrays \(p. 158\)](#)

- [Sorting Arrays \(p. 159\)](#)
- [Using Aggregation Functions with Arrays \(p. 160\)](#)
- [Converting Arrays to Strings \(p. 161\)](#)
- [Using Arrays to Create Maps \(p. 161\)](#)
- [Querying Arrays with Complex Types and Nested Structures \(p. 162\)](#)

## Creating 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}] |
+-----+
+
```

## Concatenating Strings and Arrays

### Concatenating 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
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  
This is a test.
```

## Concatenating 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  
[[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  
[Hello, Amazon, Athena, Hi, Alexa]
```

For more information about `concat()` other string functions, see [String Functions and Operators](#) in the Presto documentation.

## Converting 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}] |
+-----+
```

## Finding 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          |
+-----+
```

## Accessing 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    |
+-----+
```

## Flattening 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.

### Examples

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] |
+-----+
```

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    |
+-----+
```

## Creating 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] |
+-----+
```

## Filtering 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] |
+-----+
```

## The filter Function

```
filter(ARRAY [list_of_values], boolean_function)
```

The filter function creates an array from 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 creates an array from the 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 creates an array that consists of the non-null values from the array [-1, NULL, 10, NULL].

```
SELECT filter(ARRAY [-1, NULL, 10, NULL], q -> q IS NOT NULL)
```

### Results

```
[-1,10]
```

## Sorting Arrays

Create a sorted array of unique values from a set of rows.

```
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] |
+-----+
```

## Using 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 [Creating Arrays from Subqueries \(p. 157\)](#).

### Note

`ORDER BY` is not supported for aggregation functions, for example, you cannot use it within `array_agg(x)`.

```
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
+-----+
```

## Converting 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 |
+-----+
```

## Using 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       |  
+-----+
```

## Querying 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.

- [Creating a ROW \(p. 162\)](#)
- [Changing Field Names in Arrays Using CAST \(p. 163\)](#)
- [Filtering Arrays Using the . Notation \(p. 163\)](#)
- [Filtering Arrays with Nested Values \(p. 164\)](#)
- [Filtering Arrays Using UNNEST \(p. 165\)](#)
- [Finding Keywords in Arrays Using regexp\\_like \(p. 165\)](#)

### Creating 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} |
+-----+
```

## Changing 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`.

## Filtering 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 \(p. 203\)](#).

```
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}] |
+-----+
```

## Filtering 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  |
+-----+
```

## Filtering 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 \(p. 155\)](#).

For example, this query finds hostnames 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  |  
+-----+
```

## Finding 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 hostname, 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 | big data | 10    |

## Querying 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.

### Topics

- [What is a Geospatial Query? \(p. 168\)](#)
- [Input Data Formats and Geometry Data Types \(p. 168\)](#)
- [List of Supported Geospatial Functions \(p. 168\)](#)
- [Examples: Geospatial Queries \(p. 177\)](#)

## 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` 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 \(p. 374\)](#).

### Geometry Data Types

To handle geospatial queries, Athena supports these specialized geometry data types:

- `point`
- `line`
- `polygon`
- `multiline`
- `multipolygon`

## List of Supported Geospatial Functions

Geospatial functions in Athena have these characteristics:

- The functions follow the general principles of [Spatial Query](#).
- The functions are implemented as a Presto plugin that uses the ESRI Java Geometry Library. This library has an Apache 2 license.
- The functions rely on the [ESRI Geometry API](#).
- Not all of the ESRI-supported functions are available in Athena. This topic lists only the ESRI geospatial functions that are supported in Athena.

Athena supports four types of geospatial functions:

- [Constructor Functions](#) (p. 170)
- [Geospatial Relationship Functions](#) (p. 171)
- [Operation Functions](#) (p. 173)
- [Accessor Functions](#) (p. 174)

## Before You Begin

Create two tables, earthquakes and counties, as follows:

```
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://my-query-log/csv'
```

```
CREATE external TABLE IF NOT EXISTS counties
(
  Name string,
  BoundaryShape binary
)
ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.JsonSerde'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedJsonInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://my-query-log/json'
```

Some of the subsequent examples are based on these tables and rely on two sample files stored in the Amazon S3 location. These files are not included with Athena and are used for illustration purposes only:

- An earthquakes.csv file, which lists earthquakes that occurred in California. This file has fields that correspond to the fields in the table earthquakes.
- A california-counties.json file, which lists JSON-encoded county data in the ESRI-compliant format, and includes many fields, such as AREA, PERIMETER, STATE, COUNTY, and NAME. The counties table is based on this file and has two fields only: Name (string), and BoundaryShape (binary).

## Constructor Functions

Use constructor functions to obtain binary representations of `point`, `line`, or `polygon` geometry data types. You can also use these functions to convert binary data to text, and obtain binary values for geometry data that is expressed as Well-Known Text (WKT).

### `ST_POINT(double, double)`

Returns a binary representation of a `point` geometry data type.

To obtain the `point` geometry data type, use the `ST_POINT` function in Athena. For the input data values to this function, use geometric values, such as values in the Universal Transverse Mercator (UTM) Cartesian coordinate system, or geographic map units (longitude and latitude) in decimal degrees. The longitude and latitude values use the World Geodetic System, also known as WGS 1984, or EPSG:4326. WGS 1984 is the coordinate system used by the Global Positioning System (GPS).

For example, in the following notation, the map coordinates are specified in longitude and latitude, and the value `.072284`, which is the buffer distance, is specified in angular units as decimal degrees:

```
ST_BUFFER(ST_POINT(-74.006801, 40.705220), .072284)
```

Syntax:

```
SELECT ST_POINT(longitude, latitude) FROM earthquakes LIMIT 1;
```

Example. This example uses specific longitude and latitude coordinates from `earthquakes.csv`:

```
SELECT ST_POINT(61.56, -158.54)
FROM earthquakes
LIMIT 1;
```

It returns this binary representation of a geometry data type `point`:

```
00 00 00 00 01 01 00 00 00 48 e1 7a 14 ae c7 4e 40 e1 7a 14 ae 47 d1 63 c0
```

The next example uses specific longitude and latitude coordinates:

```
SELECT ST_POINT(-74.006801, 40.705220);
```

It returns this binary representation of a geometry data type `point`:

```
00 00 00 00 01 01 00 00 00 20 25 76 6d 6f 80 52 c0 18 3e 22 a6 44 5a 44 40
```

In the following example, we use the `ST_GEOMETRY_TO_TEXT` function to obtain the binary values from WKT:

```
SELECT ST_GEOMETRY_TO_TEXT(ST_POINT(-74.006801, 40.705220)) AS WKT;
```

This query returns a WKT representation of the `point` geometry type: `1 POINT (-74.006801 40.70522)`.

### `ST_LINE(varchar)`

Returns a value in the `line` data type, which is a binary representation of the [geometry data type \(p. 168\)](#) `line`. Example:

```
SELECT ST_Line('linestring(1 1, 2 2, 3 3)')
```

### **ST\_POLYGON**(varchar)

Returns a value in the polygon data type, which is a binary representation of the [geometry data type](#) (p. 168) polygon. Example:

```
SELECT ST_POLYGON('polygon ((1 1, 1 4, 4 4, 4 1))')
```

### **ST\_GEOMETRY\_TO\_TEXT** (varbinary)

Converts each of the specified [geometry data types](#) (p. 168) to text. Returns a value in a geometry data type, which is a WKT representation of the geometry data type. Example:

```
SELECT ST_GEOMETRY_TO_TEXT(ST_POINT(61.56, -158.54))
```

### **ST\_GEOMETRY\_FROM\_TEXT** (varchar)

Converts text into a geometry data type. Returns a value in a geometry data type, which is a binary representation of the geometry data type. Example:

```
SELECT ST_GEOMETRY_FROM_TEXT(ST_GEOMETRY_TO_TEXT(ST_Point(1, 2)))
```

#### **Note**

If you are using the AmazonAthenaPreviewFunctionality workgroup, use the syntax `ST_GEOMETRYFROMTEXT (varchar)`.

## Geospatial Relationship Functions

The following functions express relationships between two different geometries that you specify as input. They return results of type `boolean`. The order in which you specify the pair of geometries matters: the first geometry value is called the left geometry, the second geometry value is called the right geometry.

These functions return:

- `TRUE` if and only if the relationship described by the function is satisfied.
- `FALSE` if and only if the relationship described by the function is not satisfied.

### **ST\_CONTAINS** (geometry, geometry)

Returns `TRUE` if and only if the left geometry contains the right geometry. Examples:

```
SELECT ST_CONTAINS('POLYGON((0 2,1 1,0 -1,0 2))', 'POLYGON((-1 3,2 1,0 -3,-1 3))')
```

```
SELECT ST_CONTAINS('POLYGON((0 2,1 1,0 -1,0 2))', ST_Point(0, 0));
```

```
SELECT ST_CONTAINS(ST_GEOMETRY_FROM_TEXT('POLYGON((0 2,1 1,0 -1,0 2))'),  
ST_GEOMETRY_FROM_TEXT('POLYGON((-1 3,2 1,0 -3,-1 3))'))
```

### **ST\_CROSSES** (geometry, geometry)

Returns `TRUE` if and only if the left geometry crosses the right geometry. Example:



```
SELECT ST_CROSSES(ST_LINE('linestring(1 1, 2 2)'), ST_LINE('linestring(0 1, 2 2)'))
```

### ST\_DISJOINT (geometry, geometry)

Returns TRUE if and only if the intersection of the left geometry and the right geometry is empty.

Example:

```
SELECT ST_DISJOINT(ST_LINE('linestring(0 0, 0 1)'), ST_LINE('linestring(1 1, 1 0)'))
```

### ST\_EQUALS (geometry, geometry)

Returns TRUE if and only if the left geometry equals the right geometry. Example:

```
SELECT ST_EQUALS(ST_LINE('linestring( 0 0, 1 1)'), ST_LINE('linestring(1 3, 2 2)'))
```

### ST\_INTERSECTS (geometry, geometry)

Returns TRUE if and only if the left geometry intersects the right geometry. Example:

```
SELECT ST_INTERSECTS(ST_LINE('linestring(8 7, 7 8)'), ST_POLYGON('polygon((1 1, 4 1, 4 4, 1 4))'))
```

### ST\_OVERLAPS (geometry, geometry)

Returns TRUE if and only if the left geometry overlaps the right geometry. Example:

```
SELECT ST_OVERLAPS(ST_POLYGON('polygon((2 0, 2 1, 3 1))'), ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

### ST\_RELATE (geometry, geometry)

Returns TRUE if and only if the left geometry has the specified Dimensionally Extended nine-Intersection Model (DE-9IM) relationship with the right geometry. For more information, see the Wikipedia topic [DE-9IM](#). Example:

```
SELECT ST_RELATE(ST_LINE('linestring(0 0, 3 3)'), ST_LINE('linestring(1 1, 4 4)'),  
  'T*****')
```

### ST\_TOUCHES (geometry, geometry)

Returns TRUE if and only if the left geometry touches the right geometry.

Example:

```
SELECT ST_TOUCHES(ST_POINT(8, 8), ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

### ST\_WITHIN (geometry, geometry)

Returns TRUE if and only if the left geometry is within the right geometry.

Example:

```
SELECT ST_WITHIN(ST_POINT(8, 8), ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

## Operation Functions

Use operation functions to perform operations on geometry data type values. For example, you can obtain the boundaries of a single geometry data type; intersections between two geometry data types; difference between left and right geometries, where each is of the same geometry data type; or an exterior buffer or ring around a particular geometry data type.

All operation functions take as an input one of the geometry data types and return their binary representations.

### ST\_BOUNDARY (geometry)

Takes as an input one of the geometry data types, and returns a binary representation of the boundary geometry data type.

Examples:

```
SELECT ST_BOUNDARY(ST_LINE('linestring(0 1, 1 0)'))
```

```
SELECT ST_BOUNDARY(ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

### ST\_BUFFER (geometry, double)

Takes as an input one of the geometry data types, such as point, line, polygon, multiline, or multipolygon, and a distance as type `double`. Returns a binary representation of the geometry data type buffered by the specified distance (or radius). Example:

```
SELECT ST_BUFFER(ST_Point(1, 2), 2.0)
```

In the following example, the map coordinates are specified in longitude and latitude, and the value `.072284`, which is the buffer distance, is specified in angular units as decimal degrees:

```
ST_BUFFER(ST_POINT(-74.006801, 40.705220), .072284)
```

### ST\_DIFFERENCE (geometry, geometry)

Returns a binary representation of a difference between the left geometry and right geometry. Example:

```
SELECT ST_GEOMETRY_TO_TEXT(ST_DIFFERENCE(ST_POLYGON('polygon((0 0, 0 10, 10 10, 10 0))'),  
ST_POLYGON('polygon((0 0, 0 5, 5 5, 5 0))')))
```

### ST\_ENVELOPE (geometry)

Takes as an input line, polygon, multiline, and multipolygon geometry data types. Does not support point geometry data type. Returns a binary representation of an envelope, where an envelope is a rectangle around the specified geometry data type. Examples:

```
SELECT ST_ENVELOPE(ST_LINE('linestring(0 1, 1 0)'))
```

```
SELECT ST_ENVELOPE(ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

### ST\_EXTERIOR\_RING (geometry)

Returns a binary representation of the exterior ring of the input type polygon. Examples:

```
SELECT ST_EXTERIOR_RING(ST_POLYGON(1,1, 1,4, 4,1))
```

```
SELECT ST_EXTERIOR_RING(ST_POLYGON('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))'))
```

### ST\_INTERSECTION (geometry, geometry)

Returns a binary representation of the intersection of the left geometry and right geometry. Examples:

```
SELECT ST_INTERSECTION(ST_POINT(1,1), ST_POINT(1,1))
```

```
SELECT ST_INTERSECTION(ST_LINE('linestring(0 1, 1 0)'), ST_POLYGON('polygon((1 1, 1 4, 4 4, 4 1))'))
```

```
SELECT ST_GEOMETRY_TO_TEXT(ST_INTERSECTION(ST_POLYGON('polygon((2 0, 2 3, 3 0))'),  
ST_POLYGON('polygon((1 1, 4 1, 4 4, 1 4))')))
```

### ST\_SYMMETRIC\_DIFFERENCE (geometry, geometry)

Returns a binary representation of the geometrically symmetric difference between left geometry and right geometry. Example:

```
SELECT ST_GEOMETRY_TO_TEXT(ST_SYMMETRIC_DIFFERENCE(ST_LINE('linestring(0 2, 2 2)'),  
ST_LINE('linestring(1 2, 3 2)')))
```

## Accessor Functions

Accessor functions are useful to obtain values in types `varchar`, `bigint`, or `double` from different geometry data types, where `geometry` is any of the geometry data types supported in Athena: `point`, `line`, `polygon`, `multiline`, and `multipolygon`. For example, you can obtain an area of a `polygon` geometry data type, maximum and minimum X and Y values for a specified geometry data type, obtain the length of a `line`, or receive the number of points in a specified geometry data type.

### ST\_AREA (geometry)

Takes as an input a geometry data type `polygon` and returns an area in type `double`. Example:

```
SELECT ST_AREA(ST_POLYGON('polygon((1 1, 4 1, 4 4, 1 4))'))
```

### ST\_CENTROID (geometry)

Takes as an input a [geometry data type \(p. 168\)](#) `polygon`, and returns a point that is the center of the polygon's envelope in type `varchar`. Examples:

```
SELECT ST_CENTROID(ST_GEOMETRY_FROM_TEXT('polygon ((0 0, 3 6, 6 0, 0 0))'))
```

```
SELECT ST_GEOMETRY_TO_TEXT(ST_CENTROID(ST_ENVELOPE(ST_GEOMETRY_FROM_TEXT('POINT (53  
27)'))))
```

### ST\_COORDINATE\_DIMENSION (geometry)

Takes as input one of the supported [geometry data types \(p. 168\)](#), and returns the count of coordinate components in type bigint. Example:

```
SELECT ST_COORDINATE_DIMENSION(ST_POINT(1.5,2.5))
```

### ST\_DIMENSION (geometry)

Takes as an input one of the supported [geometry data types \(p. 168\)](#), and returns the spatial dimension of a geometry in type bigint. Example:

```
SELECT ST_DIMENSION(ST_POLYGON('polygon((1 1, 4 1, 4 4, 1 4))'))
```

### ST\_DISTANCE (geometry, geometry)

Returns, based on spatial ref, the two-dimensional minimum Cartesian distance between two geometries in projected units. Example:

```
SELECT ST_DISTANCE(ST_POINT(0.0,0.0), ST_POINT(3.0,4.0))
```

### ST\_IS\_CLOSED (geometry)

Takes as an input only line and multiline [geometry data types \(p. 168\)](#). Returns TRUE (type boolean) if and only if the line is closed. Example:

```
SELECT ST_IS_CLOSED(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_IS\_EMPTY (geometry)

Takes as an input only line and multiline [geometry data types \(p. 168\)](#). Returns TRUE (type boolean) if and only if the specified geometry is empty, in other words, when the line start and end values co-inside. Example:

```
SELECT ST_IS_EMPTY(ST_POINT(1.5, 2.5))
```

### ST\_IS\_RING (geometry)

Returns TRUE (type boolean) if and only if the line type is closed and simple. Example:

```
SELECT ST_IS_RING(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_LENGTH (geometry)

Returns the length of line in type double. Example:

```
SELECT ST_LENGTH(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_MAX\_X (geometry)

Returns the maximum X coordinate of a geometry in type double. Example:

```
SELECT ST_MAX_X(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_MAX\_Y (geometry)

Returns the maximum Y coordinate of a geometry in type double. Example:

```
SELECT ST_MAX_Y(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_MIN\_X (geometry)

Returns the minimum X coordinate of a geometry in type double. Example:

```
SELECT ST_MIN_X(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_MIN\_Y (geometry)

Returns the minimum Y coordinate of a geometry in type double. Example:

```
SELECT ST_MAX_Y(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_START\_POINT (geometry)

Returns the first point of a line geometry data type in type point. Example:

```
SELECT ST_START_POINT(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_END\_POINT (geometry)

Returns the last point of a line geometry data type in type point. Example:

```
SELECT ST_END_POINT(ST_LINE('linestring(0 2, 2 2)'))
```

### ST\_X (point)

Returns the X coordinate of a point in type double. Example:

```
SELECT ST_X(ST_POINT(1.5, 2.5))
```

### ST\_Y (point)

Returns the Y coordinate of a point in type double. Example:

```
SELECT ST_Y(ST_POINT(1.5, 2.5))
```

### ST\_POINT\_NUMBER (geometry)

Returns the number of points in the geometry in type bigint. Example:

```
SELECT ST_POINT_NUMBER(ST_POINT(1.5, 2.5))
```

### ST\_INTERIOR\_RING\_NUMBER (geometry)

Returns the number of interior rings in the polygon geometry in type bigint. Example:

```
SELECT ST_INTERIOR_RING_NUMBER(ST_POLYGON('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))'))
```

## 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.EnclosedJsonInputFormat` 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://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.JsonSerde'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedJsonInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://my-query-log/json/';
```

The following code example uses the `CROSS JOIN` function for the two tables created earlier. Additionally, for both tables, it uses `ST_CONTAINS` and asks for counties whose boundaries include a geographical location of the earthquakes, specified with `ST_POINT`. It then 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 (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:

- [Querying OpenStreetMap with Amazon Athena](#)
- [Visualize over 200 years of global climate data using Amazon Athena and Amazon QuickSight.](#)

## Using Athena to 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 S3 using open storage formats. Currently, Athena can read compacted Hudi datasets but not write Hudi data. Athena uses Apache Hudi version 0.5.2-incubating, subject to change. For more information about this Hudi version, see [apache/hudi release-0.5.2](#) on GitHub.com.

## Hudi Dataset Storage Types

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 logical views for accessing the data:

- **Read-optimized view** – Provides the latest committed dataset from CoW tables and the latest compacted dataset from MoR tables.
- **Incremental view** – Provides a change stream between two actions out of a CoW dataset to feed downstream jobs and extract, transform, load (ETL) workflows.
- **Real-time view** – Provides the latest committed data from a MoR table by merging the columnar and row-based files inline.

Currently, Athena supports only the first of these: the read-optimized view. Queries on a read-optimized view return all compacted data, which provides good performance but does not include the latest delta commits. For more information about the tradeoffs between storage types, see [Storage Types & Views](#) in the Apache Hudi documentation.

## Considerations and Limitations

- Athena supports reading of the compacted view of Hudi data only.
  - For Copy on Write (CoW), Athena supports snapshot queries.
  - For Merge on Read (MoR), Athena supports read optimized queries.



- Athena does not support [CTAS \(p. 124\)](#) or [INSERT INTO \(p. 396\)](#) 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 Hudi Tables](#) in the Apache Hudi documentation.
- 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 \(p. 402\)](#).

## Creating Hudi Tables

This section provides examples of CREATE TABLE statements in Athena for partitioned and nonpartitioned tables of Hudi data.

If you have Hudi tables already created in AWS Glue, you can query them directly in Athena. When you create Hudi tables in Athena, you must run ALTER TABLE ADD PARTITION to load the Hudi data before you can query it.

### Copy on Write (CoW) Create Table Examples

#### 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://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://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://bucket/folder/partition_cow/one/'
  PARTITION (event_type = 'two') LOCATION 's3://bucket/folder/partition_cow/two/'
```

## Merge on Read (MoR) Create Table Examples

Hudi creates two tables in the Hive metastore for MoR: a table with the name that you specified, which is a read-optimized view, and a table with the same name appended with `_rt`, which is a real-time view. However, when you create MoR tables in Athena, you can query only the read-optimized view.

### Nonpartitioned Merge on Read (MoR) Table

The following example creates a nonpartitioned MoR table in Athena.

```

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://bucket/folder/nonpartition_mor'
```

### Partitioned Merge on Read (MoR) Table

The following example creates a partitioned MoR table in Athena.

```

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://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://bucket/folder/partition_mor/one/'  
  PARTITION (event_type = 'two') LOCATION 's3://bucket/folder/partition_mor/two/'
```

## Querying JSON

Amazon Athena lets you parse JSON-encoded values, extract data from JSON, search for values, and find length and size of JSON arrays.

### Topics

- [Best Practices for Reading JSON Data \(p. 182\)](#)
- [Extracting Data from JSON \(p. 184\)](#)
- [Searching for Values in JSON Arrays \(p. 186\)](#)
- [Obtaining Length and Size of JSON Arrays \(p. 187\)](#)
- [Troubleshooting JSON Queries \(p. 188\)](#)

## 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 \(p. 374\)](#).

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 \(p. 374\)](#).
- Make sure that each JSON-encoded record is represented on a separate line.
- 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://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 \(p. 374\)](#).

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.

- [Converting Athena Data Types to JSON \(p. 183\)](#)
- [Converting JSON to Athena Data Types \(p. 183\)](#)

## Converting 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} |
+-----+
```

## Converting 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} |
+-----+
```

## Extracting Data from JSON

You may have source data with 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: extracting 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 blob
)
SELECT
  json_extract(blob, '$.name') AS name,
  json_extract(blob, '$.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` function. It is similar to `json_extract`, but returns only scalar values (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 blob
)
SELECT
  json_extract_scalar(blob, '$.name') AS name,
  json_extract_scalar(blob, '$.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 blob
)
SELECT json_array_get(json_extract(blob, '$.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 [Accessing Array Elements \(p. 154\)](#).

```
WITH dataset AS (
  SELECT '{"name": "Bob Smith",
         "org": "engineering",
```

```
        "projects": [{"name": "project1", "completed": false}, {"name": "project2",  
"completed": true}]]}'  
        AS blob  
    )  
    SELECT json_extract_scalar(blob, '$.projects[0].name') AS project_name  
    FROM dataset
```

It returns this result:

```
+-----+  
| project_name |  
+-----+  
| project1     |  
+-----+
```

## Searching 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                  |
+-----+
```

## Obtaining Length and Size of JSON Arrays

### 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     |

## Troubleshooting JSON Queries

For help on troubleshooting issues with JSON-related queries, 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?](#)

## Using Machine Learning (ML) with Amazon Athena (Preview)

Machine Learning (ML) with Amazon Athena (Preview) lets you use Athena to write SQL statements that run Machine Learning (ML) inference using Amazon SageMaker. 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 (Preview), you define an ML with Athena (Preview) function with the `USING FUNCTION` clause. The function points to the SageMaker 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 and how SageMaker endpoints work, see the [Amazon SageMaker Developer Guide](#).

### Considerations and Limitations

- **Available Regions** – The Athena ML feature is available in preview in the US East (N. Virginia), Asia Pacific (Mumbai), Europe (Ireland), and US West (Oregon) Regions.
- **AmazonAthenaPreviewFunctionality workgroup** – To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups \(p. 331\)](#).
- **SageMaker model endpoint must accept and return text/csv** – For more information about data formats, see [Common Data Formats for Inference](#) in the *Amazon SageMaker Developer Guide*.
- **SageMaker endpoint scaling** – Make sure that the referenced SageMaker model endpoint is sufficiently scaled up for Athena calls to the endpoint. For more information, see [Automatically Scale SageMaker Models](#) in the *Amazon SageMaker Developer Guide* and [CreateEndpointConfig](#) in the *Amazon SageMaker API Reference*.
- **IAM permissions** – To run a query that specifies an ML with Athena (Preview) function, the IAM principal running the query must be allowed to perform the `sagemaker:InvokeEndpoint` action for the referenced SageMaker model endpoint. For more information, see [Allowing Access for ML with Athena \(Preview\) \(p. 268\)](#).
- **ML with Athena (Preview) functions cannot be used in `GROUP BY` clauses directly**

### ML with Athena (Preview) Syntax

The `USING FUNCTION` clause specifies an ML with Athena (Preview) 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 example illustrates a `USING FUNCTION` clause that specifies ML with Athena (Preview) function.

```
USING FUNCTION ML_function_name(variable1 data_type[, variable2 data_type][,...])  
RETURNS data_type TYPE SAGEMAKER_INVOKE_ENDPOINT WITH (sagemaker_endpoint=  
'my_sagemaker_endpoint')[, FUNCTION...][, ...] SELECT [...] ML_function_name(expression)  
[...]
```

## Parameters

**USING 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 with its corresponding data type, which the SageMaker model can accept as input. Specify *data\_type* as one of the supported Athena data types that the SageMaker model can accept as input.

**RETURNS** *data\_type* **TYPE**

*data\_type* specifies the SQL data type that *ML\_function\_name* returns to the query as output from the SageMaker model.

**SAGEMAKER\_INVOKE\_ENDPOINT WITH** (*sagemaker\_endpoint*= '*my\_sagemaker\_endpoint*')

*my\_sagemaker\_endpoint* specifies the endpoint of the SageMaker model.

**SELECT** [...] *ML\_function\_name*(*expression*) [...]

The SELECT query that passes values to function variables and the SageMaker 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 FUNCTION clause.

## Examples

The following example demonstrates a query using ML with Athena (Preview).

### Example

```
USING FUNCTION predict_customer_registration(age INTEGER)
  RETURNS DOUBLE TYPE
  SAGEMAKER_INVOKE_ENDPOINT WITH (sagemaker_endpoint =
    '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;
```

## Querying with User Defined Functions (Preview)

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 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 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. We provide example Java source code and packages to get you started. 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](#).

## Considerations and Limitations

- **Available Regions** – The Athena UDF feature is available in preview in the US East (N. Virginia), Asia Pacific (Mumbai), Europe (Ireland), and US West (Oregon) Regions.
- **AmazonAthenaPreviewFunctionality workgroup** – To use this feature in preview, you must create an Athena workgroup named `AmazonAthenaPreviewFunctionality` and join that workgroup. For more information, see [Managing Workgroups \(p. 331\)](#).
- **Built-in Athena functions** – Built-in Presto 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 [Presto Functions in Amazon Athena \(p. 399\)](#).
- **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 that this processing design can have.
- **Java runtime only** – Currently, Athena UDFs support only the Java 8 runtime for Lambda.
- **IAM permissions** – To run a query in Athena that contains a UDF query statement and to create UDF statements, the IAM principal running the query must be allowed to perform actions in addition to Athena functions. For more information, see [Example IAM Permissions Policies to Allow Amazon Athena User Defined Functions \(UDF\) \(p. 264\)](#).
- **Lambda quotas** – Lambda quotas apply to UDFs. For more information, see [AWS Lambda Quotas](#) in the *AWS Lambda Developer Guide*.
- **Known issues** – For the most up-to-date list of known issues, see [Limitations and Issues](#) in the Athena Federated Query (Preview)

## UDF Query Syntax

The `USING 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.

### Synopsis

```
USING FUNCTION UDF_name(variable1 data_type[, variable2 data_type][,...]) RETURNS data_type
TYPE
    LAMBDA_INVOKE WITH (lambda_name = 'my_lambda_function')[, FUNCTION][[, ...] SELECT
    [...] UDF_name(expression) [...]
```

### Parameters

**USING 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 with its corresponding data type, which the UDF can accept as input. Specify *data\_type* as one of the supported Athena data types listed in the following table. The data type must map to the corresponding Java data type.

| Athena data type | Java data type                |
|------------------|-------------------------------|
| TIMESTAMP        | java.time.LocalDateTime (UTC) |
| DATE             | java.time.LocalDate (UTC)     |

| Athena data type | Java data type                |
|------------------|-------------------------------|
| TINYINT          | java.lang.Byte                |
| SMALLINT         | java.lang.Short               |
| REAL             | java.lang.Float               |
| DOUBLE           | java.lang.Double              |
| DECIMAL          | java.math.BigDecimal          |
| BIGINT           | java.lang.Long                |
| INTEGER          | java.lang.Int                 |
| VARCHAR          | java.lang.String              |
| VARBINARY        | byte[]                        |
| BOOLEAN          | java.lang.Boolean             |
| ARRAY            | java.util.List                |
| ROW              | java.util.Map<String, Object> |

### RETURNS data\_type TYPE

data\_type specifies the SQL data type that the UDF returns as output. Athena data types listed in the table above are supported.

**LAMBDA\_INVOKE WITH (lambda\_name = 'my\_lambda\_function')**

my\_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 FUNCTION clause.

## Examples

The following examples demonstrate queries using UDFs. The Athena query examples are based on the [AthenaUDFHandler.java](#) code in GitHub.

### Example – Compress and Decompress a String

#### Athena SQL

The following example demonstrates using the compress UDF defined in a Lambda function named MyAthenaUDFLambda.

```

USING FUNCTION compress(col1 VARCHAR)
  RETURNS VARCHAR TYPE
  LAMBDA_INVOKE WITH (lambda_name = 'MyAthenaUDFLambda')
SELECT
  compress('StringToBeCompressed');
```

The query result returns `ewLLinKzEsPyXdKdc7PLShKLS50TQEAUrEH9w==`.

The following example demonstrates using the `decompress` UDF defined in the same Lambda function.

```
USING FUNCTION decompress(col1 VARCHAR)
  RETURNS VARCHAR TYPE
  LAMBDA_INVOKE WITH (lambda_name = 'MyAthenaUDFLambda')
SELECT
  decompress('ewLLinKzEsPyXdKdc7PLShKLS50TQEAUrEH9w==');
```

The query result returns `StringToBeCompressed`.

## Creating and Deploying 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.

### Steps to Create a Custom UDF for Athena Using Maven

- [Clone the SDK and Prepare Your Development Environment \(p. 193\)](#)
- [Create your Maven Project \(p. 194\)](#)
- [Add Dependencies and Plugins to Your Maven Project \(p. 194\)](#)
- [Write Java Code for the UDFs \(p. 195\)](#)
- [Build the JAR File \(p. 196\)](#)
- [Deploy the JAR to AWS Lambda \(p. 196\)](#)

## 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 [Using Amazon Athena Federated Query \(Preview\) \(p. 56\)](#).

```
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>2019.48.1</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>
```

```
</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: Deploying 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://mybucket/mysarapps/athenaudf` and your YAML file was saved as `my-athena-udfs.yaml`:

```
./publish.sh mybucket/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: Creating 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
```

# Querying 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 that connect via JDBC. For more information, see [Using Athena with the JDBC Driver \(p. 72\)](#), the [AWS CLI](#), or the [Amazon Athena API Reference](#).

The topics in this section assume that you have set up both an IAM user with appropriate permissions to access Athena and the Amazon S3 bucket where the data to query should reside. For more information, see [Setting Up \(p. 6\)](#) and [Getting Started \(p. 8\)](#).

### Topics

- [Querying Application Load Balancer Logs \(p. 198\)](#)
- [Querying Classic Load Balancer Logs \(p. 200\)](#)
- [Querying Amazon CloudFront Logs \(p. 202\)](#)
- [Querying AWS CloudTrail Logs \(p. 203\)](#)
- [Querying Amazon EMR Logs \(p. 208\)](#)
- [Querying AWS Global Accelerator Flow Logs \(p. 211\)](#)
- [Querying Amazon GuardDuty Findings \(p. 213\)](#)
- [Querying Network Load Balancer Logs \(p. 214\)](#)
- [Querying Amazon VPC Flow Logs \(p. 216\)](#)
- [Querying AWS WAF Logs \(p. 218\)](#)

## Querying 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 the [User Guide for Application Load Balancers](#).

### Topics

- Prerequisites (p. 199)
- Creating the Table for ALB Logs (p. 199)
- Example Queries for ALB Logs (p. 200)

## Prerequisites

- **Enable access logging** so that Application Load Balancer logs can be saved to your Amazon S3 bucket.

## Creating the Table for ALB Logs

1. Copy and paste the following `CREATE TABLE` statement into the Athena console. Replace the values in `LOCATION 's3://your-alb-logs-directory/AWSLogs/<ACCOUNT-ID>/elasticloadbalancing/<REGION>/'` with those corresponding to your Amazon S3 bucket location. For information about each field, see [Access Log Entries](#) in the *User Guide for Application Load Balancers*.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_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 string,
    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
)
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://your-alb-logs-directory/AWSLogs/<ACCOUNT-ID>/  
elasticloadbalancing/<REGION>/';
```

2. Run the query in the Athena console. After the query completes, Athena registers the `alb_logs` table, making the data in it ready for you to issue queries.

## Example Queries for ALB 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_logs  
GROUP BY request_verb, client_ip  
LIMIT 100;
```

Another query shows the URLs visited by Safari browser users:

```
SELECT request_url  
FROM alb_logs  
WHERE user_agent LIKE '%Safari%'  
LIMIT 10;
```

The following example shows how to parse the logs by datetime:

```
SELECT client_ip, sum(received_bytes)  
FROM alb_logs_config_us  
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;
```

## Querying 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](#).

- [Create the table for Elastic Load Balancing logs \(p. 200\)](#)
- [Elastic Load Balancing Example Queries \(p. 201\)](#)

## 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://your_log_bucket/prefix/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 [Elastic Load Balancing Example Queries \(p. 201\)](#)

## Elastic Load Balancing 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](#).

## Querying 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 to which to save these logs.

### Note

This procedure works for the Web distribution access logs in CloudFront. It does not apply to streaming logs from RTMP distributions.

- [Creating the Table for CloudFront Logs \(p. 202\)](#)
- [Example Query for CloudFront logs \(p. 203\)](#)

## Creating the Table for CloudFront Logs

### To create the CloudFront table

1. Copy and paste the following DDL statement into the Athena console. Modify the `LOCATION` for the Amazon S3 bucket that stores your logs.

This query uses the [LazySimpleSerDe \(p. 378\)](#) by default and it is omitted.

The column `date` is escaped using backticks ( ``` ) because it is a reserved word in Athena. For information, see [Reserved Keywords \(p. 85\)](#).

```
CREATE EXTERNAL TABLE IF NOT EXISTS default.cloudfront_logs (  
  `date` DATE,  
  time STRING,  
  location STRING,  
  bytes BIGINT,  
  request_ip STRING,  
  method STRING,  
  host STRING,  
  uri STRING,  
  status INT,  
  referrer STRING,  
  user_agent STRING,  
  query_string STRING,  
  cookie STRING,  
  result_type STRING,  
  request_id STRING,  
  host_header STRING,  
  request_protocol STRING,  
  request_bytes BIGINT,  
  time_taken FLOAT,  
  xforwarded_for STRING,  
  ssl_protocol STRING,  
  ssl_cipher STRING,  
  response_result_type STRING,  
  http_version STRING,  
  file_status STRING,  
  file_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://CloudFront_bucket_name/CloudFront/'  
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Run the query in Athena console. After the query completes, Athena registers the `cloudfront_logs` table, making the data in it ready for you to issue queries.

## Example Query for CloudFront Logs

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_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_logs  
LIMIT 10;
```

## 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 Kinesis Analytics](#) (May 26, 2017).

## Querying AWS CloudTrail Logs

AWS CloudTrail is a service that records AWS API calls and events for AWS 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](#).

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

- [Understanding CloudTrail Logs and Athena Tables](#) (p. 204)
- [Using the CloudTrail Console to Create an Athena Table for CloudTrail Logs](#) (p. 205)
- [Manually Creating the Table for CloudTrail Logs in Athena](#) (p. 206)
- [Example Query for CloudTrail Logs](#) (p. 207)
- [Tips for Querying CloudTrail Logs](#) (p. 207)

## Understanding 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.gzip). 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 for the console. For more information, see [Creating a Trail](#) in the *AWS CloudTrail User Guide*.

Note the destination Amazon S3 bucket where you save the logs. Replace the `LOCATION` clause with the path to the CloudTrail log location and the set of objects with which to work. 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://MyLogFiles/AWSLogs/`.
- To analyze data from a specific date, account, and Region, use `LOCATION 's3://MyLogFiles/123456789012/CloudTrail/us-east-1/2016/03/14/'`.

Using the highest level in the object hierarchy gives you the greatest flexibility when you query using Athena.

## Using the CloudTrail Console to Create an Athena Table for CloudTrail Logs

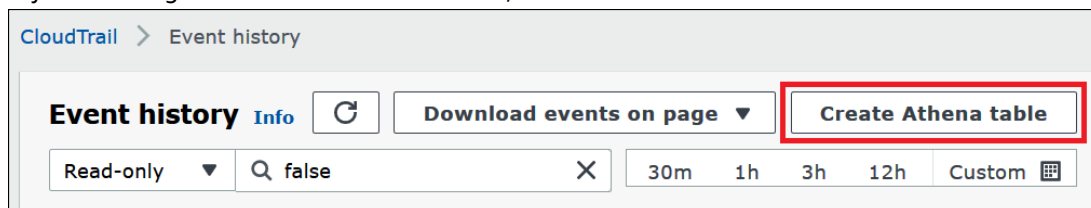
You can create a non-partitioned Athena table for querying CloudTrail logs directly from the CloudTrail console. Creating an Athena table from the CloudTrail console requires that you be logged in with an IAM user or role that has sufficient permissions to create tables in Athena.

- For information about setting up permissions for Athena, see [Setting Up \(p. 6\)](#).
- For information about creating a table with partitions, see [Manually Creating the Table for CloudTrail Logs in Athena](#).

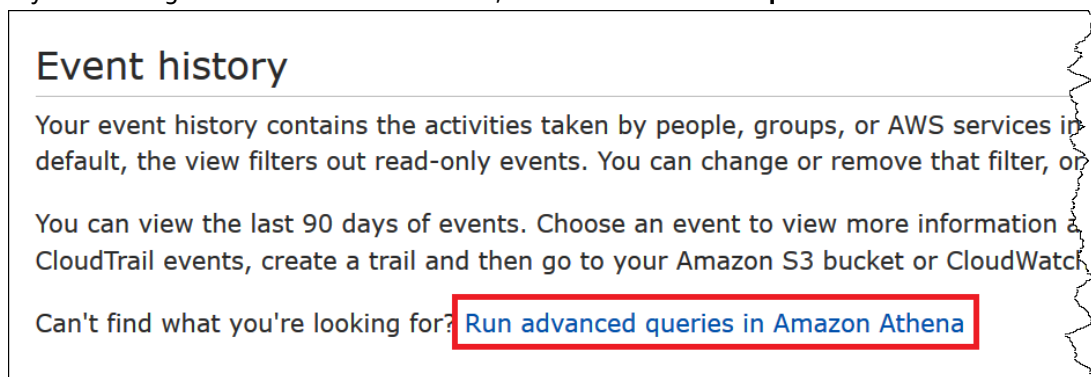
### To create an Athena table for a CloudTrail trail using the CloudTrail console

1. Open the CloudTrail console at <https://console.aws.amazon.com/cloudtrail/>.
2. In the navigation pane, choose **Event history**.
3. Do one of the following:

- If you are using the newer CloudTrail console, choose **Create Athena table**.



- If you are using the older CloudTrail console, choose **Run advanced queries in Amazon Athena**.



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.

## Manually Creating the Table for CloudTrail Logs in Athena

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. The statement is the same as the one in the CloudTrail console **Create a table in Amazon Athena** dialog box, but adds a `PARTITIONED BY` clause that makes the table partitioned.
2. Modify `s3://CloudTrail_bucket_name/AWSLogs/Account_ID/CloudTrail/` to point to the Amazon S3 bucket that contains your log data.
3. Verify that fields are listed correctly. For more information about the full list of fields in a CloudTrail record, see [CloudTrail Record Contents](#).

In this 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 "Extracting Data from JSON" \(p. 184\)](#). For performance improvements, this example partitions the data by Region, year, month, and day.

```
CREATE EXTERNAL TABLE cloudtrail_logs (  
  eventversion STRING,  
  useridentity STRUCT<  
    type:STRING,  
    principalid:STRING,  
    arn:STRING,  
    accountid:STRING,  
    invokedby:STRING,  
    accesskeyid:STRING,  
    userName:STRING,  
  sessioncontext:STRUCT<  
  attributes:STRUCT<  
    mfaauthenticated:STRING,  
    creationdate:STRING>,  
  sessionissuer:STRUCT<  
    type:STRING,  
    principalid:STRING,  
    arn:STRING,  
    accountId:STRING,  
    userName: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  
)  
PARTITIONED BY (region string, year string, month string, day string)  
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'  
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  
LOCATION 's3://CloudTrail_bucket_name/AWSLogs/Account_ID/CloudTrail/';
```

4. Run the query in the Athena console.
5. Use the [ALTER TABLE ADD PARTITION \(p. 402\)](#) 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://CloudTrail_bucket_name/AWSLogs/Account_ID/CloudTrail/us-  
east-1/2019/02/01/'
```

## Example Query for CloudTrail Logs

The following example shows a portion of a query that returns all anonymous (unsigned ) requests from the table created on top of 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 LIKE '%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](#).

## Tips for Querying CloudTrail Logs

To explore the CloudTrail logs data, use these tips:

- Before querying the logs, verify that your logs table looks the same as the one in [the section called “Manually Creating the Table for CloudTrail Logs in Athena” \(p. 206\)](#). 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 [Creating the Table for CloudTrail Logs \(p. 206\)](#).

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 [Extracting Data From JSON \(p. 184\)](#).

Now you are ready to issue queries against your CloudTrail table.

- Start by looking at which IAM 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 earlier query to further explore your data.
- To improve performance, include the `LIMIT` clause to return a specified subset of rows.

## Querying Amazon EMR Logs

Amazon EMR and big data applications that run on Amazon EMR produce log files. Logs files are written to the master 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*.

## Creating and Querying 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 AWS account `123456789012` in Region `us-west-2`. If you use a custom location, the pattern is `s3://PathToEMRLogs/ClusterID`.

For information about creating a partitioned table to potentially improve query performance and reduce data transfer, see [Creating and Querying a Partitioned Table Based on Amazon EMR Logs](#) (p. 209).

```
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'
```

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;
```

## Creating and Querying 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 [Partitioning Data \(p. 92\)](#).

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;
```

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
```

## Querying 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* (p. 362), which means that the query uses the *LazySimpleSerDe* (p. 378). 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  
)  
PARTITIONED BY (dt string)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ' '  
LOCATION 's3://your_log_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://your_log_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://your_log_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;
```

## Querying 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.

## Creating 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://*findings-bucket-name*/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,  
  `createdate` string,  
  `updatedate` string,  
  `title` string,  
  `description` string)  
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  
LOCATION 's3://findings-bucket-name/AWSLogs/account-id/GuardDuty/'  
TBLPROPERTIES ('has_encrypted_data'='true')
```

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,
    createdate,
    updatedate,
    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,
    createdate,
    updatedate,
    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 [Extracting Data from JSON \(p. 184\)](#).
- Make sure that all characters in the JSON fields are in lower case.
- For information about downloading query results, see [Downloading Query Results Files Using the Athena Console \(p. 114\)](#).

## Querying 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, see [Access Logs for Your Network Load Balancer](#).

- [Create the table for Network Load Balancer logs \(p. 215\)](#)
- [Network Load Balancer Example Queries \(p. 215\)](#)

## 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. 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 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  
)  
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://your_log_bucket/prefix/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.

## Network Load Balancer 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 the older TLS version:

```
SELECT tls_protocol_version,  
       COUNT(tls_protocol_version) AS  
         num_connections,  
       client_ip  
FROM "nlb_tls_logs"  
WHERE tls_protocol_version < 'tls12'  
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;
```

## Querying 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.

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 the default format, or you can specify a custom format. A custom format is where you specify which fields to return in the flow log, and the order in which they should appear. For more information, see [Flow Log Records](#) in the *Amazon VPC User Guide*.

- [Creating the Table for VPC Flow Logs \(p. 216\)](#)
- [Example Queries for Amazon VPC Flow Logs \(p. 217\)](#)

## Creating the Table for VPC Flow Logs

The following procedure creates an Amazon VPC table for VPC flow logs that use the default format. If you create a flow log with a custom format, you must 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 the Amazon VPC table

1. Copy and paste the following DDL statement into the Athena console Query Editor:

```
CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs (  
  version int,  
  account string,  
  interfaceid string,  
  sourceaddress string,  
  destinationaddress string,  
  sourceport int,  
  destinationport int,  
  protocol int,
```

```
numpackets int,  
numbytes bigint,  
starttime int,  
endtime int,  
action string,  
logstatus string  
)  
PARTITIONED BY (`date` date)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ' '  
LOCATION 's3://your_log_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 [LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files \(p. 378\)](#). 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 backtick characters. For more information, see [Reserved Keywords \(p. 85\)](#).

- For a VPC flow log with a custom format, modify the fields to match the fields that you specified when you created the flow log.
2. Modify the `LOCATION` `'s3://your_log_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.

```
ALTER TABLE vpc_flow_logs  
ADD PARTITION (`date`='YYYY-MM-dd')  
location 's3://your_log_bucket/prefix/AWSLogs/{account_id}/  
vpcflowlogs/{region_code}/{YYYY/MM/dd};
```

## Example Queries for Amazon VPC Flow Logs

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,  
interfaceid,  
sourceaddress,  
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 this 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(numpackets) AS  
    packetcount,  
    destinationaddress  
FROM vpc_flow_logs  
WHERE destinationport = 443 AND date > current_date - interval '7' day  
GROUP BY destinationaddress  
ORDER BY packetcount DESC  
LIMIT 10;
```

For more information, see the AWS Big Data blog post [Analyzing VPC Flow Logs with Amazon Kinesis Firehose, Athena, and Amazon QuickSight](#).

## Querying AWS WAF Logs

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 enable access logging for AWS WAF logs, save them to Amazon S3, and query the logs in Athena. For more information about enabling AWS WAF logs and about the log record structure, see [Logging Web ACL Traffic Information](#) in the *AWS WAF Developer Guide*.

Make a note of the Amazon S3 bucket to which you save these logs.

- [Creating the Table for AWS WAF Logs \(p. 218\)](#)
- [Example Queries for AWS WAF logs \(p. 219\)](#)

## Creating the Table for AWS WAF Logs

### To create the AWS WAF table

1. Copy and paste the following DDL statement into the Athena console. Modify the `LOCATION` for the Amazon S3 bucket that stores your logs.

This query uses the [OpenX JSON SerDe \(p. 375\)](#). The table format and the SerDe are suggested by the AWS Glue crawler when it analyzes AWS WAF logs.

#### Note

The SerDe expects each JSON record in the WAF logs in Amazon S3 to be on a single line of text with no line termination characters separating the fields in the record. If the WAF log JSON text is in pretty print format, you may receive the error message `HIVE_CURSOR_ERROR: Row is not a valid JSON Object` when you attempt to query the table after you create it.

```
CREATE EXTERNAL TABLE `waf_logs` (
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails` array<
    struct<
      conditiontype:string,
      location:string,
      matcheddata:array<string>
    >
  >,
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist` array<string>,
  `ratebasedrulelist` array<
    struct<
      ratebasedruleid:string,
      limitkey:string,
      maxrateallowed:int
    >
  >,
  `nonterminatingmatchingrules` array<
    struct<
      ruleid:string,
      action:string
    >
  >,
  `httprequest` struct<
    clientip:string,
    country:string,
    headers:array<
      struct<
        name:string,
        value:string
      >
    >,
    uri:string,
    args:string,
    httpversion:string,
    httpmethod:string,
    requestid:string
  >
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'paths'='action,formatVersion,httpRequest,httpSourceId,httpSourceName,nonTerminatingMatchingRules,r
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://athenawaflogs/WebACL/'
```

2. Run the query in the Athena console. After the query completes, Athena registers the `waf_logs` table, making the data in it available for queries.

## Example Queries for AWS WAF Logs

The following query counts the number of times an IP address has been blocked by the `RATE_BASED` terminating rule.

```
SELECT COUNT(httpRequest.clientIp) as count,
```



```
httpRequest.clientIp
FROM waf_logs
WHERE terminatingruletype='RATE_BASED' AND action='BLOCK'
GROUP BY httpRequest.clientIp
ORDER BY count
LIMIT 100;
```

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;
```

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;
```

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;
```

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 Developer Guide
- [Using AWS CloudTrail to identify Amazon S3 requests](#) in the Amazon Simple Storage Service Developer Guide

# Querying 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.

## Note

You can use individual hive [DDL commands \(p. 399\)](#) to extract metadata information for specific databases, tables, views, partitions, and columns from Athena, but the output is in a non-tabular format.

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.

## Important

You cannot use `CREATE VIEW` to create a view on the `information_schema` database.

## Topics

- [Listing Databases and Searching a Specified Database \(p. 221\)](#)
- [Listing Tables in a Specified Database and Searching for a Table by Name \(p. 222\)](#)
- [Listing Partitions for a Specific Table \(p. 222\)](#)
- [Listing or Searching Columns for a Specified Table or View \(p. 223\)](#)

## Listing 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

## Listing 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_accoun	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

## Listing Partitions for a Specific Table

You can use a metadata query to list the partition numbers and partition values for a specific table.

### Example – Querying the Partitions for a Table

The following example query lists the partitions for the table `CloudTrail_logs_test2`.

```
SELECT *
FROM   information_schema.__internal_partitions__
WHERE  table_schema = 'default'
      AND table_name = 'cloudtrail_logs_test2'
ORDER BY partition_number
```

If the query does not work as expected, use `SHOW PARTITIONS table_name` to extract the partition details for a specified table, as in the following example.

```
SHOW PARTITIONS CloudTrail_logs_test2
```

The following table shows sample results.

	table_catalog	table_schema	table_name	partition_num	partition_key	partition_value
1	awsdatacatalog	default	CloudTrail_logs_test2	1	year	2018
2	awsdatacatalog	default	CloudTrail_logs_test2	1	month	09
3	awsdatacatalog	default	CloudTrail_logs_test2	1	day	30

## Listing or Searching 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.

### 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_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
1	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	id	1		YES	varchar		
2	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	age	2		YES	integer		
3	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	email	3		YES	timestamp		
4	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	last_login	4		YES	timestamp		
5	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	password	5		YES	varchar		
6	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	username	6		YES	varchar		

### 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	arrayview	searchdate	1		YES	varchar		
2	awsdatacatalog	default	arrayview	sid	2		YES	varchar		
3	awsdatacatalog	default	arrayview	btid	3		YES	varchar		
4	awsdatacatalog	default	arrayview	ip	4		YES	varchar		
5	awsdatacatalog	default	arrayview	infantprice	5		YES	varchar		
6	awsdatacatalog	default	arrayview	sump	6		YES	varchar		
7	awsdatacatalog	default	arrayview	journemaparray	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	varchar		

## Querying 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

- [Querying Apache Logs Stored in Amazon S3 \(p. 225\)](#)
- [Querying Internet Information Server \(IIS\) Logs Stored in Amazon S3 \(p. 226\)](#)

## Querying 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
```

## Creating 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 \(p. 372\)](#). 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://*bucket-name*/*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://bucket-name/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.

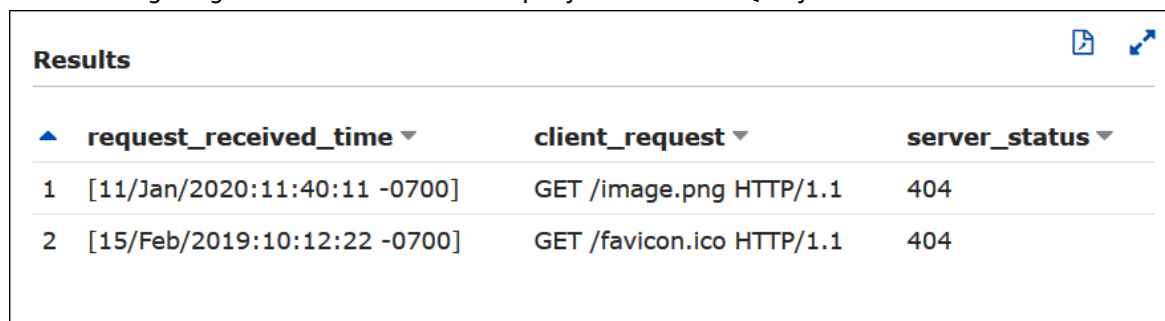
## Example Select Queries for Apache Logs

### 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 `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.



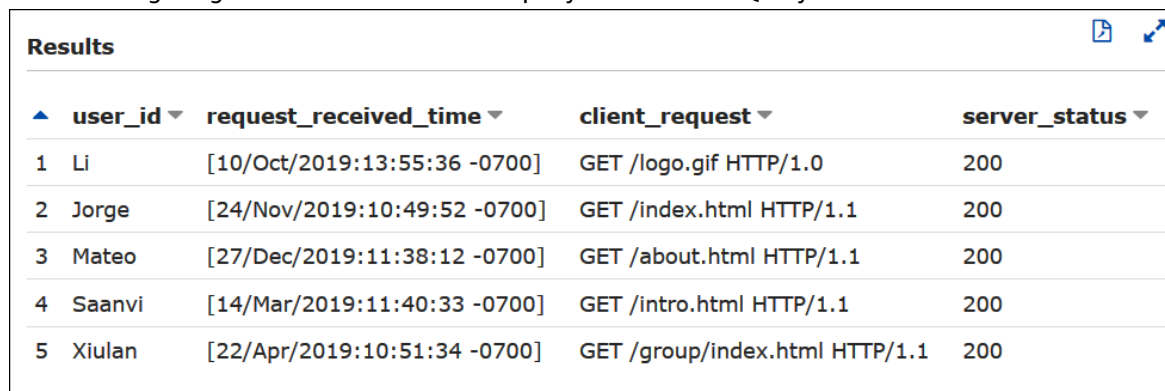
Results			
	request_received_time ▼	client_request ▼	server_status ▼
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

### Example – Filtering 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

## Querying 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](#) (p. 378) to create Athena tables for them.

## 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
```

## Creating 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](#) (p. 84), 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://bucket-name/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://bucket-name/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

## 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, -,
```

## Creating 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://bucket-name/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://bucket-name/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

## 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
```

## Creating a Table in Athena for IIS NCSA Logs

For your `CREATE TABLE` statement, you can use the [Grok SerDe](#) (p. 372) and a grok pattern similar to the one for [Apache web server logs](#) (p. 225). 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://*bucket-name*/*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://bucket-name/iis-ncsa-logs/';
```

- 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	

# 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 \(p. 232\)](#)
- [Identity and Access Management in Athena \(p. 239\)](#)
- [Logging and Monitoring in Athena \(p. 271\)](#)
- [Compliance Validation for Amazon Athena \(p. 272\)](#)
- [Resilience in Athena \(p. 272\)](#)
- [Infrastructure Security in Athena \(p. 273\)](#)
- [Configuration and Vulnerability Analysis in Athena \(p. 274\)](#)
- [Using Athena to Query Data Registered With AWS Lake Formation \(p. 275\)](#)

## Data Protection in Athena

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 does not modify it. For more information, see [Data Protection in Amazon S3](#) in the *Amazon Simple Storage Service Developer 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 \(p. 236\)](#).
- **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.

### Note

The metadata for databases and tables you use in Athena is stored in the AWS Glue Data Catalog. We highly recommend that you [upgrade \(p. 29\)](#) to using the AWS Glue Data

Catalog with Athena. For more information about the benefits of using the AWS Glue Data Catalog, see [FAQ: Upgrading to the AWS Glue Data Catalog \(p. 32\)](#).

You can [define fine-grained access policies to databases and tables \(p. 244\)](#) 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 \(p. 235\)](#) 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 [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#) in this document.

Athena retains query history for 45 days. You can [view query history \(p. 117\)](#) 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 \(p. 322\)](#) in Athena, restricting access to saved queries only to users who are authorized to view them.

#### Topics

- [Encryption at Rest \(p. 233\)](#)
- [Encryption in Transit \(p. 238\)](#)
- [Key Management \(p. 238\)](#)
- [Internetwork Traffic Privacy \(p. 239\)](#)

## Encryption at Rest

You can run queries in Amazon Athena on encrypted data in Amazon S3 in the same Region. 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 [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#).
- The data in the AWS Glue Data Catalog. For information, see [Permissions to Encrypted Metadata in the AWS Glue Data Catalog \(p. 235\)](#).

#### Note

The setup for querying an encrypted dataset in Amazon S3 and the options in Athena to encrypt query results are independent. Each option is enabled and configured separately. You can use different encryption methods or keys for each. This means that reading encrypted data in Amazon S3 doesn't automatically encrypt Athena query results in Amazon S3. The opposite is also true. Encrypting Athena query results in Amazon S3 doesn't encrypt the underlying dataset in Amazon S3.

#### Topics

- [Supported Amazon S3 Encryption Options \(p. 234\)](#)
- [Permissions to Encrypted Data in Amazon S3 \(p. 234\)](#)
- [Permissions to Encrypted Metadata in the AWS Glue Data Catalog \(p. 235\)](#)
- [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#)

- [Creating Tables Based on Encrypted Datasets in Amazon S3 \(p. 236\)](#)

## 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
<a href="#">SSE-S3</a>	Server side encryption (SSE) with an Amazon S3-managed key.	Yes
<a href="#">SSE-KMS</a>	Server-side encryption (SSE) with a AWS Key Management Service customer managed key.  <b>Note</b> With this encryption type, Athena does not require you to indicate that data is encrypted when you create a table.	Yes
<a href="#">CSE-KMS</a>	Client-side encryption (CSE) with a AWS KMS customer managed key. In Athena, this option requires that you use a <code>CREATE TABLE</code> statement with a <code>TBLPROPERTIES</code> clause that specifies <code>'has_encrypted_data'='true'</code> . For more information, see <a href="#">Creating Tables Based on Encrypted Datasets in Amazon S3 (p. 236)</a> .	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*.

## Unsupported Options

The following encryption options are not supported:

- SSE with customer-provided keys (SSE-C).
- Client-side encryption using a client-side master key.
- Asymmetric keys.

To compare Amazon S3 encryption options, see [Protecting Data Using Encryption](#) in the *Amazon Simple Storage Service Developer Guide*.

## Permissions to Encrypted Data in Amazon S3

Depending on the type of encryption you use in Amazon S3, you may 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 [IAM Policies for User Access \(p. 240\)](#) and [Amazon S3 Permissions \(p. 243\)](#).
- **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 AWS KMS customer managed keys (CMKs) that are used to encrypt data in Amazon S3. The easiest way to do this is to use the IAM console to add key users to the appropriate AWS KMS key policies. For information about how to add a user to a AWS KMS key policy, see [How to Modify a Key Policy](#) 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 may 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. In this case, you can increase your service quotas for AWS KMS. For more information, see [Quotas](#) in the *AWS Key Management Service Developer Guide*.

## 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 [Access to Encrypted Metadata in the AWS Glue Data Catalog \(p. 250\)](#).

## Encrypting Query Results Stored in Amazon S3

You set up query result encryption using the Athena console. Workgroups allow you to enforce the encryption of query results.

If you connect using the 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 the JDBC or ODBC driver to encrypt your query results using any of the encryption protocols that Athena supports, see [Connecting to Amazon Athena with ODBC and JDBC Drivers \(p. 72\)](#).

You can configure the setting for encryption of query results in two ways:

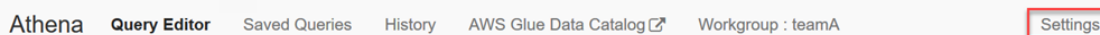
- **Client-side settings** – When you use **Settings** in the console or the API operations to indicate that you want to encrypt query results, this is known as using client-side settings. Client-side settings include query results location and encryption. If you specify them, they are used, unless they are overridden by the workgroup settings.
- **Workgroup settings** – When you [create or edit a workgroup \(p. 332\)](#) and select the **Override client-side settings** field, then all queries that run in this workgroup use the workgroup settings. For more information, see [Workgroup Settings Override Client-Side Settings \(p. 330\)](#). Workgroup settings include query results location and encryption.

### To encrypt query results stored in Amazon S3 using the console

#### Important

If your workgroup has the **Override client-side settings** field selected, then the queries use the workgroup settings. The encryption configuration and the query results location listed in **Settings**, the API operations, and the drivers are not used. For more information, see [Workgroup Settings Override Client-Side Settings \(p. 330\)](#).

1. In the Athena console, choose **Settings**.





2. For **Query result location**, enter a custom value or leave the default. This is the Amazon S3 staging directory where query results are stored.
3. Choose **Encrypt query results**.

Settings

Settings apply by default to all new queries. [Learn more](#)

Workgroup: **teamB**

Query result location  ⓘ  
Example: s3://query-results-bucket/folder/

Encrypt query results ☒ ⓘ

Encryption type  ⓘ

Encryption key  ⓘ [Create KMS key](#)

KMS key ARN

Autocomplete ☐ ⓘ

4. For **Encryption type**, choose **CSE-KMS**, **SSE-KMS**, or **SSE-S3**.
5. If you chose **SSE-KMS** or **CSE-KMS**, specify the **Encryption key**.
  - If your account has access to an existing AWS KMS customer managed key (CMK), choose its alias or choose **Enter a KMS key ARN** and then enter an ARN.
  - If your account does not have access to an existing AWS KMS customer managed key (CMK), choose **Create KMS key**, and then open the [AWS KMS console](#). In the navigation pane, choose **AWS managed keys**. 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.

6. Return to the Athena console to specify the key by alias or ARN as described in the previous step.
7. Choose **Save**.

## Creating 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 is not required when using SSE-KMS. For both SSE-S3 and AWS KMS encryption, Athena determines the proper materials to use to decrypt the dataset and create the table, so you don't need to provide key information.

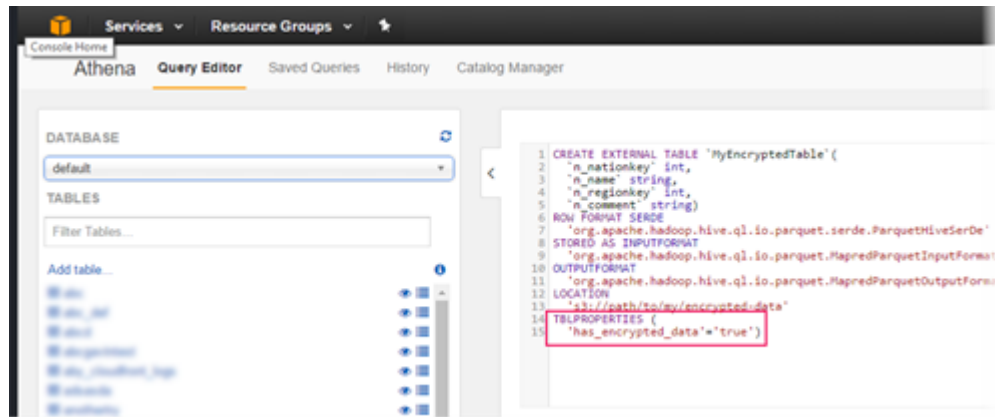
Users that run queries, including the user who creates the table, must have the appropriate permissions as described earlier in this topic.

#### Important

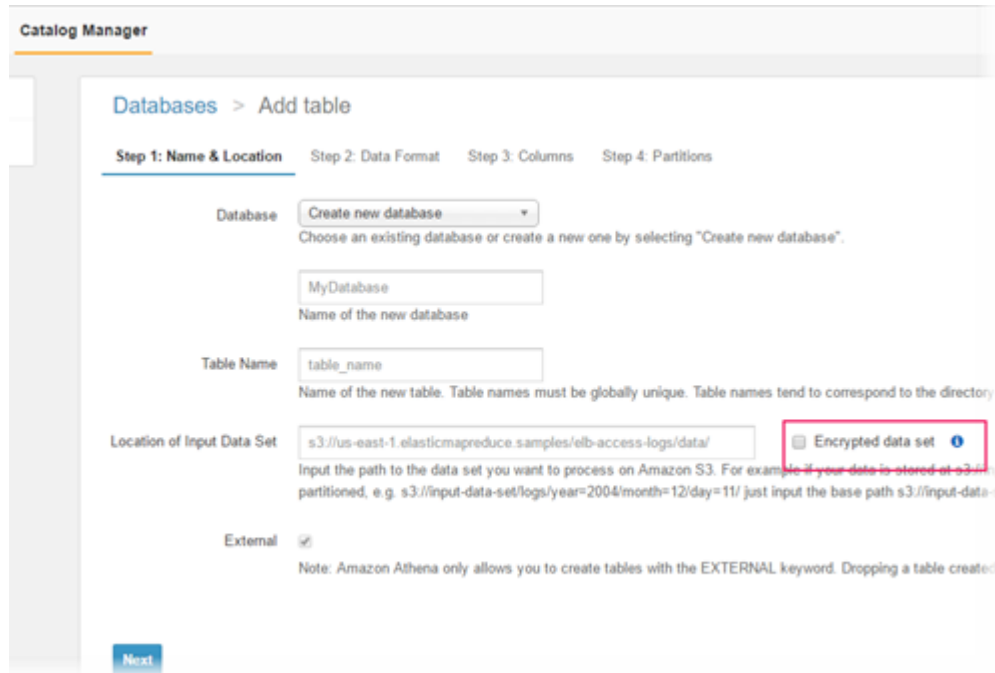
If you use Amazon EMR along with EMRFS to upload encrypted Parquet files, you must disable multipart uploads by setting `fs.s3n.multipart.uploads.enabled` to `false`. If you don't do this, Athena is unable to determine the Parquet file length and a **HIVE\_CANNOT\_OPEN\_SPLIT** error occurs. For more information, see [Configure Multipart Upload for Amazon S3](#) in the *Amazon EMR Management Guide*.

Indicate that the dataset is encrypted in Amazon S3 in one of the following ways. This step is not required if SSE-KMS is used.

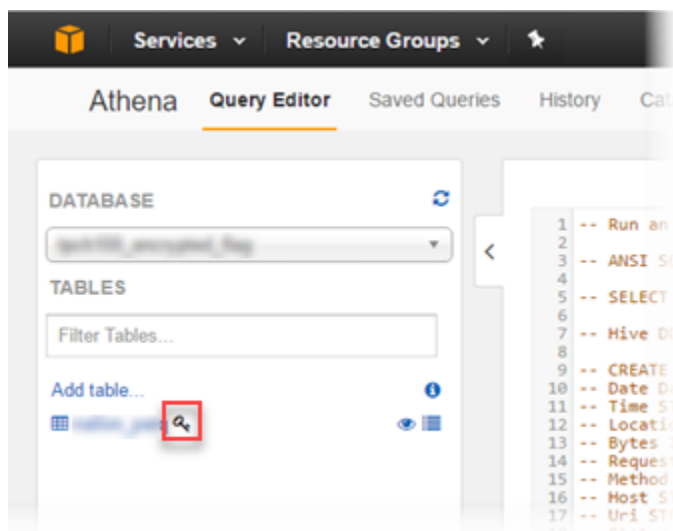
- Use the [CREATE TABLE \(p. 406\)](#) statement with a `TBLPROPERTIES` clause that specifies `'has_encrypted_data'='true'`.



- Use the [JDBC driver](#) (p. 72) and set the TBLPROPERTIES value as shown in the previous example, when you run [CREATE TABLE](#) (p. 406) using `statement.executeQuery()`.
- Use the **Add table** wizard in the Athena console, and then choose **Encrypted data set** when you specify a value for **Location of input data set**.



Tables based on encrypted data in Amazon S3 appear in the **Database** list with an encryption icon.



## 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 with the JDBC Driver](#) (p. 72) and [Connect with the ODBC Driver](#) (p. 73).

## 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 master keys (CMKs) 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 CMKs](#)

### 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 Developer 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.

## Identity and Access Management in Athena

Amazon Athena uses AWS Identity and Access Management (IAM) policies to restrict access to Athena operations.

To run queries in Athena, you must have the appropriate permissions for the following:

- Athena API actions including additional actions for Athena [workgroups](#) (p. 322).
- Amazon S3 locations where the underlying data to query is stored.
- Metadata and resources that you store in the AWS Glue Data Catalog, such as databases and tables, including additional actions for encrypted metadata.

If you are an administrator for other users, make sure that they have appropriate permissions associated with their user profiles. In addition to the following topics, see [Actions, Resources, and Condition Keys for Amazon Athena](#) in the *IAM User Guide*.

### Topics

- [Managed Policies for User Access](#) (p. 240)
- [Access through JDBC and ODBC Connections](#) (p. 243)
- [Access to Amazon S3](#) (p. 243)
- [Fine-Grained Access to Databases and Tables in the AWS Glue Data Catalog](#) (p. 244)
- [Access to Encrypted Metadata in the AWS Glue Data Catalog](#) (p. 250)
- [Cross-account Access in Athena to Amazon S3 Buckets](#) (p. 251)
- [Access to Workgroups and Tags](#) (p. 254)
- [Allow Access to an Athena Data Connector for External Hive Metastore](#) (p. 254)
- [Allow Lambda Function Access to External Hive Metastores](#) (p. 256)
- [Example IAM Permissions Policies to Allow Athena Federated Query \(Preview\)](#) (p. 260)
- [Example IAM Permissions Policies to Allow Amazon Athena User Defined Functions \(UDF\)](#) (p. 264)
- [Allowing Access for ML with Athena \(Preview\)](#) (p. 268)
- [Enabling Federated Access to the Athena API](#) (p. 268)

## Managed Policies for User Access

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 *AWS Identity and Access Management User Guide*. For a list of actions, see the [Amazon Athena API Reference](#).

*Managed* policies are easy to use and are updated automatically with the required actions as the service evolves.

Athena has these managed policies:

- The `AmazonAthenaFullAccess` managed policy grants full access to Athena. Attach it to users and other principals who need full access to Athena. See [AmazonAthenaFullAccess Managed Policy](#) (p. 240).
- The `AWSQuicksightAthenaAccess` managed policy grants access to actions that Amazon QuickSight needs to integrate with Athena. Attach this policy to principals who use Amazon QuickSight in conjunction with Athena. See [AWSQuicksightAthenaAccess Managed Policy](#) (p. 242).

*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 *AWS Identity and Access Management 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 [Service Actions for JDBC Connections](#) (p. 243).

If you use AWS Glue with Athena, and have encrypted the AWS Glue Data Catalog, you must specify additional actions in the identity-based IAM policies for Athena. For more information, see [Access to Encrypted Metadata in the AWS Glue Data Catalog](#) (p. 250).

### Important

If you create and use workgroups, make sure your policies include appropriate access to workgroup actions. For detailed information, see [the section called "IAM Policies for Accessing Workgroups"](#) (p. 325) and [the section called "Workgroup Example Policies"](#) (p. 326).

## AmazonAthenaFullAccess Managed Policy

The `AmazonAthenaFullAccess` managed policy grants full access to Athena.

Managed policy contents change, so the policy shown here may be out-of-date. Check the IAM console for the most up-to-date policy.

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

```

    ],
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateDatabase",
        "glue:DeleteDatabase",
        "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"
      ],
      "Resource": [
        "arn:aws:s3::aws-athena-query-results-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::athena-examples*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:ListAllMyBuckets"
      ],
      "Resource": [
        "*"
      ]
    }
  ],

```

```
{
  "Effect": "Allow",
  "Action": [
    "sns:ListTopics",
    "sns:GetTopicAttributes"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:PutMetricAlarm",
    "cloudwatch:DescribeAlarms",
    "cloudwatch:DeleteAlarms"
  ],
  "Resource": [
    "*"
  ]
}
]
```

## AWSQuicksightAthenaAccess Managed Policy

An additional managed policy, `AWSQuicksightAthenaAccess`, grants access to actions that Amazon QuickSight needs to integrate 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. Attach this policy only to principals who use Amazon QuickSight with Athena.

Managed policy contents change, so the policy shown here may be out-of-date. Check the IAM console for the most up-to-date policy.

```
{
  "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"
      ],
      "Resource": [
        "*"
      ]
    },
    {
```

```
    "Effect": "Allow",
    "Action": [
        "glue:CreateDatabase",
        "glue>DeleteDatabase",
        "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"
    ],
    "Resource": [
        "arn:aws:s3:::aws-athena-query-results-*"
    ]
}
]
```

## 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 [AWSQuicksightAthenaAccess Managed Policy](#) (p. 242).

For information about the latest versions of the JDBC and ODBC drivers and their documentation, see [Using Athena with the JDBC Driver](#) (p. 72) and [Connecting to Amazon Athena with ODBC](#) (p. 73).

## Access to Amazon S3

You can grant access to Amazon S3 locations using identity-based policies, bucket resource policies, or both.

For detailed information and examples about how to grant Amazon S3 access, see the following resources:



- [Example Walkthroughs: Managing Access](#) in the *Amazon Simple Storage Service Developer Guide*.
- [How can I provide cross-account access to objects that are in Amazon S3 buckets?](#) in the AWS Knowledge Center.
- [Cross-account Access in Athena to Amazon S3 Buckets \(p. 251\)](#).

**Note**

Athena does not support restricting or allowing access to Amazon S3 resources based on the `aws:SourceIp` condition key.

## Fine-Grained 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 following Data Catalog objects that are used in Athena: databases and tables.

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 *AWS Identity and Access Management User Guide*.

See the following topics for these tasks:

To perform this task	See the following topic
Create an IAM policy that defines fine-grained access to resources	<a href="#">Creating IAM Policies</a> in the <i>AWS Identity and Access Management User Guide</i> .
Learn about IAM identity-based policies used in AWS Glue	<a href="#">Identity-Based Policies (IAM Policies)</a> in the <i>AWS Glue Developer Guide</i> .

**In this section**

- [Limitations \(p. 244\)](#)
- [Mandatory: Access Policy to the Default Database and Catalog per AWS Region \(p. 245\)](#)
- [Table Partitions and Versions in AWS Glue \(p. 246\)](#)
- [Fine-Grained Policy Examples \(p. 246\)](#)

## Limitations

Consider the following limitations when using fine-grained access control with the AWS Glue Data Catalog and Athena:

- You can limit access only to databases and tables. Fine-grained access controls apply at the table level and you cannot limit access to individual partitions within a table. For more information, see [Table Partitions and Versions in AWS Glue \(p. 246\)](#).
- Athena does not support cross-account access to the AWS Glue Data Catalog.
- 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 [Fine-Grained Policy Examples \(p. 246\)](#).

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 [Fine-Grained Policy Examples \(p. 246\)](#).
- 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 descendants. 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 [Fine-Grained Policy Examples \(p. 246\)](#).
- When limiting access to a specific database in the Data Catalog, you must also specify the access policy to the default database and catalog for each AWS Region for `GetDatabase` and `CreateDatabase` actions. If you use Athena in more than one Region, add a separate line to the policy for the resource ARN for each default database and catalog in each Region.

For example, to allow `GetDatabase` access to `example_db` in the `us-east-1` (N.Virginia) Region, also include the default database and catalog in the policy for that Region for two actions: `GetDatabase` and `CreateDatabase`:

```
{
  "Effect": "Allow",
  "Action": [
    "glue:GetDatabase",
    "glue:CreateDatabase"
  ],
  "Resource": [
    "arn:aws:glue:us-east-1:123456789012:catalog",
    "arn:aws:glue:us-east-1:123456789012:database/default",
    "arn:aws:glue:us-east-1:123456789012:database/example_db"
  ]
}
```

## Mandatory: Access Policy to the Default Database and Catalog per AWS Region

For Athena to work with the AWS Glue Data Catalog, the following access policy to the default database and to the AWS Glue Data Catalog per AWS Region for `GetDatabase` and `CreateDatabase` must be present :

```
{
  "Effect": "Allow",
  "Action": [
```

```
    "glue:GetDatabase",  
    "glue:CreateDatabase"  
  ],  
  "Resource": [  
    "arn:aws:glue:us-east-1:123456789012:catalog",  
    "arn:aws:glue:us-east-1:123456789012:database/default"  
  ]  
}
```

## 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 fine-grained access control, the following access permissions apply:

- Fine-grained access 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

Having access to all partitions within a table is not sufficient if you need to run actions in AWS Glue on partitions. To run actions on partitions, you need permissions for those actions. For example, to run `GetPartitions` on table `myTable` in the database `myDB`, you need permissions for the action `glue:GetPartitions` in the Data Catalog, the `myDB` database, and `myTable`.

- Fine-grained 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 Fine-Grained Permissions to Tables and Databases

The following table lists examples of IAM identity-based policies that allow fine-grained 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 the access policy to the `default` database and catalog, for `GetDatabase` and `CreateDatabase` actions. This policy is required for Athena and the AWS Glue Data Catalog to work together. For multiple AWS Regions, include this policy for each of the `default` databases and their catalogs, one line for each Region.

In addition, replace the `example_db` database and `test` table names with the names for your databases and tables.

DDL Statement	Example of an IAM access policy granting access to the resource
CREATE DATABASE	<p>Allows you to create the database named <code>example_db</code>.</p> <pre>{   "Effect": "Allow",   "Action": [     "glue:GetDatabase",     "glue:CreateDatabase"   ], }</pre>

DDL Statement	Example of an IAM access policy granting access to the resource
	<pre>"Resource": [   "arn:aws:glue:us-east-1:123456789012:catalog",   "arn:aws:glue:us-east-1:123456789012:database/default",   "arn:aws:glue:us-east-1:123456789012:database/example_db" ] }</pre>
ALTER DATABASE	<p>Allows you to modify the properties for the example_db database.</p> <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/default"   ] }, {   "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>

DDL Statement	Example of an IAM access policy granting access to the resource
DROP DATABASE	<p>Allows you to drop the <code>example_db</code> database, including all tables in it.</p> <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/default"   ] }, {   "Effect": "Allow",   "Action": [     "glue:GetDatabase",     "glue&gt;DeleteDatabase",     "glue:GetTables",     "glue:GetTable",     "glue&gt;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_db/*",     "arn:aws:glue:us-east-1:123456789012:userDefinedFunction/example_db/*"   ] }</pre>
SHOW DATABASES	<p>Allows you to list all databases in the AWS Glue Data Catalog.</p> <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/default"   ] }, {   "Effect": "Allow",   "Action": [     "glue:GetDatabases"   ],   "Resource": [     "arn:aws:glue:us-east-1:123456789012:catalog",     "arn:aws:glue:us-east-1:123456789012:database/*"   ] }</pre>

DDL Statement	Example of an IAM access policy granting access to the resource
CREATE TABLE	<p>Allows you to create a table named <code>test</code> in the <code>example_db</code> database.</p> <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/default"   ] }, {   "Effect": "Allow",   "Action": [     "glue:GetDatabase",     "glue:GetTable",     "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_db/test"   ] } </pre>
SHOW TABLES	<p>Allows you to list all tables in the <code>example_db</code> database.</p> <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/default"   ] }, {   "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_db/*"   ] } </pre>

DDL Statement	Example of an IAM access policy granting access to the resource
DROP TABLE	<p>Allows you to drop a partitioned table named <code>test</code> in the <code>example_db</code> database. If your table does not have partitions, do not include partition actions.</p> <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/default"   ] }, {   "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"   ] }</pre>

## Access 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 key being used to encrypt the catalog)"
  }
}
```

## 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 in AWS Glue, see [Cross-account AWS Glue Data Catalog access with Amazon Athena](#) in the AWS Big Data blog, or [Granting Cross-Account Access](#) in the *AWS Glue Developer Guide*.

The following example bucket policy, created and applied to the source data bucket `s3://my-athena-data-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",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3::my-athena-data-bucket",
        "arn:aws:s3::my-athena-data-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`.

## 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 AWS 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 Account B.
- The AWS KMS key policy in Account A must grant access to the user in Account B.
- The AWS Identity and Access Management (IAM) user policy in Account B must grant the user 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:::awsexamplebucket/*",
      "Principal": {
        "AWS": [
          "111122223333"
        ]
      }
    }
  ]
}
```

### To grant access to the user in Account B from the AWS KMS key policy in Account A

- In the AWS KMS key policy for Account A, grant the user in 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 user or role.

```
{
  "Sid": "Allow use of the key",
  "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 User Policy in Account B

1. From Account B, open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Open the IAM user or role associated with the user in Account B.
3. Review the list of permissions policies applied to IAM user or role.
4. Ensure that a policy is applied that grants access to the bucket.

The following example statement grants the IAM user access to the `s3:GetObject` and `s3:PutObject` operations on the bucket `awsexamplebucket`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt2",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::awsexamplebucket/*"
    }
  ]
}
```

5. Ensure that a policy is applied that grants access to the key.

#### Note

If the IAM user or role in 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 user access to use the key `arn:aws:kms:example-region-1:123456789098: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:example-
region-1:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd"
}
]
```

For instructions on how to add or correct the IAM user's permissions, see [Changing Permissions for an IAM User](#).

## 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?](#).

## 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 AWS account `123456789012`, specify the workgroup as a resource using the following ARN:

```
"Resource":["arn:aws:athena:us-east-2:123456789012:workgroup/test_workgroup"]
```

- For a list of workgroup policies, see [the section called “Workgroup Example Policies” \(p. 326\)](#).
- For a list of tag-based policies for workgroups, see [Tag-Based IAM Access Control Policies \(p. 353\)](#).
- For more information about creating IAM policies for workgroups, see [Workgroup IAM Policies \(p. 325\)](#).
- 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*.

## Allow Access to an 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 \(p. 255\)](#)
- [Example Policy to Allow an IAM Principal to Create an Athena Data Connector for External Hive Metastore \(p. 256\)](#)

## 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 [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#), 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:*:MyAWSAcctId:function:MyAthenaLambdaFunction",
        "arn:aws:lambda:*:MyAWSAcctId:function:AnotherAthenaLambdaFunction",
        "arn:aws:lambda:*:MyAWSAcctId: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:MyAthenaLambda 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:MyAthenaLambda

Allowed Actions	Explanation
	where <i>MyAthenaLambdaFunction</i> 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 [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#), 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:*:MyAWSacctId: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:

- `AWSLambdaVPCLambdaAccessExecutionRole` – 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 \(p. 240\)](#) 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::mybucket/spill"
      ]
    }
  ]
}
```

## Creating 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](#) (p. 260).

## Catalog Registration and Metadata API Operations

For access to catalog registration API and metadata API operations, use the [AmazonAthenaFullAccess managed policy](#) (p. 240). If you do not use this policy, add the following API operations to your Athena policies:

```
{
  "Effect": "Allow",
  "Action": [
    "athena:ListDataCatalogs",
    "athena:GetDataCatalog",
    "athena:CreateDataCatalog",
    "athena:UpdateDataCatalog",
    "athena>DeleteDataCatalog",
    "athena:GetDatabase",
    "athena:ListDatabases",
    "athena:GetTableMetadata",
    "athena:ListTableMetadata"
  ],
  "Resource": [
    "*"
  ]
}
```

## Cross Region Lambda Invocation

To invoke a Lambda function in a region other than the region in which you are running Athena queries, use the full ARN of the Lambda function. By default, Athena invokes Lambda functions defined in the

same region. If you need to invoke a Lambda function to access a Hive metastore in a region other than the region in which you run Athena queries, you must provide the full ARN of the Lambda function.

For example, suppose you define the catalog ehms on the Europe (Frankfurt) Region eu-central-1 to use the following Lambda function in the US East (N. Virginia) 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 ehms should be registered in the same region that you run Athena queries.

## Cross Account Lambda Invocation

Sometimes you might require access to a Hive metastore from a different account. For example, to run a Hive metastore, you might launch an EMR cluster from 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](#) 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 [Cross Region Lambda Invocation \(p. 257\)](#) earlier in this topic.

## Granting 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.

```
{
  "Sid": "Stmt1234567890123",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::444455556666:user/perf1-test"
  },
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::athena-test/lambda/dataset/*"
}
```

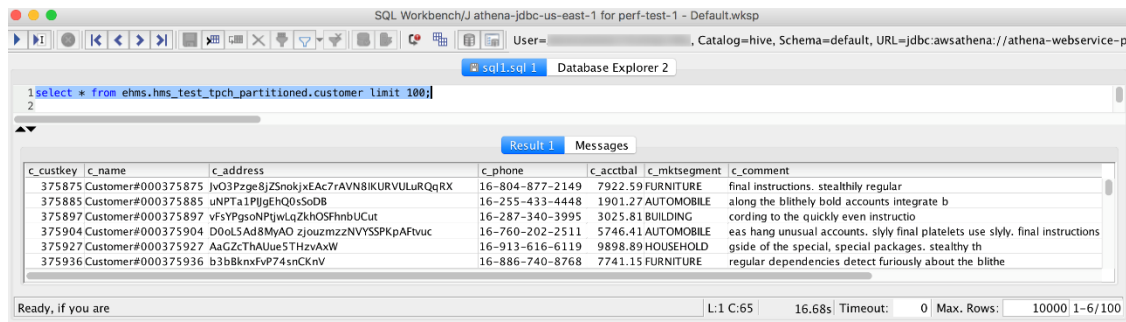
### Note

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`.





The screenshot shows the SQL Workbench/J interface. The query bar contains: `1 select * from ehms.hms_test_tpch_partitioned.customer limit 100;`. The results pane shows a table with columns: c\_custkey, c\_name, c\_address, c\_phone, c\_acctbal, c\_mktsegment, and c\_comment. The status bar at the bottom indicates: Ready, if you are L:1 C:65 16.68s Timeout: 0 Max. Rows: 10000 1-6/100.

c_custkey	c_name	c_address	c_phone	c_acctbal	c_mktsegment	c_comment
375875	Customer#000375875	JvO3Pzge8JZSnokjxEAc7rAVN8IKURVULuRQqRX	16-804-877-2149	7922.59	FURNITURE	final instructions. stealthily regular
375885	Customer#000375885	uNPTa1PIjgEhQ0sSoDB	16-255-433-4448	1901.27	AUTOMOBILE	along the blithely bold accounts integrate b
375897	Customer#000375897	vFsyPgs0NPjwLqZkhOSFhnbUCut	16-287-340-3995	3025.81	BUILDING	ording to the quickly even instructio
375904	Customer#000375904	D0oL5Ad8MyAO zjouzmzzNVYSSPKpAFtvuc	16-760-202-2511	5746.41	AUTOMOBILE	eas hang unusual accounts. slyly final platelets use slyly. final instructions
375927	Customer#000375927	AaGZcThAUue5THzvAXW	16-913-616-6119	9898.89	HOUSEHOLD	gside of the special, special packages. stealthy th
375936	Customer#000375936	b3bBknxfvP74snCKnV	16-886-740-8768	7741.15	FURNITURE	regular dependencies detect furiously about the blithe

## Example IAM Permissions Policies to Allow Athena Federated Query (Preview)

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 \(Preview\) \(p. 260\)](#)
- [Example Policy to Allow an IAM Principal to Create a Data Source Connector \(p. 262\)](#)

### Example – Allow an IAM Principal to Run and Return Results Using Athena Federated Query (Preview)

The following identity-based permissions policy allows actions that a user or other IAM principal requires to use Athena Federated Query (Preview). 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": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "athena:GetWorkGroup",
        "s3:PutObject",
        "s3:GetObject",
        "athena:StartQueryExecution",
        "s3:AbortMultipartUpload",
        "lambda:InvokeFunction",
        "athena:CancelQueryExecution",
        "athena:StopQueryExecution",
        "athena:GetQueryExecution",
        "athena:GetQueryResults",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:athena:*:MyAWSAcctId:workgroup/AmazonAthenaPreviewFunctionality",
        "arn:aws:s3:::MyQueryResultsBucket/*",
        "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillPrefix",
        "arn:aws:lambda:*:MyAWSAcctId:function:OneAthenaLambdaFunction",

```

```

        "arn:aws:lambda:*:MyAWSAcctId:function:AnotherAthenaLambdaFunction"
    ],
    {
        "Sid": "VisualEditor1",
        "Effect": "Allow",
        "Action": "athena:ListWorkGroups",
        "Resource": "*"
    },
    {
        "Sid": "VisualEditor2",
        "Effect": "Allow",
        "Action": [
            "s3:ListBucket",
            "s3:GetBucketLocation"
        ],
        "Resource": "arn:aws:s3:::MyLambdaSpillBucket"
    }
]
}

```

### Explanation of Permissions

Allowed Actions	Explanation
"athena:StartQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:CancelQueryExecution", "athena:StopQueryExecution", "athena:GetQueryExecution",	Athena permissions that are required to run queries in the AmazonAthenaPreviewFunctionality work group.
"s3:PutObject", "s3:GetObject", "s3:AbortMultipartUpload"	<p>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:::<i>MyQueryResultsBucket</i>/<i>*</i> resource identifier, where <i>MyQueryResultsBucket</i> is the Athena query results bucket. For more information, see <a href="#">Working with Query Results, Output Files, and Query History</a> (p. 110).</p> <p>s3:GetObject allows reading of query results and query history for the resource specified as arn:aws:s3:::<i>MyQueryResultsBucket</i>, where <i>MyQueryResultsBucket</i> is the Athena query results bucket.</p> <p>s3:GetObject also allows reading from the resource specified as "arn:aws:s3:::<i>MyLambdaSpillBucket</i>/<i>MyLambdaSpillPrefix</i>", where <i>MyLambdaSpillPrefix</i> is specified in the configuration of the Lambda function or functions being invoked.</p>
"lambda:InvokeFunction"	Allows queries to invoke the AWS Lambda functions for the AWS Lambda functions specified in the Resource block. For example,

Allowed Actions	Explanation
	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 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:*:MyAWSAcctId:function:MyAthenaLambdaFunctionsPrefix*",
        "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
        "arn:aws:iam:*:role/*",
        "arn:aws:iam:MyAWSAcctId: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:*:MyAWSAcctId:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
        "arn:aws:cloudformation:*:MyAWSAcctId:stack/
serverlessrepo-MyCFStackPrefix*/*",
        "arn:aws:cloudformation:*:*:transform/Serverless-*",
        "arn:aws:cloudformation:*:MyAWSAcctId:stackset/aws-serverless-
repository-MyCFStackPrefix*/*",
        "arn:aws:cloudformation:*:MyAWSAcctId:stackset/
serverlessrepo-MyCFStackPrefix*/*"
    ]
},
{
    "Sid": "VisualEditor3",
    "Effect": "Allow",
    "Action": "serverlessrepo:*",
    "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
}
]
}

```

## Explanation of Permissions

Allowed Actions	Explanation
<pre> "lambda:CreateFunction", "lambda:ListVersionsByFunction", "lambda:GetFunctionConfiguration", "lambda:PutFunctionConcurrency", "lambda:ListTags", "lambda&gt;DeleteFunction", "lambda:GetAlias", "lambda:InvokeFunction", "lambda:GetFunction", "lambda:ListAliases", "lambda:UpdateFunctionConfiguration", "lambda:UpdateFunctionCode", "lambda:AddPermission", "lambda&gt;DeleteFunctionConcurrency", "lambda:RemovePermission", "lambda:GetPolicy" "lambda:GetAccountSettings", "lambda:ListFunctions", "lambda:ListEventSourceMappings", </pre>	<p>Allow the creation and management of Lambda functions listed as resources. In the example, a name prefix is used in the resource identifier <code>arn:aws:lambda:*:MyAWSAcctId:function:MyAthenaLambdaFunctionPrefix</code> where <code>MyAthenaLambdaFunctionPrefix</code> 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.</p>

Allowed Actions	Explanation
"s3:GetObject"	Allows reading of a bucket that AWS Serverless Application Repository requires as specified by the resource identifier <code>arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*</code> . This bucket may be specific to your account.
"cloudformation:*"	Allows the creation and management of AWS CloudFormation stacks specified by the resource <code>MyCFStackPrefix</code> . 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 <code>arn:aws:serverlessrepo:*:*:applications/*</code> .

## Example IAM Permissions Policies to Allow Amazon Athena User Defined Functions (UDF)

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 \(p. 264\)](#)
- [Example Policy to Allow an IAM Principal to Create an Athena UDF \(p. 266\)](#)

### 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:CancelQueryExecution",
        "athena:StopQueryExecution",

```

```

        "athena:GetQueryExecution",
        "s3:GetBucketLocation"
    ],
    "Resource": [
        "arn:aws:athena:*:MyAWSAcctId:workgroup/AmazonAthenaPreviewFunctionality",
        "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:CancelQueryExecution", "athena:StopQueryExecution", "athena:GetQueryExecution",	Athena permissions that are required to run queries in the AmazonAthenaPreviewFunctionality work group.
"s3:PutObject", "s3:GetObject", "s3:AbortMultipartUpload"	<p>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:::<b>MyQueryResultsBucket</b>/* resource identifier, where <b>MyQueryResultsBucket</b> is the Athena query results bucket. For more information, see <a href="#">Working with Query Results, Output Files, and Query History</a> (p. 110).</p> <p>s3:GetObject allows reading of query results and query history for the resource specified as arn:aws:s3:::<b>MyQueryResultsBucket</b>, where <b>MyQueryResultsBucket</b> is the Athena query results bucket. For more information, see <a href="#">Working with Query Results, Output Files, and Query History</a> (p. 110).</p> <p>s3:GetObject also allows reading from the resource specified as "arn:aws:s3:::<b>MyLambdaSpillBucket</b>/<b>MyLambdaSpillPrefix</b>*", where <b>MyLambdaSpillPrefix</b> is specified in the configuration of the Lambda function or functions being invoked.</p>
"lambda:InvokeFunction"	Allows queries to invoke the AWS Lambda functions specified in the Resource block. For example,

Allowed Actions	Explanation
	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:*:MyAWSAcctId:function:MyAthenaLambdaFunctionsPrefix*",
        "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
        "arn:aws:iam:*:role/*",
        "arn:aws:iam:MyAWSAcctId: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:*:MyAWSAcctId:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
      "arn:aws:cloudformation:*:MyAWSAcctId:stack/
serverlessrepo-MyCFStackPrefix*/*",
      "arn:aws:cloudformation:*:*:transform/Serverless-*",
      "arn:aws:cloudformation:*:MyAWSAcctId:stackset/aws-serverless-
repository-MyCFStackPrefix*/*",
      "arn:aws:cloudformation:*:MyAWSAcctId:stackset/
serverlessrepo-MyCFStackPrefix*/*"
    ]
  },
  {
    "Sid": "VisualEditor3",
    "Effect": "Allow",
    "Action": "serverlessrepo:*",
    "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
  }
]
}

```

### 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 <code>arn:aws:lambda:*:MyAWSAcctId:function:MyAthenaLambdaFunctionPrefix</code> where <code>MyAthenaLambdaFunctionPrefix</code> 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.



Allowed Actions	Explanation
"s3:GetObject"	Allows reading of a bucket that AWS Serverless Application Repository requires as specified by the resource identifier <code>arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*</code> .
"cloudformation:*"	Allows the creation and management of AWS CloudFormation stacks specified by the resource <i>MyCFStackPrefix</i> . 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 <code>arn:aws:serverlessrepo:*:*:applications/*</code> .

## Allowing Access for ML with Athena (Preview)

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 [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#), 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](#) in the *IAM User Guide*.

```
{
    "Effect": "Allow",
    "Action": [
        "sagemaker:invokeEndpoint"
    ],
    "Resource": "arn:aws:sagemaker:us-west-2:123456789012:workteam/public-crowd/
default"
}
```

## Enabling 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 SAML2.0 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. Access is established through the versions of JDBC or ODBC drivers that support SAML 2.0. For information, see [Using Athena with the JDBC Driver \(p. 72\)](#) and [Connecting to Amazon Athena with ODBC \(p. 73\)](#).

### Topics

- [Before You Begin \(p. 269\)](#)
- [Architecture Diagram \(p. 269\)](#)
- [Procedure: SAML-based Federated Access to the Athena API \(p. 270\)](#)

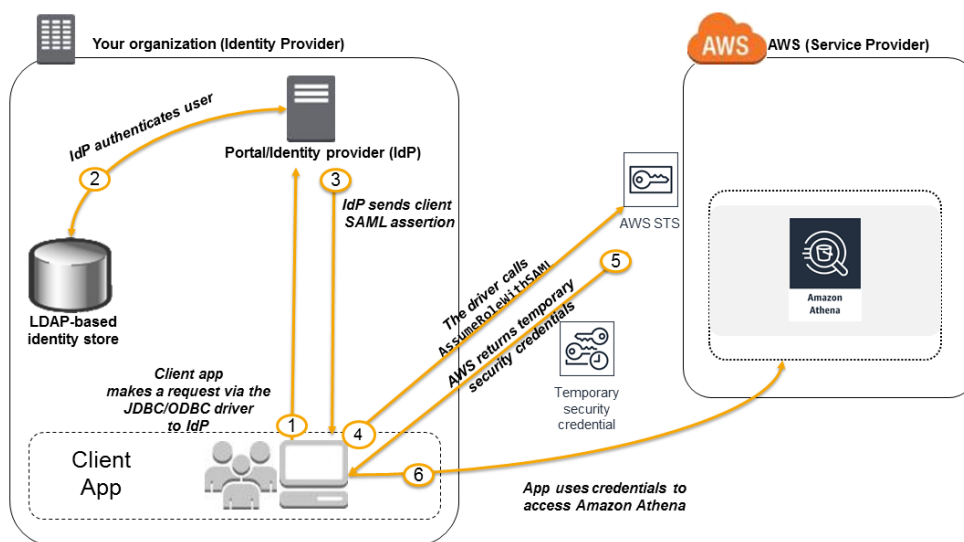
## Before You Begin

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 [Using Athena with the JDBC Driver \(p. 72\)](#) and [Connecting to Amazon Athena with ODBC \(p. 73\)](#).

## Architecture Diagram

The following diagram illustrates this process.



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: 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 [Using Athena with the JDBC Driver \(p. 72\)](#) and [Connecting to Amazon Athena with ODBC \(p. 73\)](#).

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 [Using Athena with the JDBC Driver \(p. 72\)](#) and [Connecting to Amazon Athena with ODBC \(p. 73\)](#) 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 \(p. 269\)](#).
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.

## Logging and Monitoring in 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. You can also use Athena to query CloudTrail log files for insight. For more information, see [Querying AWS CloudTrail Logs \(p. 203\)](#) and [CloudTrail SerDe \(p. 367\)](#).
- **Use CloudWatch Events with Athena** – CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources. CloudWatch Events 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. To use CloudWatch Events with Athena, create a rule that triggers on an Athena API call via CloudTrail. For more information, see [Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using CloudTrail](#) in the *Amazon CloudWatch Events 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 a high-level procedure, see [Setting up Workgroups](#) (p. 324). Use resource-level IAM permissions to control access to a specific workgroup. For more information, see [Using Workgroups for Running Queries](#) (p. 322) and [Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events](#) (p. 338).

## Compliance Validation for Amazon 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 Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [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 the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Athena through the network. Clients must support TLS (Transport Layer Security) 1.0. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Use IAM policies to restrict access to Athena operations. Athena [managed policies](#) (p. 240) 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](#) (p. 251).

### Topics

- [Connect to Amazon Athena Using an Interface VPC Endpoint](#) (p. 273)

## Connect to Amazon Athena Using an Interface VPC Endpoint

You can connect directly to Athena using an [interface VPC endpoint](#) (AWS PrivateLink) in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use an interface VPC endpoint, communication between your VPC and Athena is conducted entirely within the AWS network. 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*.

### Note

AWS PrivateLink for Athena is not supported in the Europe (Stockholm) Region. Athena supports VPC endpoints in all other AWS Regions where both [Amazon VPC](#) and [Athena](#) are available.

You can create an interface VPC endpoint to connect to Athena using the AWS 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 the following:

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

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

### Example – VPC Endpoint Policy for Athena Actions

The endpoint to which this policy is attached grants access to the listed athena actions to all principals in *workgroupA*.

```
{
  "Statement": [{
    "Principal": "*",
    "Effect": "Allow",
    "Action": [
      "athena:StartQueryExecution",
      "athena:RunQuery",
      "athena:GetQueryExecution",
      "athena:GetQueryResults",
      "athena:CancelQueryExecution",
      "athena:ListWorkGroups",
      "athena:GetWorkGroup",
      "athena:TagResource"
    ],
    "Resource": [
      "arn:aws:athena:us-west-1:AWSAccountId:workgroup/workgroupA"
    ]
  }]
}
```

## 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 resources:



- [Shared Responsibility Model](#)
- [Amazon Web Services: Overview of Security Processes](#) (whitepaper)

## Using 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. Lake Formation provides an authorization and governance layer on data stored in Amazon S3. 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 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. To use Athena with data registered using Lake Formation, Athena must be configured to use the AWS Glue Data Catalog.

Lake Formation permissions do not apply when writing objects to Amazon S3, nor do they apply when querying data stored in Amazon S3 or metadata that are not registered with Lake Formation. For source data in Amazon S3 and metadata that is not registered with Lake Formation, access is determined by IAM permissions policies for Amazon S3 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](#).

### Topics

- [How Athena Accesses Data Registered With Lake Formation](#) (p. 275)
- [Considerations and Limitations When Using Athena to Query Data Registered With Lake Formation](#) (p. 277)
- [Managing Lake Formation and Athena User Permissions](#) (p. 279)
- [Applying Lake Formation Permissions to Existing Databases and Tables](#) (p. 281)
- [Using Lake Formation and the Athena JDBC and ODBC Drivers for Federated Access to Athena](#) (p. 281)

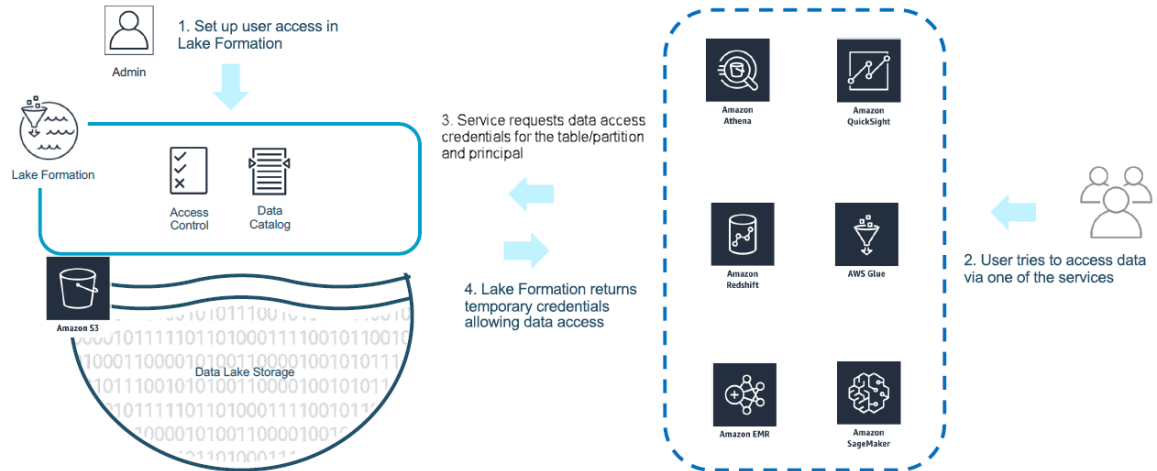
## How Athena Accesses Data Registered With Lake Formation

The access workflow described in this section applies only when running Athena queries on Amazon S3 locations and 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 and 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*.

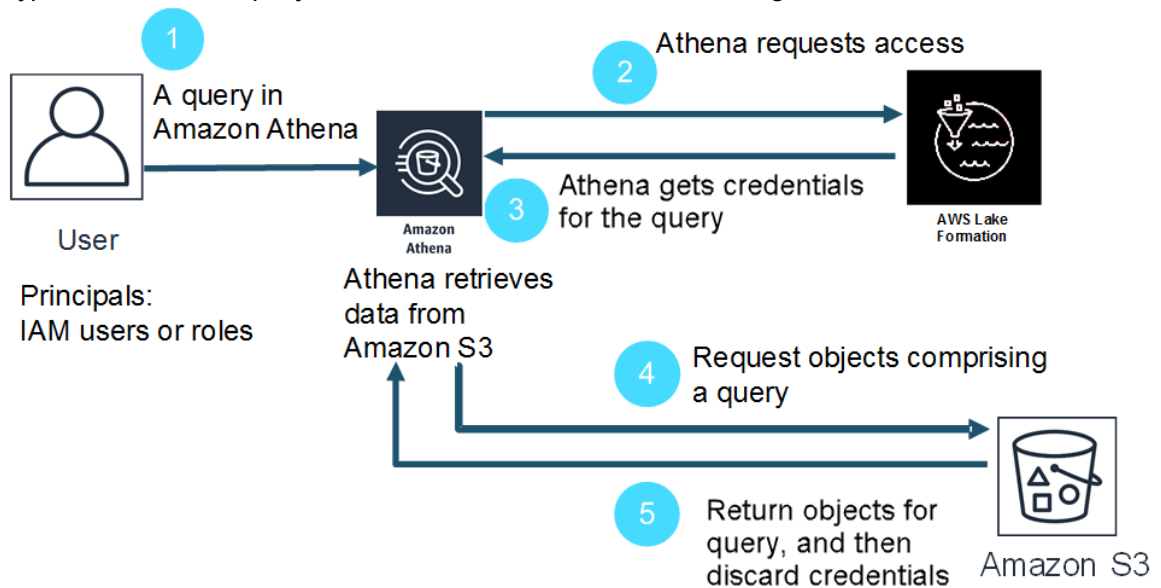


Each time an Athena principal (user, group, or role) runs a query on data registered using Lake Formation, Lake Formation verifies that the principal has the appropriate Lake Formation permissions to the database, table, and Amazon S3 location as appropriate for the query. If the principal has access, Lake Formation *vends* temporary credentials to Athena, and the query runs.

The following diagram illustrates the flow described above.



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 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, table partitions (if applicable), 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 and accessing metadata from the Data Catalog.
5. Lake Formation returns query results to Athena. After the query completes, Athena discards the credentials.

## Considerations and Limitations When Using Athena to Query 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 \(p. 277\)](#)
- [Working With Lake Formation Permissions To Views \(p. 277\)](#)
- [Athena Query Results Location In Amazon S3 Not Registered With Lake Formation \(p. 277\)](#)
- [Use Athena Workgroups To Limit Access To Query History \(p. 278\)](#)
- [Cross-Account Data Catalog Access \(p. 278\)](#)
- [CSE-KMS Encrypted Amazon S3 Locations Registered With Lake Formation Cannot Be Queried in Athena \(p. 278\)](#)
- [Partitioned Data Locations Registered with Lake Formation Must Be In Table Sub-Directories \(p. 279\)](#)
- [Create Table As Select \(CTAS\) Queries Require Amazon S3 Write Permissions \(p. 279\)](#)

## 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 Avro storage format or using a custom Serializer/Deserializers (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.

## Working With Lake Formation Permissions To 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.

## 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 do not limit access to these locations. Unless you limit access, Athena users can access query result files and metadata when they do not 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 [Working with Query Results, Output Files, and Query History](#) (p. 110).

## 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 [Using Workgroups to Control Query Access and Costs](#) (p. 322).

## Cross-Account Data Catalog Access

To access a data catalog in another account, you can use one of the following methods:

- Set up cross-account access in Lake Formation.
- Use an Athena cross-account [AWS Lambda](#) function to federate queries to the Data Catalog of your choice.

### Setting Up 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 more information, 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](#)

### Using an Athena Cross-Account Lambda Function

You can use Athena to [connect to an external Hive metastore](#) (p. 34). The Hive metastore functionality uses a Lambda function to federate queries to the Data Catalog of your choice. This same functionality can proxy catalog queries to a Data Catalog in a different account.

For steps, see [Cross-account AWS Glue Data Catalog access with Amazon Athena](#) in the AWS Big Data Blog.

## CSE-KMS Encrypted Amazon S3 Locations Registered With Lake Formation Cannot Be Queried in Athena

Amazon S3 data locations that are registered with Lake Formation and encrypted using client-side encryption (CSE) with AWS KMS customer-managed keys (CSE-KMS) cannot be queried using Athena. You still can use Athena to query CSE-KMS encrypted Amazon S3 data locations that are not registered with Lake Formation and use IAM policies to allow or deny access.

## Partitioned Data Locations Registered with Lake Formation Must Be In Table Sub-Directories

Partitioned tables registered with Lake Formation must have partitioned data in directories that are sub-directories of the table in Amazon S3. For example, a table with the location `s3://mydata/mytable` and partitions `s3://mydata/mytable/dt=2019-07-11`, `s3://mydata/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://mydata/mytable` and partitions located in `s3://mydata/dt=2019-07-11`, `s3://mydata/dt=2019-07-12`, and so on, cannot be registered with Lake Formation. You can set up access for these tables using IAM permissions outside of Lake Formation to query them in Athena. For more information, see [Partitioning Data](#) (p. 92).

## 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 [Creating a Table from Query Results \(CTAS\)](#) (p. 124).

## Managing Lake Formation and Athena User Permissions

Lake Formation vends credentials to query Amazon S3 data stores that are registered with Lake Formation. If you previously used IAM policies to allow or deny permissions to read data locations in Amazon S3, you can use Lake Formation permissions instead. However, other IAM permissions are still required.

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](#) (p. 279)
- [Amazon S3 Permissions For Athena Query Results Locations](#) (p. 280)
- [Athena Workgroup Memberships To Query History](#) (p. 280)
- [Lake Formation Permissions To Data](#) (p. 280)
- [IAM Permissions to Write to Amazon S3 Locations](#) (p. 280)
- [Permissions to Encrypted Data, Metadata, and Athena Query Results](#) (p. 280)
- [Resource-Based Permissions for Amazon S3 Buckets in External Accounts \(Optional\)](#) (p. 281)

## 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 [AmazonAthenaFullAccess Managed Policy](#) (p. 240) 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. 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*.

An Lake Formation user can use Athena to query databases, tables, table columns, and underlying Amazon S3 data stores 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 AWS account users. In addition, federated access might be provided through the SAML-based authentication available with Athena drivers. For more information, see [Using Workgroups to Control Query Access and Costs](#) (p. 322), [IAM Policies for Accessing Workgroups](#) (p. 325), and [Enabling Federated Access to the Athena API](#) (p. 268).

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 do not limit access to these locations. Unless you limit access, Athena users can access query result files and metadata when they do not 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 [Working with Query Results, Output Files, and Query History](#) (p. 110).

## 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 [Using Workgroups to Control Query Access and Costs](#) (p. 322).

## 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 [Creating a Table from Query Results \(CTAS\)](#) (p. 124).

## Permissions to Encrypted Data, Metadata, and Athena Query Results

Underlying source data in Amazon S3 and metadata in the Data 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 [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#).

- **Encrypting source data** – SSE-S3 and CSE-KMS encryption of Amazon S3 data locations source data is supported. SSE-KMS encryption is not 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 [Permissions to Encrypted Data in Amazon S3 \(p. 234\)](#).
- **Encrypting metadata** – Encrypting metadata in the 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 [Access to Encrypted Metadata in the AWS Glue Data Catalog \(p. 250\)](#).

## 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 [Cross-account Access in Athena to Amazon S3 Buckets \(p. 251\)](#).

For information about accessing a Data Catalog in another account, see [Cross-Account Data Catalog Access \(p. 278\)](#).

## Applying 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 Amazon S3 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 in Amazon S3—and AWS Glue, if applicable—can continue to query data not registered with Lake Formation.

If you have existing Athena users who query 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 [Fine-Grained Access to Databases and Tables in the AWS Glue Data Catalog \(p. 244\)](#) and [Upgrading AWS Glue Data Permissions to the AWS Lake Formation Model](#) in the *AWS Lake Formation Developer Guide*.

## Using Lake Formation and the Athena JDBC and 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 Access and Identity Management (IAM) roles. For detailed steps, see [Tutorial: Configuring Federated Access for Okta Users to Athena Using Lake Formation and JDBC \(p. 282\)](#).

## 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 \(p. 73\)](#).

## 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 and Microsoft Active Directory Federation Services (AD FS) 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.
- 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: Configuring Federated Access for Okta Users to Athena Using Lake Formation and JDBC \(p. 282\)](#)

## Tutorial: Configuring 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 AWS account. To create an account, visit the [Amazon Web Services home page](#).
- [Set up a query results location \(p. 115\)](#) 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 \(p. 76\)](#) 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 \(p. 283\)](#)
- [Step 2: Add users and groups to Okta \(p. 283\)](#)
- [Step 3: Set up an Okta Application for SAML Authentication \(p. 289\)](#)
- [Step 4: Create an AWS SAML Identity Provider and Lake Formation Access IAM Role \(p. 297\)](#)
- [Step 5: Add the IAM Role and SAML Identity Provider to the Okta Application \(p. 303\)](#)
- [Step 6: Grant user and group permissions through AWS Lake Formation \(p. 308\)](#)
- [Step 7: Verify access through the Athena JDBC client \(p. 312\)](#)
- [Conclusion \(p. 320\)](#)
- [Related Resources \(p. 320\)](#)

## 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

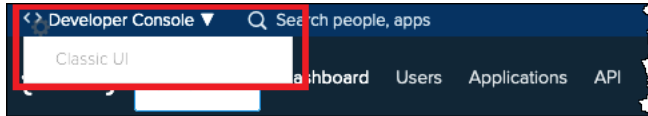
In this step, you use the Okta console to perform the following tasks:

- Create two Okta users.
- Create two Okta groups.
- Add one Okta user to each Okta group.

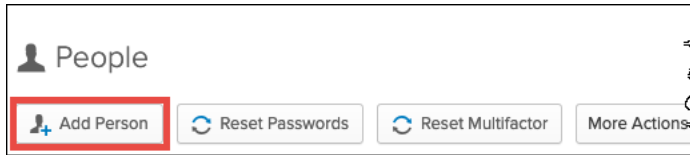
### To add users to Okta

1. After you activate your Okta account, log in as administrative user to the assigned Okta domain.
2. If you are in the **Developer Console**, use the option on the top left of the page to choose the **Classic UI**.





3. In the **Classic UI**, choose **Directory**, and then choose **People**.
4. Choose **Add Person** to add a new user who will access Athena through the JDBC driver.



5. 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.

A screenshot of the 'Add Person' dialog box. The form contains the following fields and options:

- User type**: A dropdown menu with 'User' selected.
- First name**: A text input field containing 'athena-okta-user'.
- Last name**: A text input field containing 'athena-okta-user'.
- Username**: A text input field containing 'athena-okta-user@anycompany.com'.
- Primary email**: A text input field containing 'athena-okta-user@anycompany.com'.
- Secondary email (optional)**: An empty text input field.
- Groups (optional)**: A section header with the text 'You haven't added any groups' below it.
- Password**: A dropdown menu with 'Set by admin' selected, and a text input field below it containing 'Enter password'.
- User must change password on first login**: A checkbox that is unchecked, highlighted with a red rectangular box.

At the bottom of the dialog, there are three buttons: 'Save', 'Save and Add Another' (highlighted with a red rectangular box), and 'Cancel'.

6. Choose **Save and Add Another**.
7. 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) You haven't added any [groups](#)

**Password** ? Set by admin

.....

☐ User must change password on first login

**Save** **Save and Add Another** Cancel

8. 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. From the Okta classic UI, 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*.

A screenshot of the 'Add Group' dialog box. It has a blue header with the text 'Add Group'. Below the header, there's a message: 'Add groups so you can quickly perform actions across large sets of people.' There are two input fields: 'Name' with the value 'lf-business-analyst' and 'Group Description' with the value 'Business Analysts'. At the bottom right, there are two buttons: 'Add Group' (highlighted with a red rectangle) and 'Cancel'.

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*.

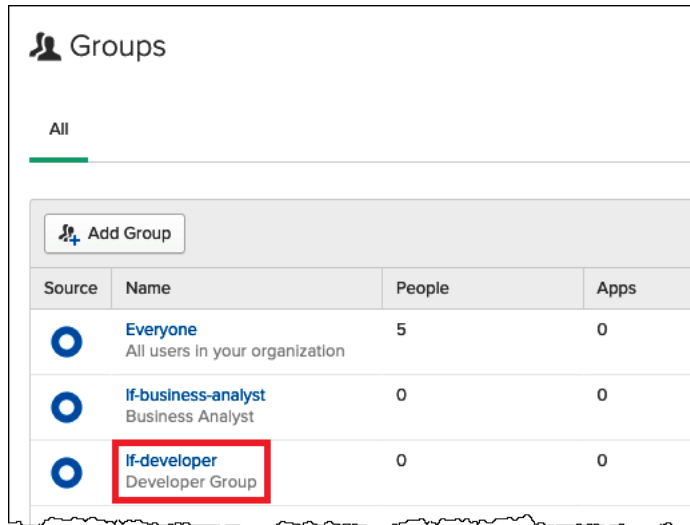
A screenshot of the 'Add Group' dialog box, similar to the previous one. It has a blue header with the text 'Add Group'. Below the header, there's a message: 'Add groups so you can quickly perform actions across large sets of people.' There are two input fields: 'Name' with the value 'lf-developer' and 'Group Description' with the value 'Developers'. At the bottom right, there are two buttons: 'Add Group' (highlighted with a red rectangle) and 'Cancel'.

7. Choose **Add Group**.

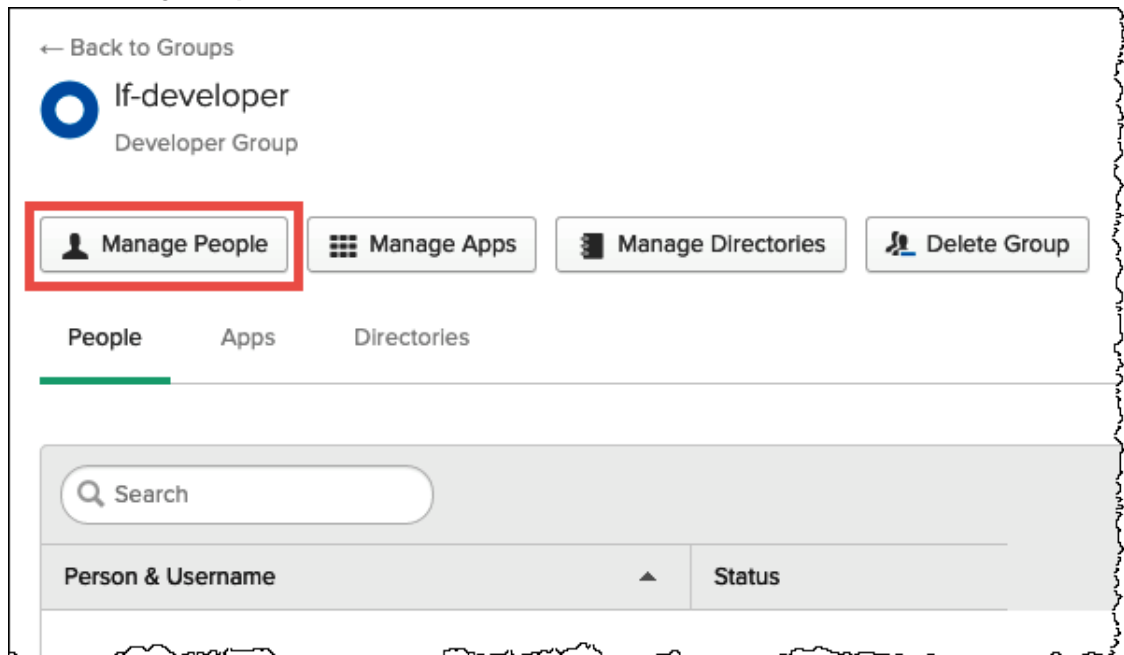
Now that you have two users and two groups, you are ready to add a user to each group.

## To add users to groups


1. On the **Groups** page, choose the **If-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**.

[← Back to Group](#)  
 **If-developer**  
Developers

Add or remove people from the If-developer group


Cancel

Save

Search by person 

+ Add All 4



- Remove All 0

 **Not Members** Showing 1 - 4 of 4


Person & Username ▼

athena-ba-user athena-ba-user  
athena-ba-user@anycompany.com

athena-okta-user athena-okta-user  
athena-okta-user@anycompany.com

First Previous 1 Next Last

 **Members**

Person & Username ▲

First Previous Next Last


Cancel

Save

The entry for the user moves from the **Not Members** list on the left to the **Members** list on the right.

288


← Back to Group

 **If-developer**  
Developers

Add or remove people from the If-developer group

3


1

 **Not Members** Showing 1 - 3 of 3

Person & Username

athena-ba-user athena-ba-user  
athena-ba-user@anycompany.com

First Previous **1** Next Last

 **Members** Showing 1 - 1 of 1

Person & Username

athena-okta-user athena-okta-user  
athena-okta-user@anycompany.com

First Previous **1** Next Last

4. Choose **Save**.
5. Choose **Back to Groups**, or choose **Directory**, and then choose **Groups**.
6. Choose the **If-business-analyst** group.
7. Choose **Manage People**.
8. Add the **athena-ba-user** to the **Members** list of the **If-business-analyst** group, and then choose **Save**.
9. Choose **Back to Groups**, or choose **Directory, Groups**.

The **Groups** page now shows that each group has one Okta user.

Source	Name	People	Apps
	<b>If-business-analyst</b> Business Analyst	1	0
	<b>If-developer</b> Developer Group	1	0

### Step 3: Set up an Okta Application for SAML Authentication

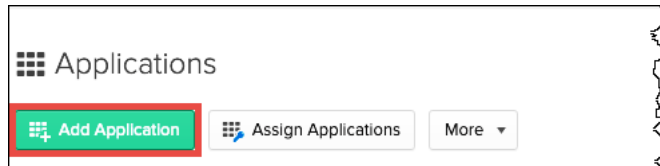
In this step, you use the Okta developer console to perform the following tasks:

- Add a SAML application for use with AWS.
- Assign the application to the Okta user.
- Assign the application to an Okta group.

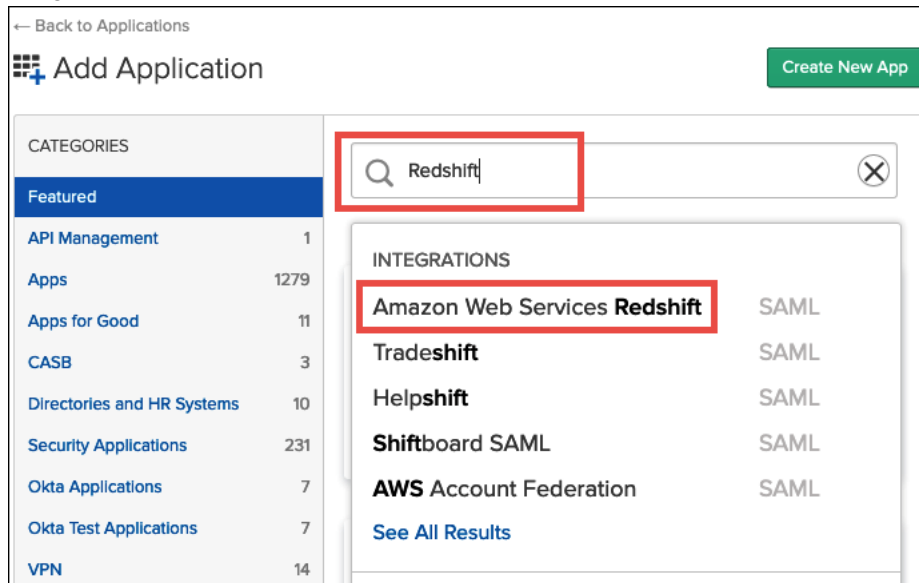
- Download the resulting identity provider metadata for later use with AWS.

### To add an application for SAML authentication

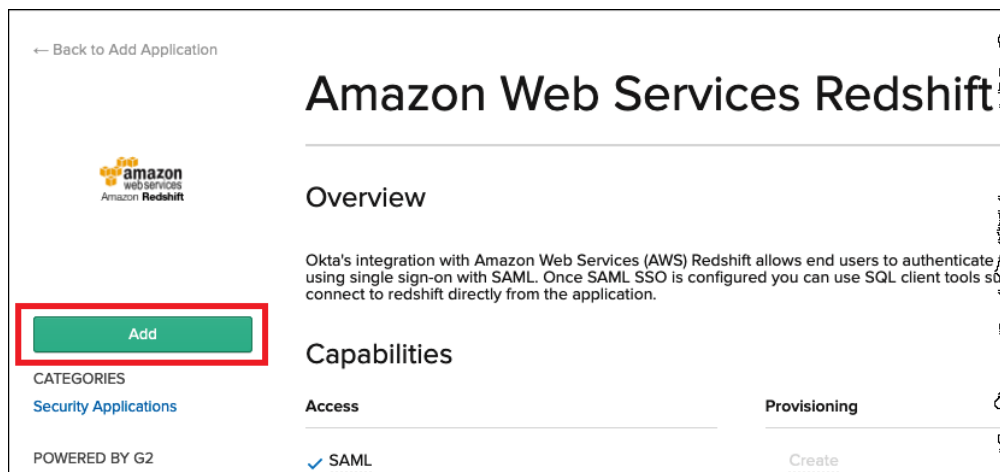
1. From the menu, choose **Applications** so that you can configure an Okta application for SAML authentication to Athena.
2. Click **Add Application**.



3. In the search box, search for **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**.

The screenshot shows the 'Add Amazon Web Services Redshift' dialog box with the 'General Settings' tab selected. The 'Application label' field is highlighted with a red box and contains the text 'Athena-LakeFormation-Okta'. Below it, a note states 'This label displays under the app on your home page'. Under the 'Application Visibility' section, there are two unchecked checkboxes: 'Do not display application icon to users' and 'Do not display application icon in the Okta Mobile App'. At the bottom, the 'Done' button is highlighted with a red box, while the 'Cancel' button is not.

Now that you have created an Okta application, you can assign it to the users and groups that you created.

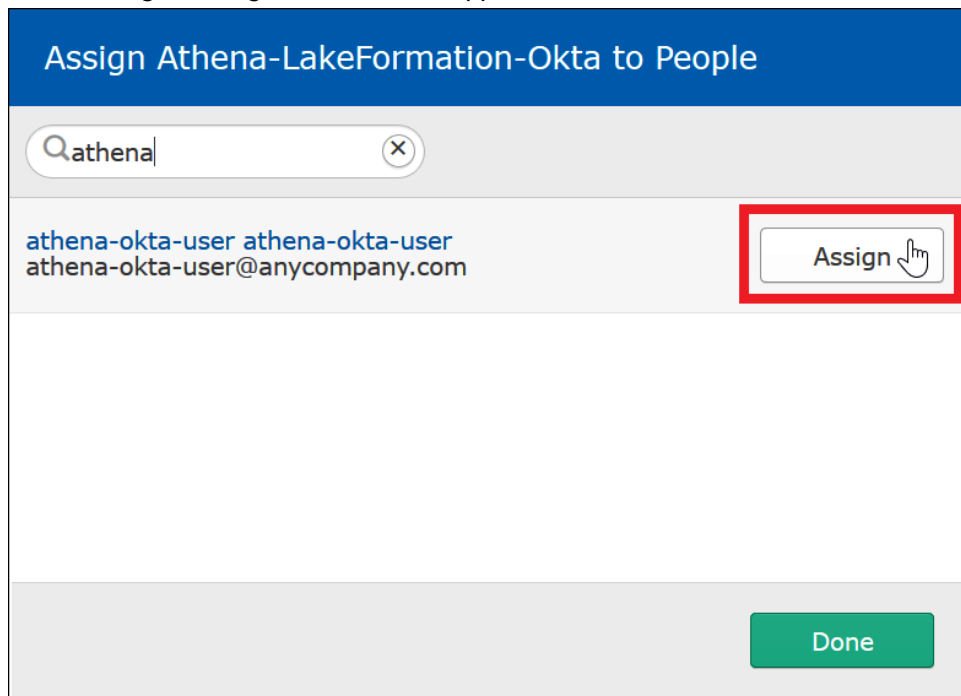
### To assign the application to users and groups

1. On the application **Assignments** tab, choose **Assign, Assign to People**.

The screenshot shows the Okta application page for 'Athena-LakeFormation-Okta'. The 'Assignments' tab is selected and highlighted with a red box. Above the tabs, there is a 'Back to Applications' link, the application logo, an 'Active' status dropdown, a 'View Logs' button, and a search bar. Below the tabs, there is an 'Assign' button (highlighted with a red box) and a 'Convert Assignments' button. A dropdown menu is open from the 'Assign' button, showing 'Assign to People' (highlighted with a red box) and 'Assign to Groups'. At the bottom, there is a 'Groups' link.



2. In the **Assign Athena-LakeFormation-Okta to People** dialog box, find the **athena-okta-user** user that you created previously.
3. Choose **Assign** to assign the user to the application.



Assign Athena-LakeFormation-Okta to People

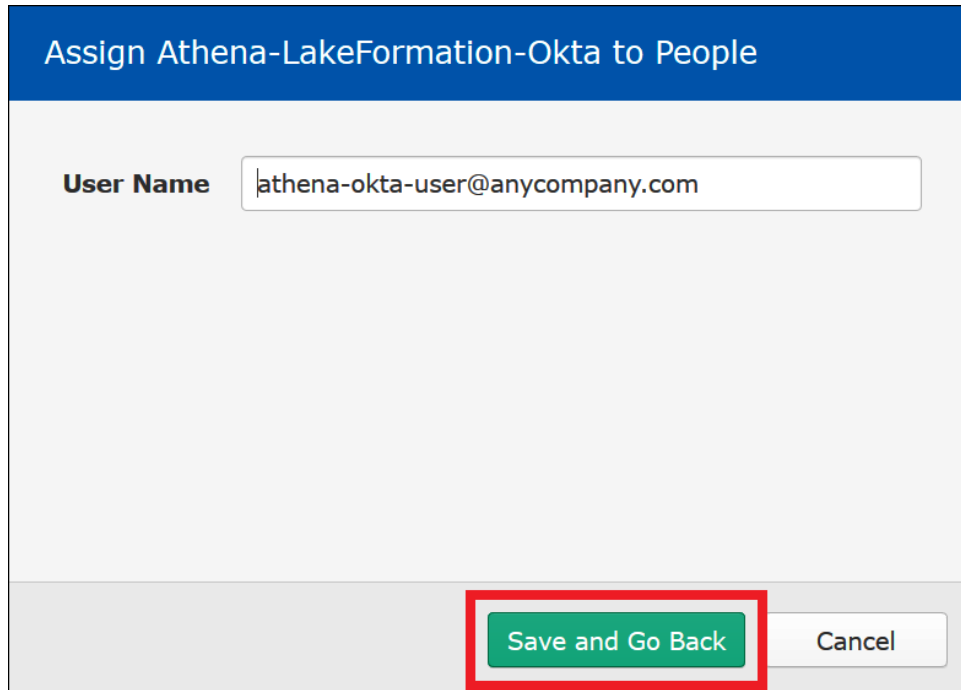
Search: athena

athena-okta-user athena-okta-user  
athena-okta-user@anycompany.com

Assign

Done

4. Choose **Save and Go Back**.

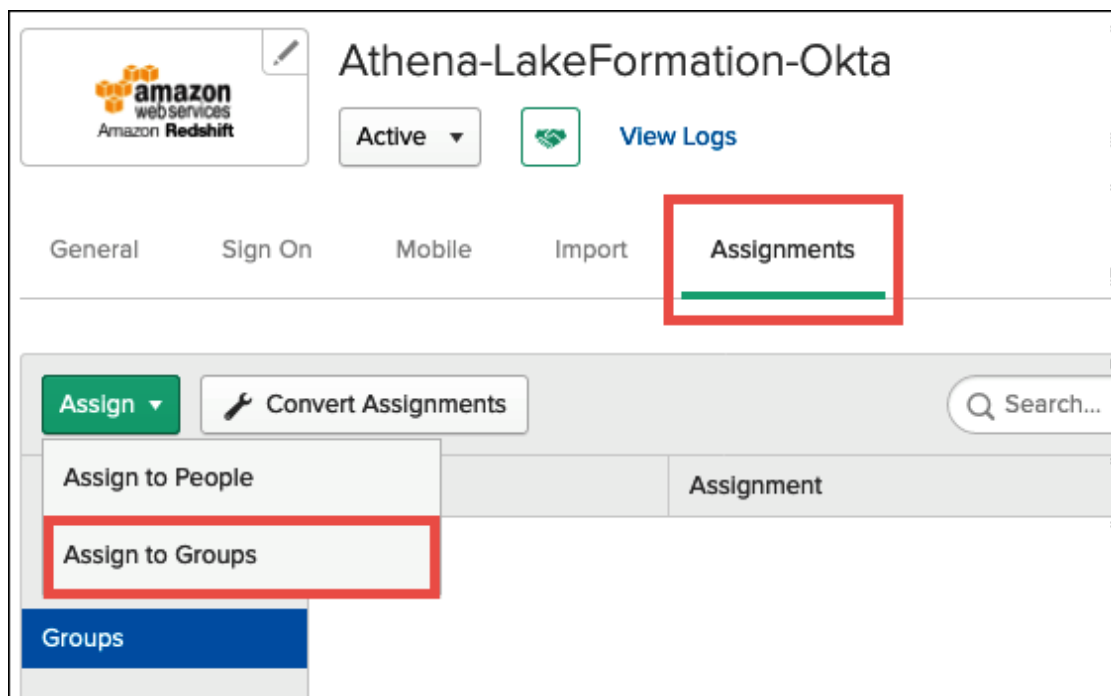


Assign Athena-LakeFormation-Okta to People

User Name: athena-okta-user@anycompany.com

Save and Go Back Cancel

5. Choose **Done**.
6. On the **Assignments** tab for the **Athena-LakeFormation-Okta** application, choose **Assign, Assign to Groups**.



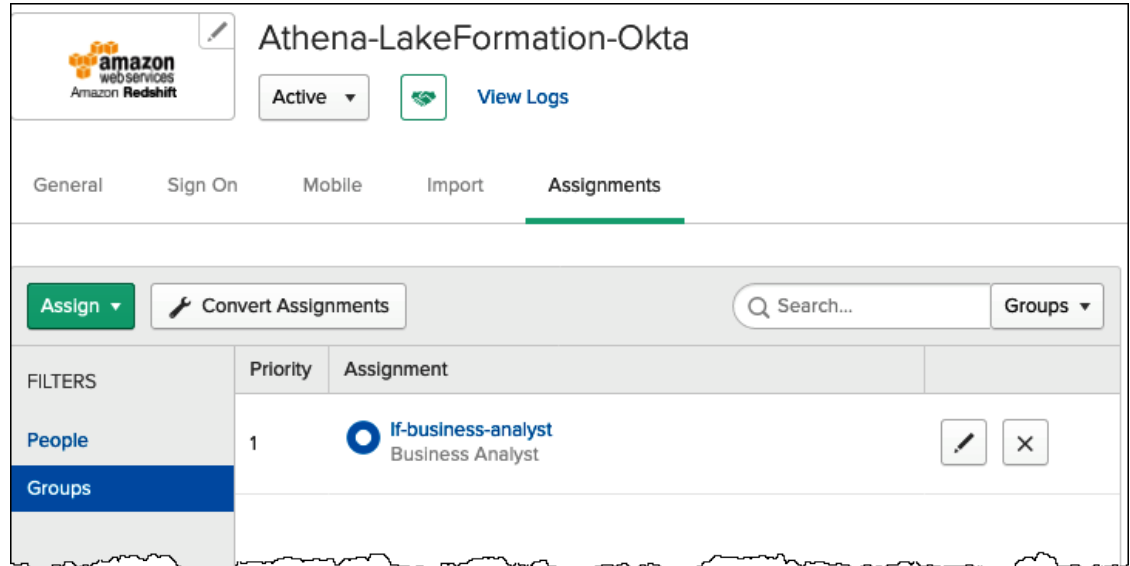
7. For **lf-business-analyst**, choose **Assign** to assign the **Athena-LakeFormation-Okta** application to the **lf-business-analyst** group, and then choose **Done**.

### Assign Athena-LakeFormation-Okta to Groups

<input type="radio"/>	<b>Everyone</b> All users in your organization	Assign
<input type="radio"/>	<b>If-business-analyst</b> Business Analyst	Assign
<input type="radio"/>	<b>If-developer</b> Developer Group	Assign

Done

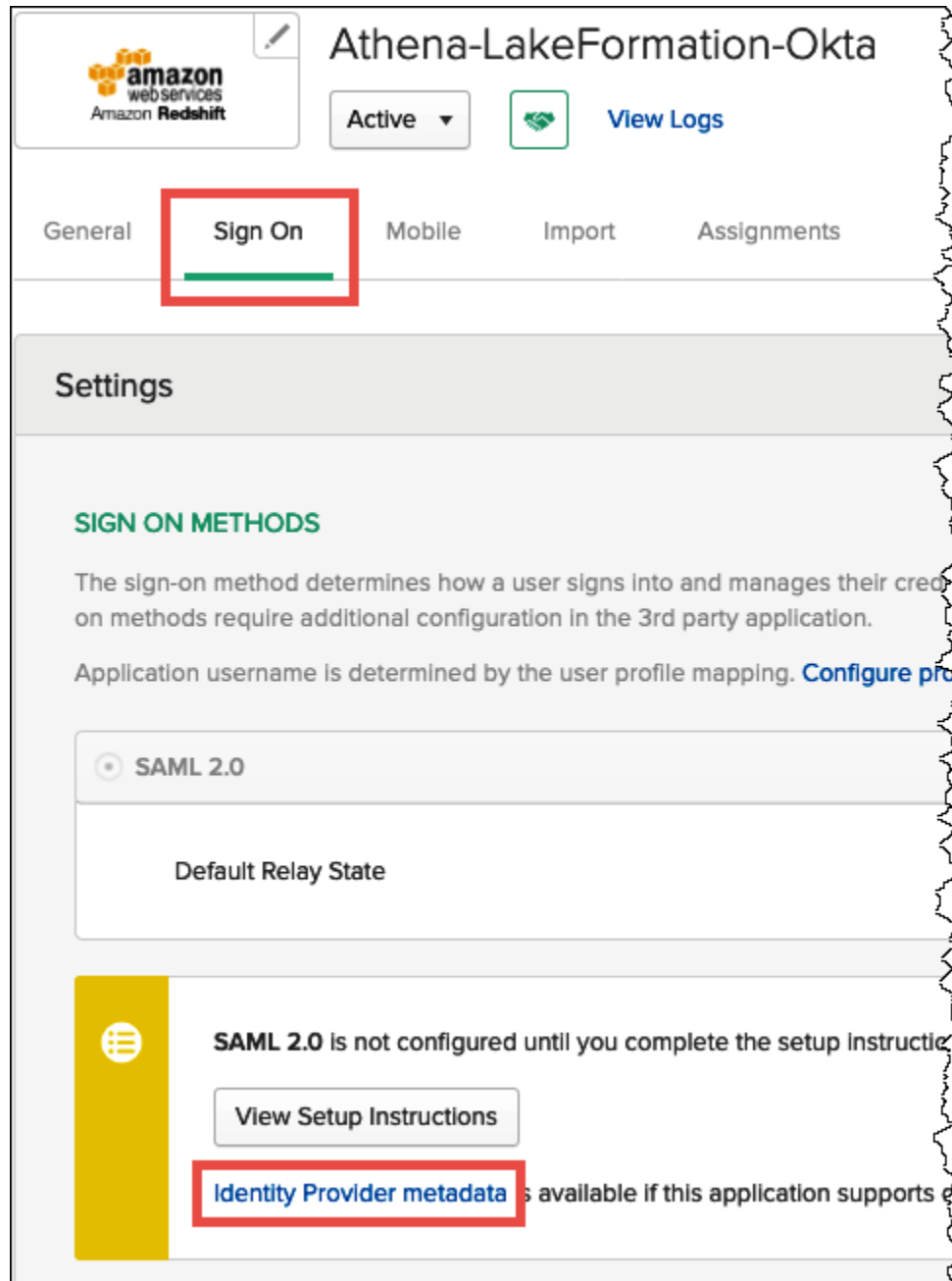
The group appears in the list of groups for the application.



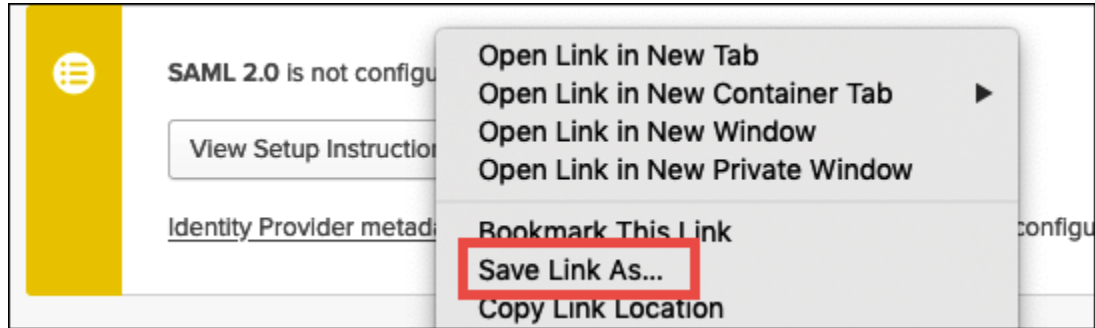
Now you are ready to download the identity provider application metadata for use with AWS.

#### 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`).



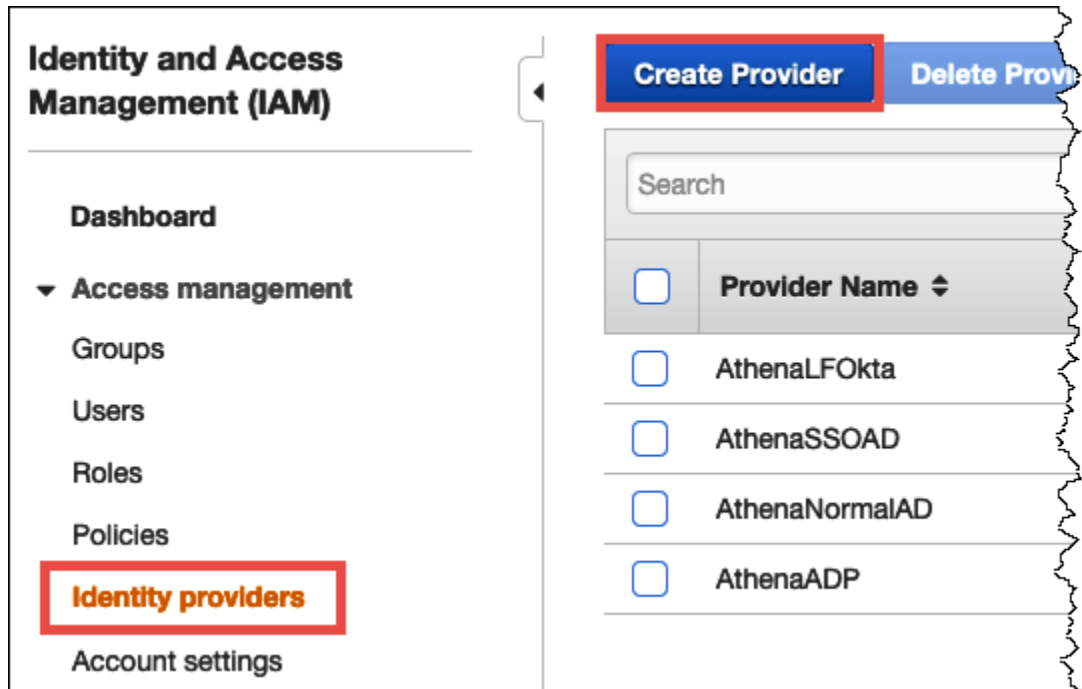
## Step 4: Create an AWS SAML Identity Provider and Lake Formation Access IAM Role

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 **AWS account console** as **AWS 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 **Create Provider**.



3. On the **Configure Provider** screen, enter the following information:

- For **Provider Type**, choose SAML.
- For **Provider Name**, enter `AthenaLakeFormationOkta`.
- For **Metadata Document**, choose the identity provider (IdP) metadata XML file that you downloaded.

## Configure Provider

Choose a provider type.

**Provider Type\*** SAML

**Provider Name\*** AthenaLakeFormationOkta  
Maximum 128 characters. Use alphanumeric and '.', '\_' characters.

**Metadata Document\*** C:\fakepath\Athena-Lakeformati Choose File

4. Choose **Next Step**.
5. On the **Verify Provider Information** page, choose **Create**.

Create Provider

[Step 1 : Configure Provider](#)

**Step 2 : Verify**

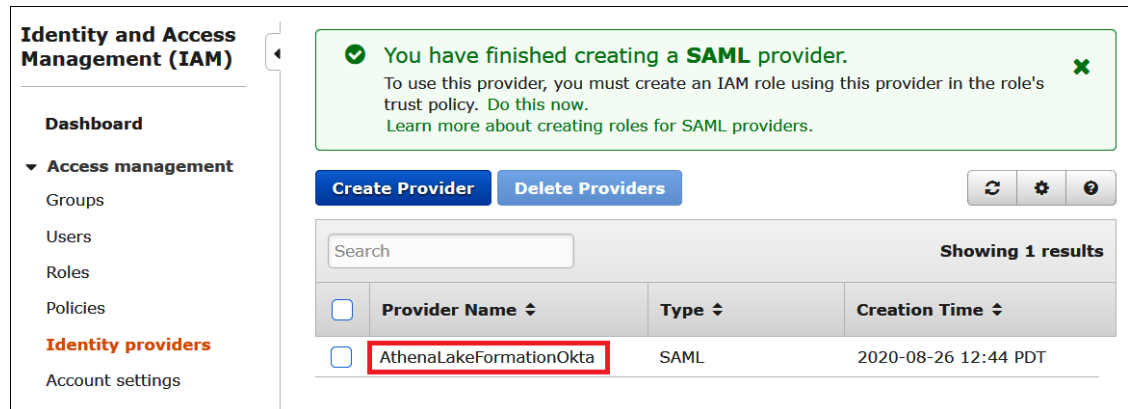
## Verify Provider Information

Verify the following provider information. Click **Create** to finish.

<b>Provider Name</b>	AthenaLakeFormationOkta
<b>Type</b>	SAML

[Cancel](#) [Previous](#) [Create](#)

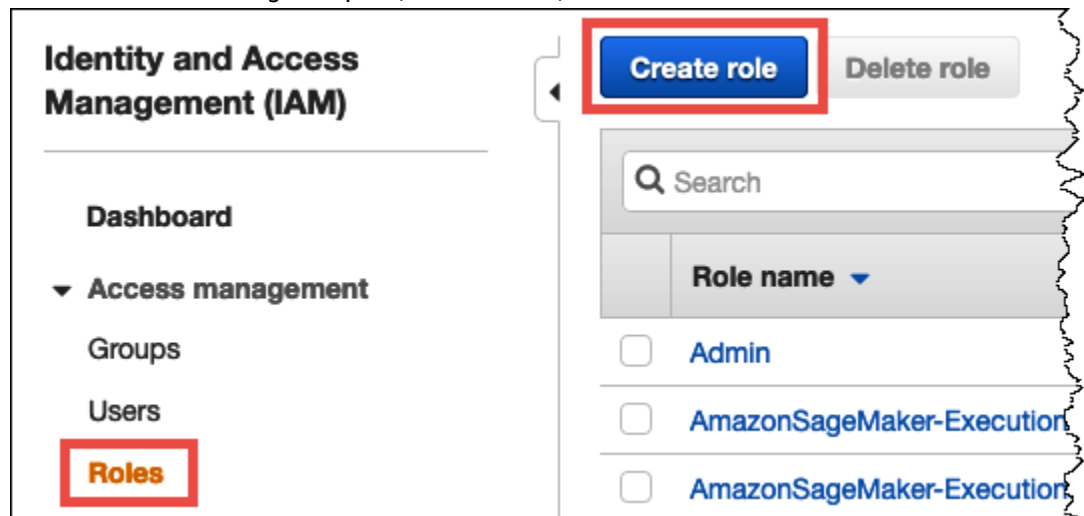
In the IAM console, the **AthenaLakeFormationOkta** provider that you created appears in the list of identity providers.



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**.





### Create role


1234

#### Select type of trusted entity

**AWS service**  
EC2, Lambda and others

**Another AWS account**  
Belonging to you or 3rd party

**Web identity**  
Cognito or any OpenID provider

**SAML 2.0 federation**  
Your corporate directory

Allows users that are federated with SAML 2.0 to assume this role to perform actions in your account. [Learn more](#)

#### Choose a SAML 2.0 provider

If you're creating a role for API access, choose an Attribute and then type a Value to include in the role. This restricts access to users with the specified attributes.

**SAML provider** AthenaLakeFormationOkta

[Create new provider](#) [Refresh](#)

☐ Allow programmatic access only

☒ Allow programmatic and AWS Management Console access

**Attribute** SAML:aud

**Value\*** https://signin.aws.amazon.com/saml

**Condition** [+ Add condition \(optional\)](#)

\* Required

[Cancel](#) [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**.

Create role

1234

▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies Athena Showing 2 results

	Policy name	Used as
<input checked="" type="checkbox"/>	AmazonAthenaFullAccess	Permissions policy (3)
<input type="checkbox"/>	AWSQuicksightAthenaAccess	None

► Set permissions boundary

\* Required Cancel Previous Next: Tags

- On the **Add tags** page, choose **Next: Review**.
- On the **Review** page, for **Role name**, enter a name for the role (for example, *Athena-LakeFormation-OktaRole*), and then choose **Create role**.

Create role

1234

Review

Provide the required information below and review this role before you create it.

Role name\* Athena-LakeFormation-OktaRole  
Use alphanumeric and '+,=,@,\_' characters. Maximum 64 characters.

Role description  
Maximum 1000 characters. Use alphanumeric and '+,=,@,\_' characters.

Trusted entities The identity provider(s) arn:aws:iam:::saml-provider/AthenaLakeFormationOkta

Policies AmazonAthenaFullAccess

Permissions boundary Permissions boundary is not set

No tags were added.

\* Required Cancel Previous Create role

Next, you add inline policies that allow access to Lake Formation, AWS Glue APIs, and Athena query results in Amazon S3.

### 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",
      "lakeformation:GetMetadataAccess",
      "glue:GetUnfiltered*",
      "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.

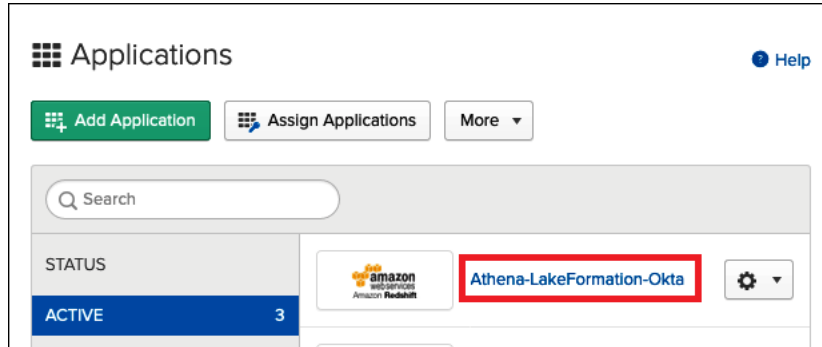
### Step 5: Add the IAM Role and SAML Identity Provider to the Okta Application

In this step, you return to the Okta developer console and perform the following tasks:

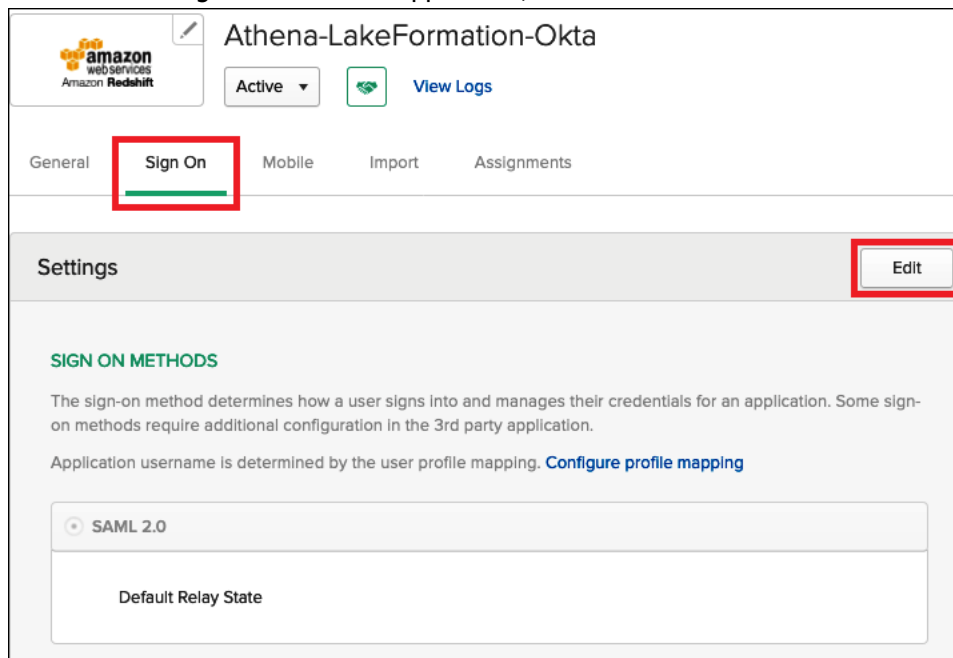
- Add user and group Lake Formation URL attributes to the Okta application.
- Add the ARN for the identity provider and the ARN for the IAM role to the Okta application.
- Copy the Okta application ID. The Okta application ID is required in the JDBC profile that connects to Athena.

#### 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. Expand **Attributes (optional)**.
5. Under **Attribute Statements (optional)**, add the following attribute:
- For **Name**, enter `https://lakeformation.amazonaws.com/SAML/Attributes/Username`.
  - For **Value**, enter `user.login`

**Athena-LakeFormation-Okta**

Active View Logs

General **Sign On** Mobile Import Assignments

**Settings** Cancel

**SIGN ON METHODS**

The sign-on method determines how a user signs into and manages their credentials for an application. Some sign-on methods require additional configuration in the 3rd party application.

Application username is determined by the user profile mapping. [Configure profile mapping](#)

**SAML 2.0**

Default Relay State

All IDP-initiated requests will include this RelayState.

☒ **Attributes (Optional)** [Learn More](#)

Attribute Statements (optional)

Name	Name format (optional)	Value
IL/Attributes/Username	Unspecified	user.login

Add Another

6. Under **Group Attribute Statements (optional)**, add the following attribute:
- For **Name**, enter `https://lakeformation.amazonaws.com/SAML/Attributes/Groups`.
  - For **Name format**, enter **Basic**
  - For **Filter**, choose **Matches regex**, and then enter `.*` in the filter box.

SAML 2.0

Default Relay State

All IDP-Initiated requests will include this RelayState.

Attributes (Optional)

Learn More

Attribute Statements (optional)

Name	Name format (optional)	Value
https://lakeformation.ar	Unspecified	user.login

Add Another

Group Attribute Statements (optional)

Name	Name format (optional)	Filter
/SAML/Attributes/Groups	Basic	Matches regex .*

Add Another

Preview SAML

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
```

**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**

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**

Set the user's session duration in seconds here.  
Valid range is 900 to 43200.

**DB User Format (Redshift)**

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

**CREDENTIALS DETAILS**

**Application username format**

**Password reveal**☐ Allow users to securely see their password (Recommended)

 Password reveal is disabled, since this app is using SAML with no password.

**Save**

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.



The screenshot shows the Okta application settings page for an application named 'Athena-LakeFormation-Okta'. At the top left is the Amazon Redshift logo. To its right, the application name 'Athena-LakeFormation-Okta' is displayed, followed by an 'Active' status indicator and a 'View Logs' link. Below this is a horizontal navigation bar with tabs: 'General' (highlighted with a red box), 'Sign On', 'Mobile', 'Import', and 'Assignments'. The main content area is titled 'App Settings' and contains a table with the following information:

Application label	Athena-LakeFormation-Okta

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/`.

The screenshot shows the 'App Embed Link' section of the Okta application settings. It includes an 'Edit' button in the top right corner. The section is titled 'EMBED LINK' and contains the text: 'You can use the URL below to sign into Amazon Web Services Redshift from a portal or other location outside of Okta.' Below this text is a text input field containing the URL: `https://dev-                    .okta.com/home/amazon_aws_redshift/`. The portion of the URL after `amazon_aws_redshift/` is highlighted with a red box. Below the URL field, there are two sections: 'APPLICATION LOGIN PAGE' and 'APPLICATION ACCESS ERROR PAGE'. Each section has a description and two radio button options.

**APPLICATION LOGIN PAGE**

If someone who is not authenticated attempts to access this application, they will be redirected to a default login page or one that can be customized. An application level setting will override default URL settings and IdP routing rules for this app.

☒ Use the default organization login page.

☐ Use a custom login page for this application.

**APPLICATION ACCESS ERROR PAGE**

If someone who is not assigned to the application attempts to use an embed link, they will be redirected to a default error page or one that can be customized. An application level setting will override default URL settings.

☒ Use the error page setting on the [global settings](#) page

☐ Use a custom error page for this application

## Step 6: Grant user and group permissions through AWS Lake Formation

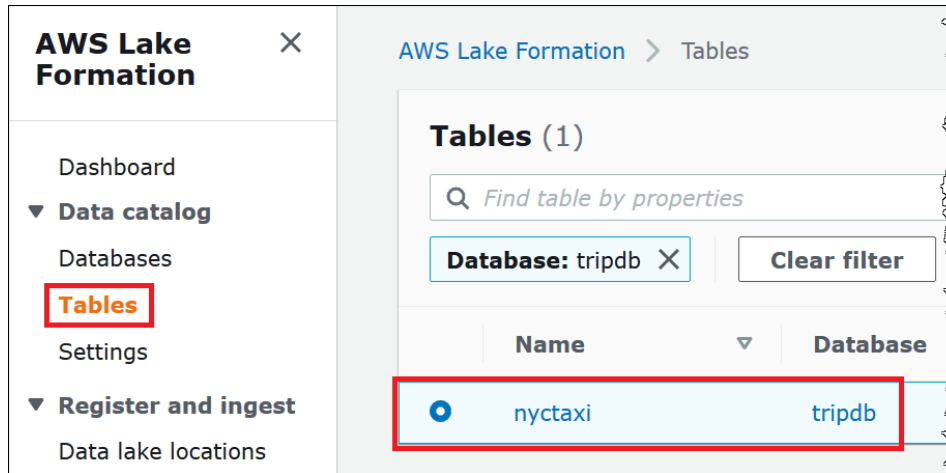
In this step, you use the Lake Formation console to grant permissions on a table to the SAML user and group. You perform the following tasks:

- Specify the ARN of the Okta SAML user and associated user permissions on the table.

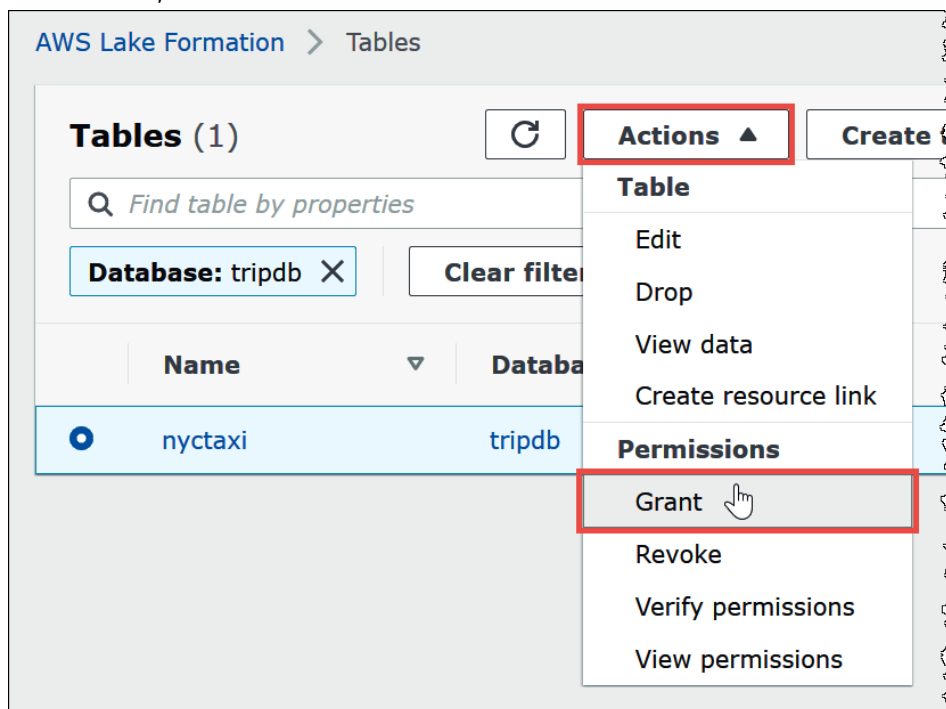
- Specify the ARN of the Okta SAML group and associated group permissions on the table.
- Verify the permissions that you granted.

### 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.

**Grant permissions: nyctaxi**  
Choose the access permissions to grant.

☒ **My account**  
User or role from this AWS account.

☐ **External account**  
AWS account or AWS organization outside of my account.

**IAM users and roles**  
Add one or more IAM users or roles.

**SAML and Amazon QuickSight users and groups**  
Enter a SAML user or group ARN or Amazon QuickSight ARN. Press Enter to add additional ARNs.  
**-provider/AthenaLakeFormationOkta:user/athena-okta-user@anycompany.com**

**Columns - optional**  
Choose filter type

**Table permissions**  
Choose the specific access permissions to grant.  
☐ Alter ☐ Insert ☐ Drop ☐ Delete ☒ **Select** ☐ Describe

☐ **Super**  
This permission is the union of the individual permissions above and supersedes them. [See here](#)

**Grantable permissions**  
Choose the permissions that may be granted to others.  
☐ Alter ☐ Insert ☐ Drop ☐ Delete ☐ Select ☐ Describe

☐ **Super**  
This permission allows the principal to grant any of the above permissions and supersedes those grantable permissions.

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.

☒ **My account**  
User or role from this AWS account.

☐ **External account**  
AWS account or AWS organization outside of my account.

**IAM users and roles**  
Add one or more IAM users or roles.

**SAML and Amazon QuickSight users and groups**  
Enter a SAML user or group ARN or Amazon QuickSight ARN. Press Enter to add additional ARNs.

**Columns - optional**  
Choose filter type

**Include columns**  
Grant permissions to access the selected columns.

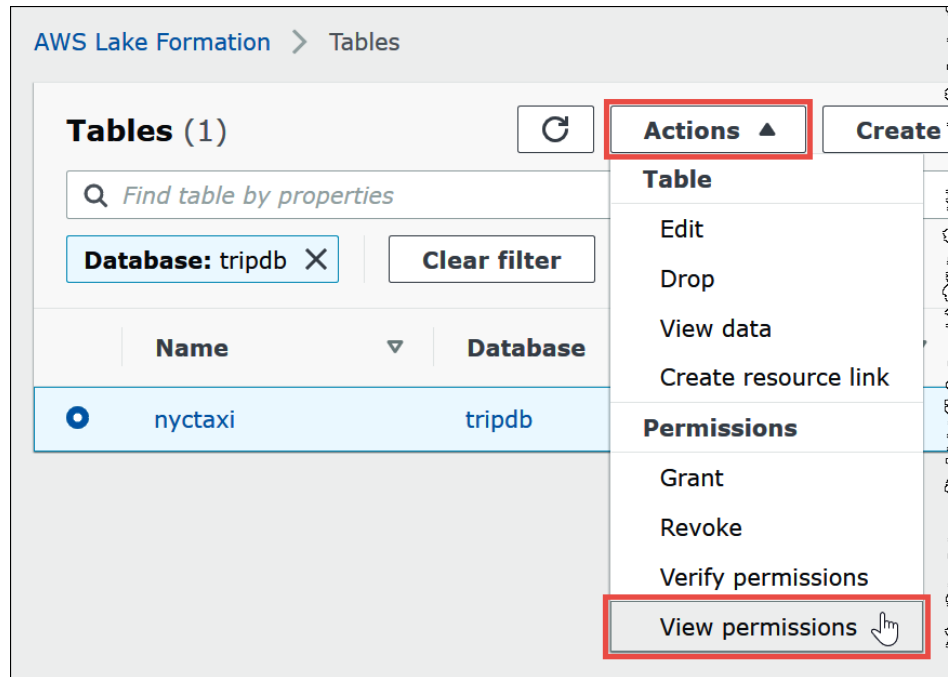
**Table permissions**  
Choose the specific access permissions to grant.  
☐ Alter ☐ Insert ☐ Drop ☐ Delete ☒ **Select**

☐ **Super**  
This permission is the union of the individual permissions above and supersedes them. [See here](#)

**Grantable permissions**  
Choose the permissions that may be granted to others.  
☐ Alter ☐ Insert ☐ Drop ☐ Delete ☐ Select

☐ **Super**  
This permission allows the principal to grant any of the above permissions and supersedes those grantable permissions.

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.

The screenshot shows the 'Data permissions (10)' page. It has a search bar 'Find by properties' and filters for 'Database: tripdb' and 'Table: nyctaxi'. Below the filters is a table with columns: Principal, Principal type, Resource type, Resource, and Permissions. The table contains two rows, both highlighted with a red border.

	Principal	Principal type	Resource type	Resource	Permissions
<input type="radio"/>	lf-business-analyst	AD group	Column	Include: tripdb.nyctaxi. [lpep_dropoff_dateti me, lpep_pickup_datetim e, vendorid]	Select
<input type="radio"/>	athena-okta- user@anycompany .com	AD user	Column	tripdb.nyctaxi.*	Select

## 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.

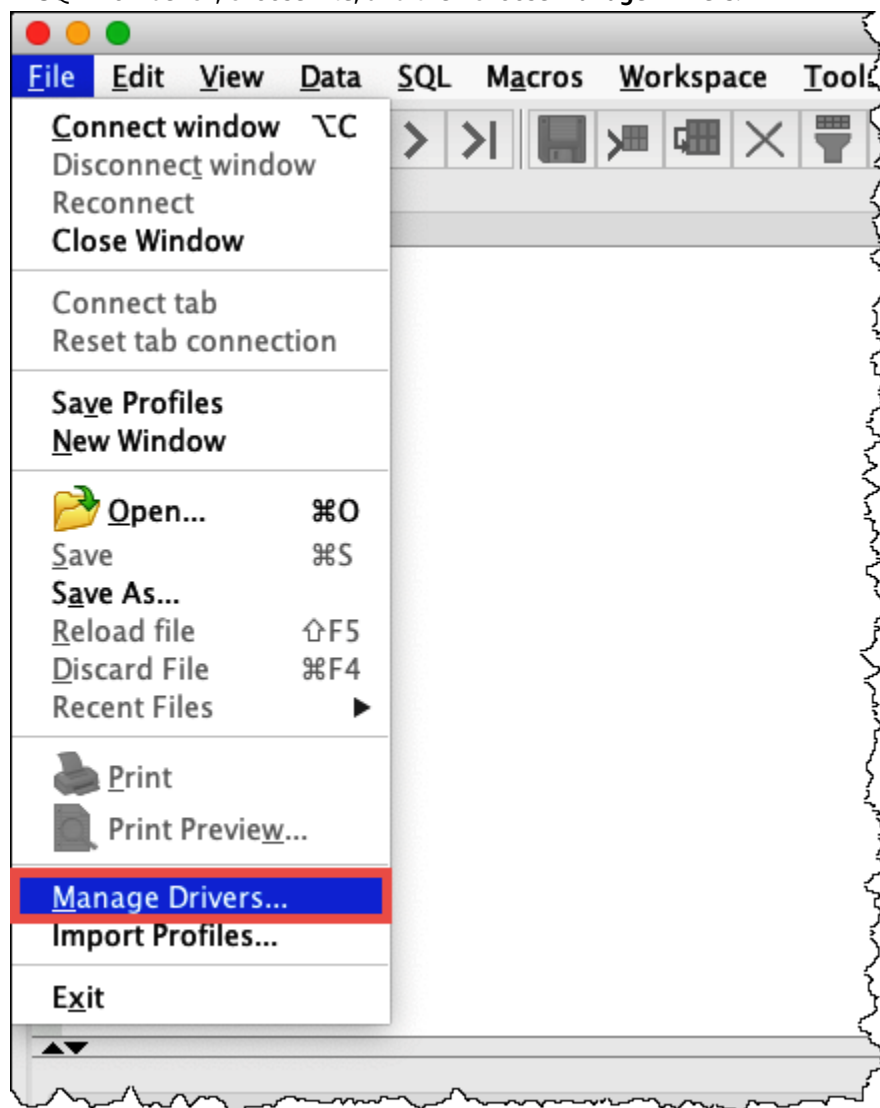
In this section, you perform the following tasks:

- 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.

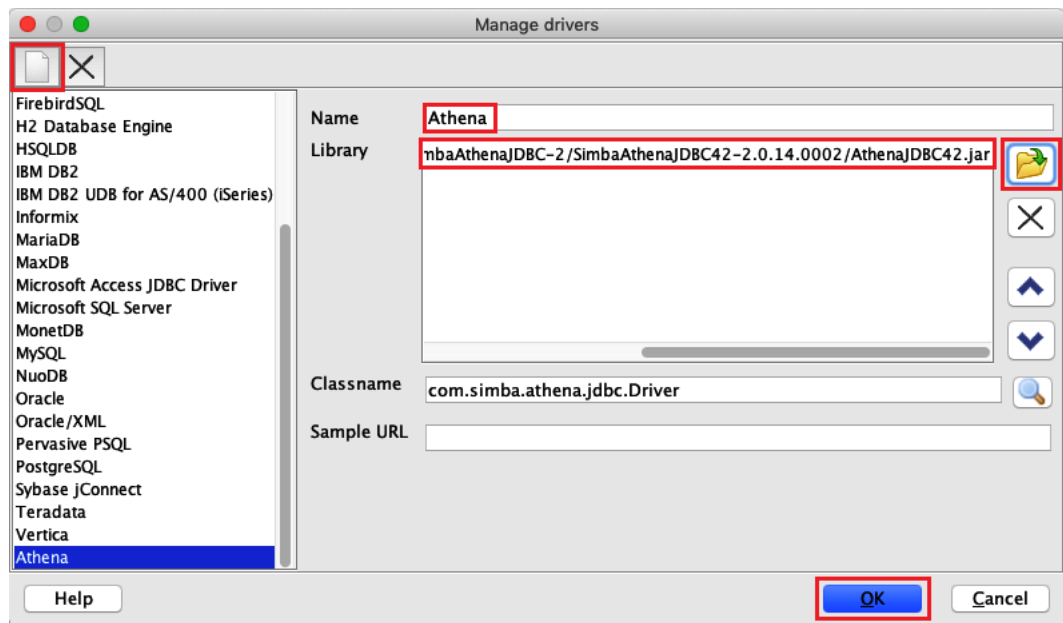
### To prepare the test client

1. Download and extract the Lake Formation compatible Athena JDBC driver (2.0.14 or later version) from [Using Athena with the JDBC Driver \(p. 72\)](#).
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.

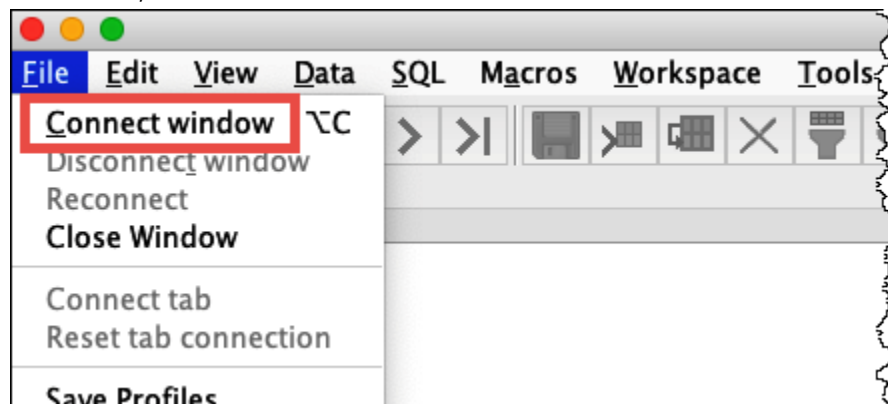
- b. For **Name**, enter **Athena**.
- c. For **Library**, browse to and choose the Simba Athena JDBC .jar file that you just downloaded.
- d. Choose **OK**.



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://athena-query-results-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.

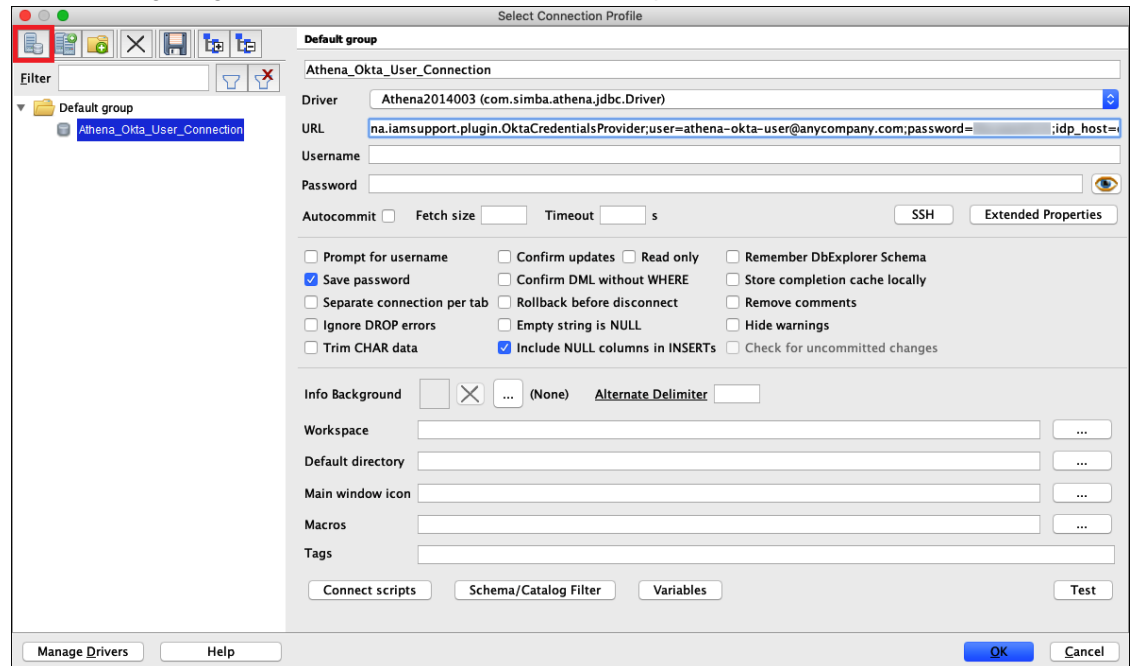
```
[athena_lf_dev]  
plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider  
idp_host=okta-idp-domain  
app_id=okta-app-id  
uid=athena-okta-user@anycompany.com  
pwd=password
```

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://athena-query-results-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 [Simba Athena JDBC driver installation guide \(p. 72\)](#). The JDBC installation guide also provides sample Java code for connecting to Athena programmatically.

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.

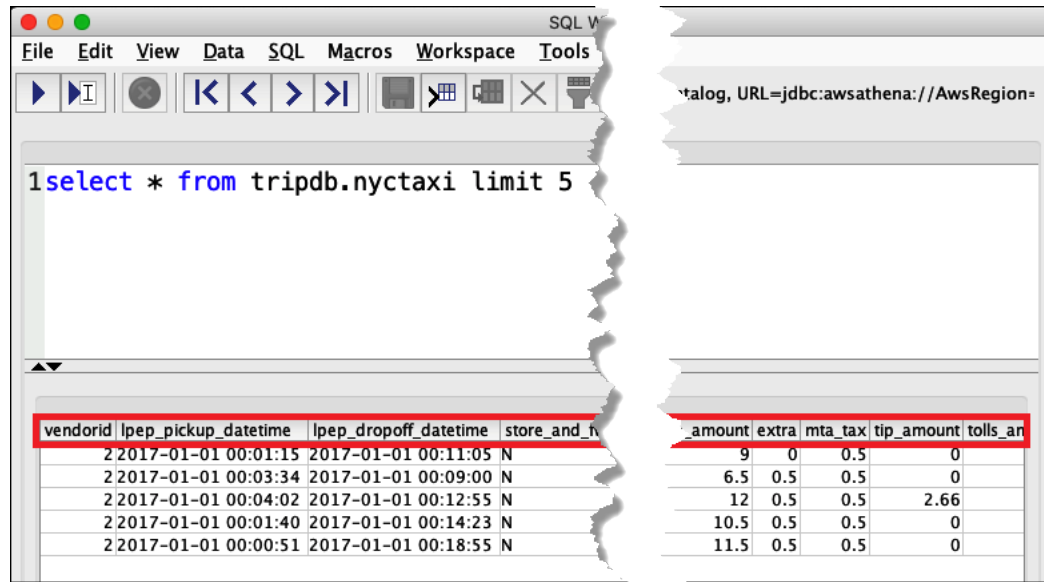
```
DESCRIBE "tripdb"."nyctaxi"
```

The screenshot shows the SQL Workbench interface with the title 'SQL Workbench/J Athena\_AD\_User\_Connection - Default.wksp'. The 'Statement 1' tab is active, displaying the command `1 describe "tripdb"."nyctaxi"`. Below the command, the 'Database Explorer 2' tab shows the results for 'tripdb.nyctaxi (EXTERNAL\_TABLE)'. A table of column metadata is displayed, with the first column, 'COLUMN\_NAME', highlighted by a red box. The table has 9 columns: COLUMN\_NAME, DATA\_TYPE, PK, NULLABLE, DEFAULT, AUTOINCREMENT, COMPUTED, REMARKS, and POSITION. It lists 19 columns from 'vendorid' to 'trip\_type'.

COLUMN_NAME	DATA_TYPE	PK	NULLABLE	DEFAULT	AUTOINCREMENT	COMPUTED	REMARKS	POSITION
vendorid	bigint	NO	YES		NO	NO		1
lpep_pickup_datetime	string(255)	NO	YES		NO	NO		2
lpep_dropoff_datetime	string(255)	NO	YES		NO	NO		3
store_and_fwd_flag	string(255)	NO	YES		NO	NO		4
ratecodeid	bigint	NO	YES		NO	NO		5
pulocationid	bigint	NO	YES		NO	NO		6
dolocationid	bigint	NO	YES		NO	NO		7
passenger_count	bigint	NO	YES		NO	NO		8
trip_distance	double	NO	YES		NO	NO		9
fare_amount	double	NO	YES		NO	NO		10
extra	double	NO	YES		NO	NO		11
mta_tax	double	NO	YES		NO	NO		12
tip_amount	double	NO	YES		NO	NO		13
tolls_amount	double	NO	YES		NO	NO		14
ehail_fee	string(255)	NO	YES		NO	NO		15
improvement_surcharge	double	NO	YES		NO	NO		16
total_amount	double	NO	YES		NO	NO		17
payment_type	bigint	NO	YES		NO	NO		18
trip_type	bigint	NO	YES		NO	NO		19

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://athena-query-results-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://athena-query-results-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.

The first screenshot shows the SQL Workbench interface with the query `1 describe tripdb.nyctaxi` entered in the SQL Statement window. Below the query, the results of the `DESCRIBE` command are displayed in a table. The table has columns: COLUMN\_NAME, DATA\_TYPE, PK, NULLABLE, DEFAULT, AUTOINCREMENT, COMPUTED, REMARKS, and POSITION. The first three columns are highlighted with a red box.

COLUMN_NAME	DATA_TYPE	PK	NULLABLE	DEFAULT	AUTOINCREMENT	COMPUTED	REMARKS	POSITION
vendorid	bigint	NO	YES		NO	NO		1
lpep_pickup_datetime	string(255)	NO	YES		NO	NO		2
lpep_dropoff_datetime	string(255)	NO	YES		NO	NO		3

The second screenshot shows the SQL Workbench interface with the query `1 select * from tripdb.nyctaxi limit 5` entered in the SQL Statement window. Below the query, the results of the `SELECT` command are displayed in a table. The table has columns: vendorid, lpep\_pickup\_datetime, and lpep\_dropoff\_datetime. The first three columns are highlighted with a red box.

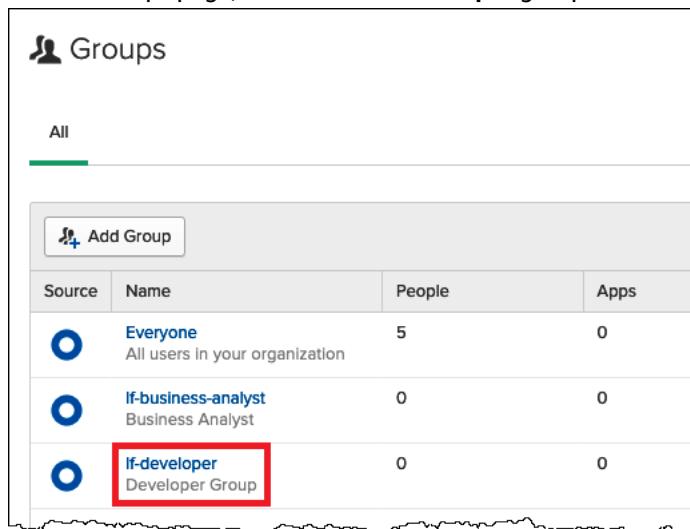
vendorid	lpep_pickup_datetime	lpep_dropoff_datetime
2	2017-01-01 00:01:15	2017-01-01 00:11:05
2	2017-01-01 00:03:34	2017-01-01 00:09:00
2	2017-01-01 00:04:02	2017-01-01 00:12:55
2	2017-01-01 00:01:40	2017-01-01 00:14:23
2	2017-01-01 00:00:51	2017-01-01 00:18:55

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. Switch to the **Classic UI**.
3. Choose **Directory**, and then choose **Groups**.
4. On the Groups page, choose the **lf-developer** group.



5. Choose **Manage People**.
6. From the **Not Members** list, choose the **athena-ba-user** to add it to the **lf-developer** group.
7. 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

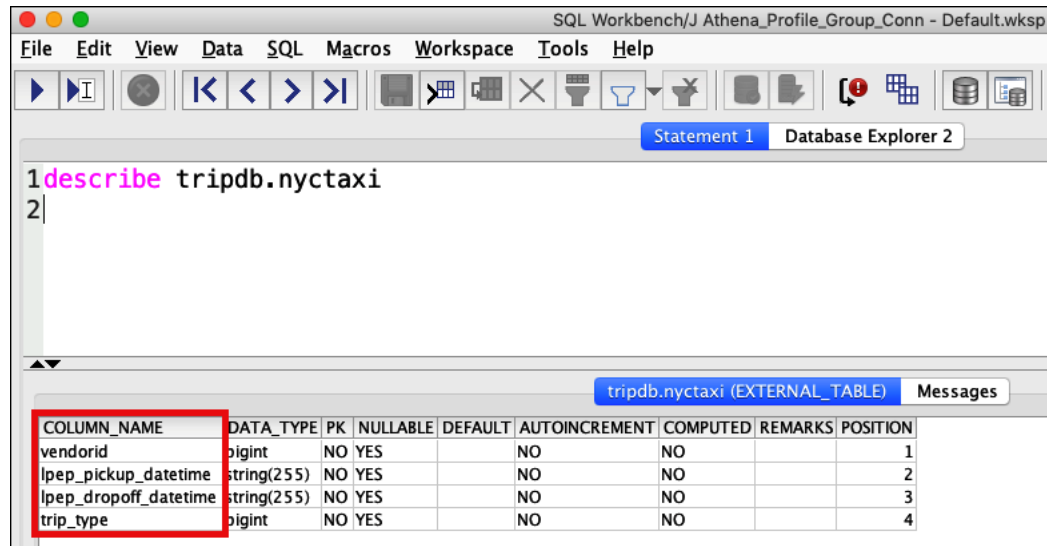
1. Log into the Lake Formation console as Data Lake administrator.
2. In the navigation pane, choose **Tables**.
3. Select the **nyctaxi** table.
4. Choose **Actions, Grant**.
5. In the **Grant Permissions** dialog, enter the following information:
  - For **SAML and Amazon QuickSight users and groups**, 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.

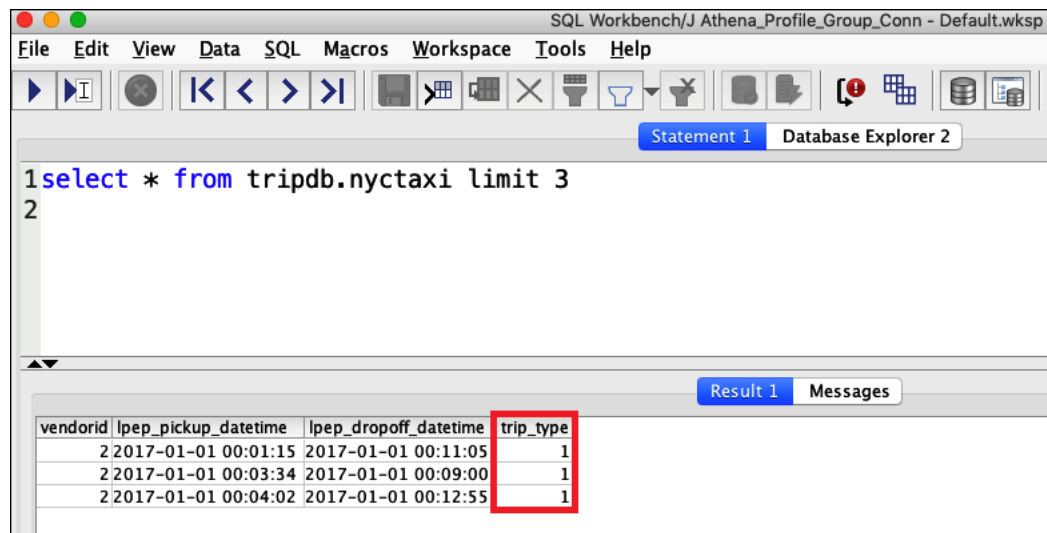
#### To verify the change in permissions for athena-ba-user

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.

- [Using Athena with the JDBC Driver \(p. 72\)](#)
- [Enabling Federated Access to the Athena API \(p. 268\)](#)
- [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*.

# Using Workgroups to Control Query Access and Costs

Use workgroups to separate users, teams, applications, or workloads, to set limits on amount of data each query or the entire workgroup can process, and to track costs. Because workgroups act as resources, you can use resource-level identity-based policies to control access to a specific workgroup. You can also view query-related metrics in Amazon CloudWatch, control costs by configuring limits on the amount of data scanned, create thresholds, and trigger actions, such as Amazon SNS, when these thresholds are breached.

Workgroups integrate with IAM, CloudWatch, and Amazon Simple Notification Service as follows:

- IAM identity-based policies with resource-level permissions control who can run queries in a workgroup.
- Athena publishes the workgroup query metrics to CloudWatch, if you enable query metrics.
- In Amazon SNS, you can create Amazon SNS topics that issue alarms to specified workgroup users when data usage controls for queries in a workgroup exceed your established thresholds.

## Topics

- [Using Workgroups for Running Queries \(p. 322\)](#)
- [Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events \(p. 338\)](#)

## Using Workgroups for Running Queries

We recommend using workgroups to isolate queries for teams, applications, or different workloads. For example, you may create separate workgroups for two different teams in your organization. You can also 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. You can switch between workgroups.

## Topics

- [Benefits of Using Workgroups \(p. 322\)](#)
- [How Workgroups Work \(p. 323\)](#)
- [Setting up Workgroups \(p. 324\)](#)
- [IAM Policies for Accessing Workgroups \(p. 325\)](#)
- [Workgroup Settings \(p. 330\)](#)
- [Managing Workgroups \(p. 331\)](#)
- [Athena Workgroup APIs \(p. 336\)](#)
- [Troubleshooting Workgroups \(p. 336\)](#)

## Benefits of Using Workgroups

Workgroups allow you to:

Isolate users, teams, applications, or workloads into groups.	Each workgroup has its own distinct query history and a list of saved queries. For more information, see <a href="#">How Workgroups Work (p. 323)</a> .
---------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------

	For all queries in the workgroup, you can choose to configure workgroup settings. They include an Amazon S3 location for storing query results, and encryption configuration. You can also enforce workgroup settings. For more information, see <a href="#">Workgroup Settings (p. 330)</a> .
Enforce costs constraints.	<p>You can set two types of cost constraints for queries in a workgroup:</p> <ul style="list-style-type: none"><li>• <b>Per-query limit</b> is a threshold for the amount of data scanned for each query. Athena cancels queries when they exceed the specified threshold. The limit applies to each running query within a workgroup. You can set only one per-query limit and update it if needed.</li><li>• <b>Per-workgroup limit</b> is a threshold you can set for each workgroup for the amount of data scanned by queries in the workgroup. Breaching a threshold activates an Amazon SNS alarm that triggers an action of your choice, such as sending an email to a specified user. You can set multiple per-workgroup limits for each workgroup.</li></ul> <p>For detailed steps, see <a href="#">Setting Data Usage Control Limits (p. 344)</a>.</p>
Track query-related metrics for all workgroup queries in CloudWatch.	For each query that runs in a workgroup, if you configure the workgroup to publish metrics, Athena publishes them to CloudWatch. You can <a href="#">view query metrics (p. 339)</a> for each of your workgroups within the Athena console. In CloudWatch, you can create custom dashboards, and set thresholds and alarms on these metrics.

## How Workgroups Work

Workgroups in Athena have the following characteristics:

- By default, each account has a primary workgroup and the default permissions allow all authenticated users access to this workgroup. The primary workgroup cannot be deleted.
- Each workgroup that you create shows saved queries and query history only for queries that ran in it, and not for all queries in the account. This separates your queries from other queries within an account and makes it more efficient for you to locate your own saved queries and queries in history.
- Disabling a workgroup prevents queries from running in it, until you enable it. Queries sent to a disabled workgroup fail, until you enable it again.
- If you have permissions, you can delete an empty workgroup, and a workgroup that contains saved queries. In this case, before deleting a workgroup, Athena warns you that saved queries are deleted. Before deleting a workgroup to which other users have access, make sure its users have access to other workgroups in which they can continue to run queries.
- You can set up workgroup-wide settings and enforce their usage by all queries that run in a workgroup. The settings include query results location in Amazon S3 and encryption configuration.

### Important

When you enforce workgroup-wide settings, all queries that run in this workgroup use workgroup settings. This happens even if their client-side settings may differ from workgroup settings. For information, see [Workgroup Settings Override Client-Side Settings \(p. 330\)](#).

## Limitations for Workgroups

- You can create up to 1000 workgroups per Region in your account.
- The primary workgroup cannot be deleted.



- 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.

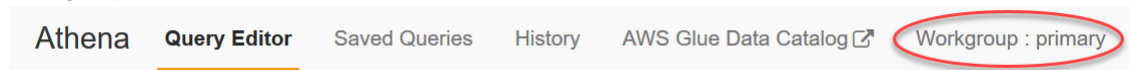
## Setting up Workgroups

Setting up workgroups involves creating them and establishing permissions for their usage. First, decide which workgroups your organization needs, and create them. Next, set up IAM workgroup policies that control user access and actions on a `workgroup` resource. Users with access to these workgroups can now run queries in them.

### Note

Use these tasks for setting up workgroups when you begin to use them for the first time. If your Athena account already uses workgroups, each account's user requires permissions to run queries in one or more workgroups in the account. Before you run queries, check your IAM policy to see which workgroups you can access, adjust your policy if needed, and [switch \(p. 335\)](#) to a workgroup you intend to use.

By default, if you have not created any workgroups, all queries in your account run in the primary workgroup:



Workgroups display in the Athena console in the **Workgroup:<workgroup\_name>** tab. The console lists the workgroup that you have switched to. When you run queries, they run in this workgroup. You can run queries in the workgroup in the console, or by using the API operations, the command line interface, or a client application through the JDBC or ODBC driver. When you have access to a workgroup, you can view workgroup's settings, metrics, and data usage control limits. Additionally, you can have permissions to edit the settings and data usage control limits.

### To Set Up Workgroups

1. Decide which workgroups to create. For example, you can decide the following:
  - Who can run queries in each workgroup, and who owns workgroup configuration. This determines IAM policies you create. For more information, see [IAM Policies for Accessing Workgroups \(p. 325\)](#).
  - Which locations in Amazon S3 to use for the query results for queries that run in each workgroup. A location must exist in Amazon S3 *before* you can specify it for the workgroup query results. All users who use a workgroup must have access to this location. For more information, see [Workgroup Settings \(p. 330\)](#).
  - Which encryption settings are required, and which workgroups have queries that must be encrypted. We recommend that you create separate workgroups for encrypted and non-encrypted queries. That way, you can enforce encryption for a workgroup that applies to all queries that run in it. For more information, see [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#).
2. Create workgroups as needed, and add tags to them. Open the Athena console, choose the **Workgroup:<workgroup\_name>** tab, and then choose **Create workgroup**. For detailed steps, see [Create a Workgroup \(p. 332\)](#).
3. 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 detailed steps, see [IAM Policies for Accessing Workgroups \(p. 325\)](#). For example JSON policies, see [Workgroup Example Policies \(p. 254\)](#).
4. Set workgroup settings. Specify a location in Amazon S3 for query results and encryption settings, if needed. You can enforce workgroup settings. For more information, see [workgroup settings \(p. 330\)](#).

### Important

If you [override client-side settings \(p. 330\)](#), Athena will use the workgroup's settings. This affects queries that you run in the console, by using the drivers, the command line interface, or the API operations.

While queries continue to run, automation built based on availability of results in a certain Amazon S3 bucket may break. We recommend that you inform your users before overriding. After workgroup settings are set to override, you can omit specifying client-side settings in the drivers or the API.

5. Notify users which workgroups to use for running queries. Send an email to inform your account's users about workgroup names that they can use, the required IAM policies, and the workgroup settings.
6. Configure cost control limits, also known as data usage control limits, for queries and workgroups. To notify you when a threshold is breached, create an Amazon SNS topic and configure subscriptions. For detailed steps, see [Setting Data Usage Control Limits \(p. 344\)](#) and [Creating an Amazon SNS Topic](#) in the *Amazon Simple Notification Service Getting Started Guide*.
7. Switch to the workgroup so that you can run queries. To run queries, switch to the appropriate workgroup. For detailed steps, see [the section called "Specify a Workgroup in Which to Run Queries" \(p. 336\)](#).

## IAM Policies for Accessing Workgroups

To control access to workgroups, 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 workgroup policies, see [Workgroup Example Policies \(p. 326\)](#).

### 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 *IAM User Guide*.
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 [Workgroup Example Policies \(p. 326\)](#).
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 and specify the workgroup resources they can access.

For more information, see the following topics in 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 workgroup policies, see [Workgroup Example Policies](#) (p. 326).

For a complete list of Amazon Athena actions, see the API action names in the [Amazon Athena API Reference](#).

## Workgroup Example Policies

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

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 [Workgroup IAM Policies](#) (p. 325).

- [Example Policy for Full Access to All Workgroups](#) (p. 326)
- [Example Policy for Full Access to a Specified Workgroup](#) (p. 327)
- [Example Policy for Running Queries in a Specified Workgroup](#) (p. 328)
- [Example Policy for Running Queries in the Primary Workgroup](#) (p. 328)
- [Example Policy for Management Operations on a Specified Workgroup](#) (p. 329)
- [Example Policy for Listing Workgroups](#) (p. 329)
- [Example Policy for Running and Stopping Queries in a Specific Workgroup](#) (p. 329)
- [Example Policy for Working with Named Queries in a Specific Workgroup](#) (p. 329)

### 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:ListWorkGroups",
        "athena:GetExecutionEngine",
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
        "athena:GetCatalogs",
        "athena:GetNamespaces",
        "athena:GetTables",
        "athena:GetTable"
      ],
      "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"
      ],
      "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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "athena:ListWorkGroups",
        "athena:GetExecutionEngine",
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
        "athena:GetCatalogs",
        "athena:GetNamespaces",
        "athena:GetTables",
        "athena:GetTable"
      ],
      "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:GetWorkGroup"
      ],
      "Resource": [
        "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
      ]
    }
  ]
}
```

### Example Example Policy for Running Queries in the Primary Workgroup

In the following example, we use the policy that allows a particular user to run queries in the primary workgroup.

#### Note

We recommend that you add this policy to all users who are otherwise configured to run queries in their designated workgroups. Adding this policy 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 the following policy to a resource section of the [Example Policy for Running Queries in a Specified Workgroup](#) (p. 328).

```
"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`.

```
{
  "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:

```
{
  "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:

```
{
  "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:

```
{
  "Effect": "Allow",
  "Action": [
    "athena:CreateNamedQuery",
    "athena:GetNamedQuery",
    "athena>DeleteNamedQuery"
  ],
  "Resource": [
    "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
  ]
}
```

}

## Workgroup Settings

Each workgroup has the following settings:

- A unique name. It can contain from 1 to 128 characters, including alphanumeric characters, dashes, and underscores. After you create a workgroup, you cannot change its name. You can, however, create a new workgroup with the same settings and a different name.
- Settings that apply to all queries running in the workgroup. They include:
  - **A location in Amazon S3 for storing query results** for all queries that run in this workgroup. This location must exist before you specify it for the workgroup when you create it. For information on creating an Amazon S3 bucket, see [Create a Bucket](#).
  - **An encryption setting**, if you use encryption for all workgroup queries. You can encrypt only all queries in a workgroup, not just some of them. It is best to create separate workgroups to contain queries that are either encrypted or not encrypted.

In addition, you can [override client-side settings \(p. 330\)](#). Before the release of workgroups, you could specify results location and encryption options as parameters in the JDBC or ODBC driver, or in the **Properties** tab in the Athena console. These settings could also be specified directly via the API operations. These settings are known as "client-side settings". With workgroups, you can configure these settings at the workgroup level and enforce control over them. This spares your users from setting them individually. If you select the **Override Client-Side Settings**, queries use the workgroup settings and ignore the client-side settings.

If **Override Client-Side Settings** is selected, the user is notified on the console that their settings have changed. If workgroup settings are enforced this way, users can omit corresponding client-side settings. In this case, if you run queries in the console, the workgroup's settings are used for them even if any queries have client-side settings. Also, if you run queries in this workgroup through the command line interface, API operations, or the drivers, any settings that you specified are overwritten by the workgroup's settings. This affects the query results location and encryption. To check which settings are used for the workgroup, [view workgroup's details \(p. 334\)](#).

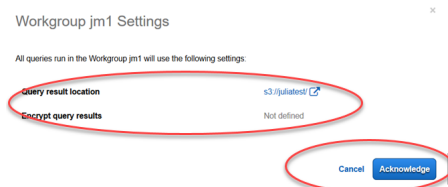
You can also [set query limits \(p. 338\)](#) for queries in workgroups.

## Workgroup Settings Override Client-Side Settings

The **Create workgroup** and **Edit workgroup** dialogs have a field titled **Override client-side settings**. This field is unselected 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. In this case, for all queries that run in this workgroup, Athena uses the clients-side settings for query results location and encryption. Each user can specify client-side settings in the **Settings** menu on the console. If the client-side settings are not used, the workgroup-wide settings apply, but are not enforced. Also, if you run queries in this workgroup through the API operations, the command line interface, or the JDBC and ODBC drivers, and specify your query results location and encryption there, your queries continue using those settings.
- If **Override client-side settings** is selected, Athena uses the workgroup-wide settings for query results location and encryption. It also overrides any other settings that you specified for the query in the console, by using the API operations, or with the drivers. This affects you only if you run queries in this workgroup. If you do, workgroup settings are used.

If you override client-side settings, then the next time that you or any workgroup user open the Athena console, the notification dialog box displays, as shown in the following example. It notifies you that queries in this workgroup use workgroup's settings, and prompts you to acknowledge this change.



### Important

If you run queries through the API operations, the command line interface, or the JDBC and ODBC drivers, and have not updated your settings to match those of the workgroup, your queries run, but use the workgroup's settings. For consistency, we recommend that you omit client-side settings in this case or update your query settings to match the workgroup's settings for the results location and encryption. To check which settings are used for the workgroup, [view workgroup's details \(p. 334\)](#).

## Managing Workgroups

In the <https://console.aws.amazon.com/athena/>, you can perform the following tasks:

Statement	Description
<a href="#">Create a Workgroup (p. 332)</a>	Create a new workgroup.
<a href="#">Edit a Workgroup (p. 333)</a>	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.
<a href="#">View the Workgroup's Details (p. 334)</a>	View the workgroup's details, such as its name, description, data usage limits, location of query results, and encryption. You can also verify whether this workgroup enforces its settings, if <b>Override client-side settings</b> is checked.
<a href="#">Delete a Workgroup (p. 334)</a>	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.
<a href="#">Switch between Workgroups (p. 335)</a>	Switch between workgroups to which you have access.
<a href="#">Enable and Disable a Workgroup (p. 335)</a>	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.
<a href="#">Specify a Workgroup in Which to Run Queries (p. 336)</a>	Before you can run queries, you must specify to Athena which workgroup to use. You must have permissions to the workgroup.

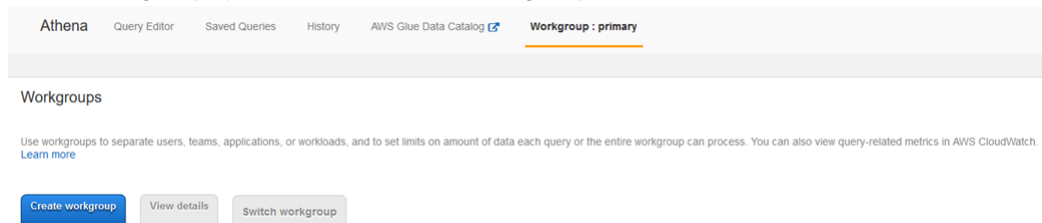


## Create a Workgroup

Creating a workgroup requires permissions to `CreateWorkgroup` API actions. See [Access to Athena Workgroups \(p. 254\)](#) and [IAM Policies for Accessing Workgroups \(p. 325\)](#). If you are adding tags, you also need to add permissions to `TagResource`. See [Tag Policy Examples for Workgroups \(p. 353\)](#).

### To create a workgroup in the console

1. In the Athena console, choose the **Workgroup:<workgroup\_name>** tab. A **Workgroups** panel displays.
2. In the **Workgroups** panel, choose **Create workgroup**.



3. In the **Create workgroup** dialog box, fill in the fields as follows:

Field	Description
<b>Workgroup name</b>	Required. Enter a unique name for your workgroup. Use 1 - 128 characters. (A-Z,a-z,0-9,-,_,.). This name cannot be changed.
<b>Description</b>	Optional. Enter a description for your workgroup. It can contain up to 1024 characters.
<b>Query result location</b>	Optional. Enter a path to an Amazon S3 bucket or prefix. This bucket and prefix must exist before you specify them.  <b>Note</b> If you run queries in the console, specifying the query results location is optional. If you don't specify it for the workgroup or in <b>Settings</b> , Athena uses the default query result location. If you run queries with the API or the drivers, you <i>must</i> specify query results location in at least one of the two places: for individual queries with <a href="#">OutputLocation</a> , or for the workgroup, with <a href="#">WorkGroupConfiguration</a> .
<b>Encrypt query results</b>	Optional. Encrypt results stored in Amazon S3. If selected, all queries in the workgroup are encrypted.  If selected, you can select the <b>Encryption type</b> , the <b>Encryption key</b> and enter the <b>KMS Key ARN</b> .  If you don't have the key, open the <a href="#">AWS KMS console</a> to create it. For more information, see <a href="#">Creating Keys</a> in the <i>AWS Key Management Service Developer Guide</i> .
<b>Publish to CloudWatch</b>	This field is selected by default. Publish query metrics to Amazon CloudWatch. See <a href="#">Viewing Query Metrics (p. 339)</a> .
<b>Override client-side settings</b>	This field is unselected by default. If you select it, workgroup settings apply to all queries in the workgroup and override client-side settings.

Field	Description
	For more information, see <a href="#">Workgroup Settings Override Client-Side Settings (p. 330)</a> .
<b>Tags</b>	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 <a href="#">best practices for AWS tagging strategies</a> 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 <a href="#">Tagging Resources (p. 348)</a> .
<b>Requester Pays S3 buckets</b>	Optional. Choose <b>Enable queries on Requester Pays buckets in Amazon S3</b> 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 <a href="#">Requester Pays Buckets</a> in the <i>Amazon Simple Storage Service Developer Guide</i> .

4. Choose **Create workgroup**. The workgroup appears in the list in the **Workgroups** panel.

Alternatively, use the API operations to create a workgroup.

### Important

After you create workgroups, create [IAM Policies for Workgroups \(p. 325\)](#) IAM that allow you to run workgroup-related actions.

## Edit a Workgroup

Editing a workgroup requires permissions to `UpdateWorkgroup` API operations. See [Access to Athena Workgroups \(p. 254\)](#) and [IAM Policies for Accessing Workgroups \(p. 325\)](#). If you are adding or editing tags, you also need to have permissions to `TagResource`. See [Tag Policy Examples for Workgroups \(p. 353\)](#).

### To edit a workgroup in the console

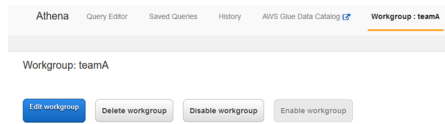
1. In the Athena console, choose the **Workgroup:<workgroup\_name>** tab. A **Workgroups** panel displays, listing all of the workgroups in the account.

The screenshot shows the Athena console interface. At the top, there's a navigation bar with tabs like 'Athena', 'Query Editor', 'Saved Queries', 'History', 'AWS Glue Data Catalog', and 'Workgroup: jm3'. Below this, the 'Workgroups' panel is displayed. It contains a sub-header 'Workgroups' and a description: '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. You can also view query-related metrics in AWS CloudWatch. [Learn more](#)'. Below the description are three buttons: 'Create workgroup', 'View details', and 'Switch workgroup'. The main part of the panel is a table listing workgroups.

	Name	Description	Creation time	Workgroup status
<input type="radio"/>	jm3	tags	2019/02/20 18:09:10 UTC-5	Enabled
<input type="radio"/>	teamB	This is the workgroup for queries by team B	2018/11/27 17:40:50 UTC-5	Enabled
<input type="radio"/>	teamA	This is the workgroup for team A queries	2018/11/27 17:39:59 UTC-5	Enabled
<input type="radio"/>	primary		2018/11/27 17:37:19 UTC-5	Enabled

2. In the **Workgroups** panel, choose the workgroup that you want to edit. The **View details** panel for the workgroup displays, with the **Overview** tab selected.

3. Choose **Edit workgroup**.



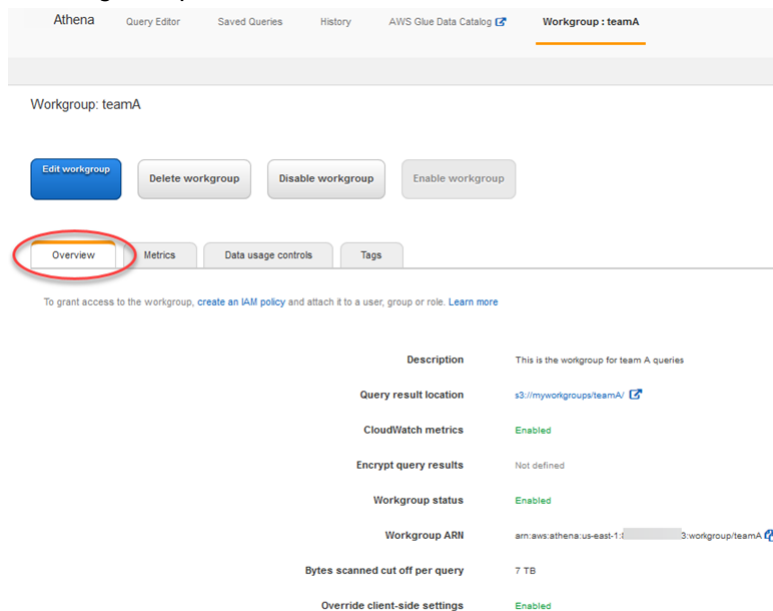
4. Change the fields as needed. For the list of fields, see [Create workgroup \(p. 332\)](#). You can change all fields except for the workgroup's name. If you need to change the name, create another workgroup with the new name and the same settings.
5. Choose **Save**. The updated workgroup appears in the list in the **Workgroups** panel.

## 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 and encryption configuration. If a workgroup has data usage limits, they are also displayed.

### To view the workgroup's details

- In the **Workgroups** panel, choose the workgroup that you want to edit. The **View details** panel for the workgroup displays, with the **Overview** tab selected. The workgroup details display, as in the following example:



## 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 \(p. 328\)](#).

### To delete a workgroup in the console

1. In the Athena console, choose the **Workgroup:<workgroup\_name>** tab. A **Workgroups** panel displays.
2. In the **Workgroups** panel, choose the workgroup that you want to delete. The **View details** panel for the workgroup displays, with the **Overview** tab selected.
3. Choose **Delete workgroup**, and confirm the deletion.

To delete a workgroup with the API operation, use the `DeleteWorkGroup` action.

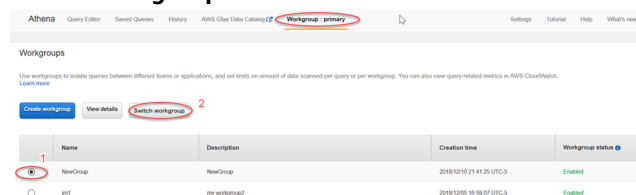
## Switch between Workgroups

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 between workgroups

1. In the Athena console, choose the **Workgroup:<workgroup\_name>** tab. A **Workgroups** panel displays.
2. In the **Workgroups** panel, choose the workgroup that you want to switch to, and then choose **Switch workgroup**.



3. Choose **Switch**. The console shows the **Workgroup: <workgroup\_name>** tab with the name of the workgroup that you switched to. You can now run queries in this workgroup.

## Enable and 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, choose the **Workgroup:<workgroup\_name>** tab. A **Workgroups** panel displays.
2. In the **Workgroups** panel, choose the workgroup, and then choose **Enable workgroup** or **Disable workgroup**. 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.

## Specify a Workgroup in Which to Run Queries

Before you can run queries, you must specify to Athena which workgroup to use. You need to have permissions to the workgroup.

### To specify a workgroup to Athena

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 "IAM Policies for Accessing Workgroups" \(p. 325\)](#).
2. To specify the workgroup to Athena, use one of these options:
  - If you are accessing Athena via the console, set the workgroup by [switching workgroups \(p. 335\)](#).
  - 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://<athena-output>-<AWSREGION>;
Workgroup=<WORKGROUPNAME>;
```

For more information, search for "Workgroup" in the driver documentation link included in [JDBC Driver Documentation \(p. 72\)](#).

## 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](#)

## Troubleshooting Workgroups

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 [IAM Policies for Accessing Workgroups \(p. 325\)](#).

- 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 in Which to Run Queries”](#) (p. 336).
- 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 *IAM User Guide*.

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 “Athena Workgroup APIs”](#) (p. 336).

- If the workgroup in which a query will run is configured with an [enforced query results location](#) (p. 330), 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://my_test/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: <code>arn:aws:iam::123456789012:user/abc</code> is not authorized to perform: <code>athena:StartQueryExecution</code> on resource: <code>arn:aws:athena:us-east-1:123456789012:workgroup/workgroupname</code>	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

Error	Occurs when...
	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 location provided. An output location is required either through the Workgroup result configuration setting or as an API input.	A user runs a query with the API without specifying the location for query results. You must set the output location for query results using one of the two ways: either for individual queries, using <a href="#">OutputLocation</a> (client-side), or in the workgroup, using <a href="#">WorkGroupConfiguration</a> .
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 <a href="#">enforced query results location</a> (p. 330), and you specify an external_location for the CTAS query. In this case, remove the external_location and rerun the query.

## Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events

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](#) (p. 338). When metrics are published, they are displayed under the **Metrics** tab in the **Workgroups** panel. Metrics are disabled by default for the primary workgroup.

### Topics

- [Enabling CloudWatch Query Metrics](#) (p. 338)
- [Monitoring Athena Queries with CloudWatch Metrics](#) (p. 339)
- [Monitoring Athena Queries with CloudWatch Events](#) (p. 342)
- [Setting Data Usage Control Limits](#) (p. 344)

## Enabling CloudWatch Query Metrics

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. Choose the **Workgroup** tab.
3. Choose the workgroup that you want to modify, and then choose **View details**.
4. Choose **Edit workgroup**.
5. On the **Edit workgroup** page, under **Metrics**, select or clear the **Publish query metrics to AWS CloudWatch** option.

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"  
    ....  
}
```

## Monitoring Athena Queries with CloudWatch Metrics

Athena publishes query-related metrics to Amazon CloudWatch, when **Publish to CloudWatch** 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:

- `EngineExecutionTime` – in milliseconds
- `ProcessedBytes` – the total amount of data scanned per DML query
- `QueryPlanningTime` – in milliseconds
- `QueryQueueTime` – in milliseconds
- `ServiceProcessingTime` – in milliseconds
- `TotalExecutionTime` – in milliseconds, for DDL and DML queries

These metrics have the following dimensions:

- `QueryState` – QUEUED, RUNNING, SUCCEEDED, FAILED, or CANCELLED
- `QueryType` – DML or DDL
- `WorkGroup` – name of the workgroup

For more information, see the [List of CloudWatch Metrics and Dimensions for Athena \(p. 341\)](#) later in this topic.

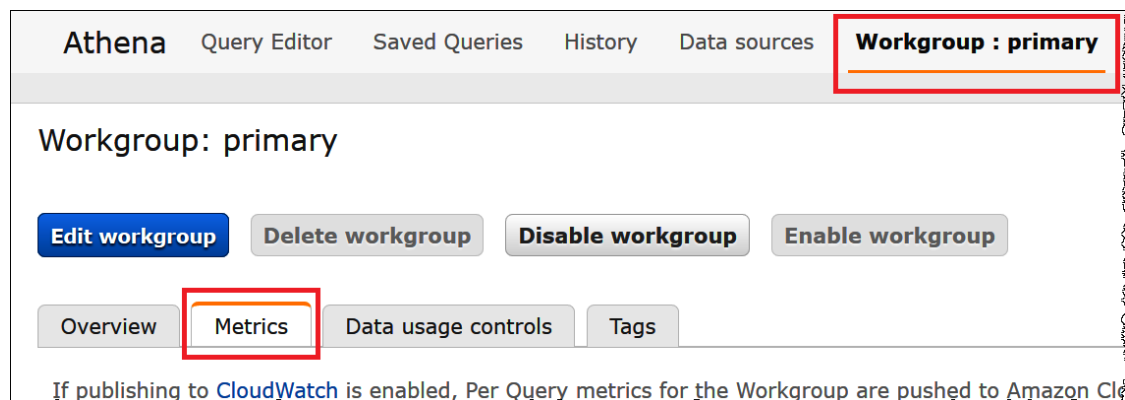
### To view query metrics for a workgroup in the console

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose the **Workgroup:<name>** tab.



To view a workgroup's metrics, you don't need to switch to it and can remain in another workgroup. You do need to select the workgroup from the list. You also must have permissions to view its metrics.

3. Select the workgroup from the list, and then choose **View details**. If you have permissions, the workgroup's details display in the **Overview** tab.
4. Choose the **Metrics** tab.

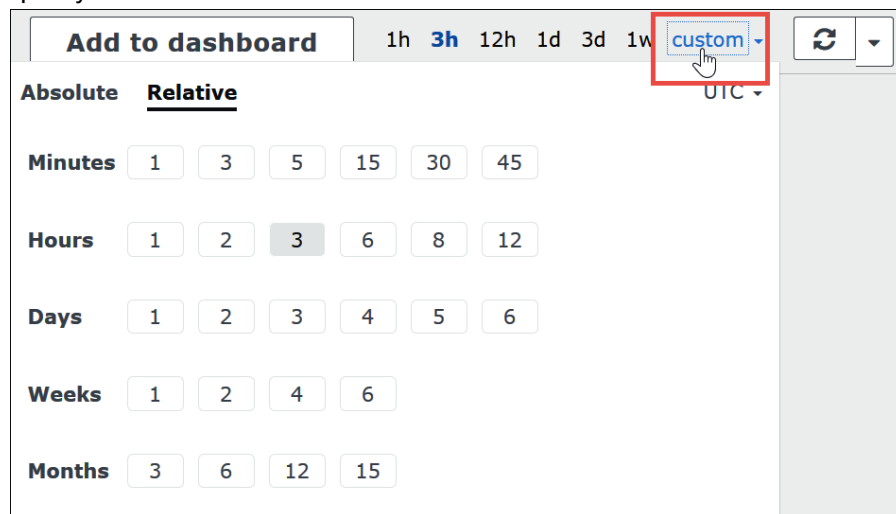


The metrics dashboard displays.

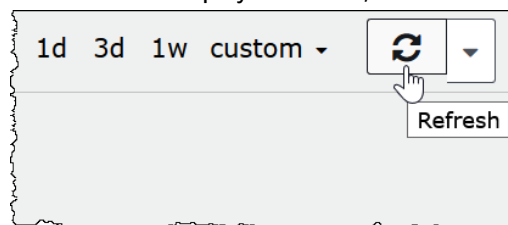
**Note**

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.

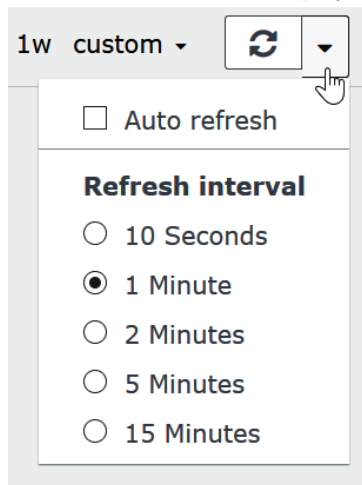
5. Choose the metrics interval that Athena should use to fetch the query metrics from CloudWatch, or specify a custom interval.



6. To refresh the displayed metrics, choose the refresh icon.



- Click the down arrow next to the refresh icon to choose the **Auto refresh** option and a refresh interval for the metrics display.



#### To view metrics in the Amazon CloudWatch console

- Open the Amazon CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
- In the navigation pane, choose **Metrics**.
- Select the **AWS/Athena** namespace.

#### To view metrics with the CLI

- 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 --namespace "AWS/Athena"
```

## 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 metrics use the **AWS/Athena** namespace.

Metric Name	Description
EngineExecutionTime	The number of milliseconds that the query took to run.
ProcessedBytes	The amount of data in megabytes 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 or CTAS queries.

Metric Name	Description
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.
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.
ServiceProcessingTime	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 ServiceProcessingTime.

CloudWatch metrics for Athena have the following dimensions.

Dimension	Description
QueryState	<p>The query state.</p> <p>Valid statistics: QUEUED, RUNNING, SUCCEEDED, FAILED, or CANCELLED.</p> <p><b>Note</b> Athena automatically retries your queries in cases of certain transient errors. As a result, you may see the query state transition from RUNNING or FAILED to QUEUED.</p>
QueryType	<p>The query type.</p> <p>Valid statistics: DDL or DML.</p>
WorkGroup	The name of the workgroup.

## Monitoring Athena Queries with CloudWatch Events

You can use Amazon Athena with Amazon CloudWatch to receive real-time notifications regarding the state of your queries. When a query you have submitted transitions states, Athena publishes an event to CloudWatch Events 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.

Before you create event rules for Athena, you should do the following:

- Familiarize yourself with events, rules, and targets in CloudWatch Events. For more information, see [What Is Amazon CloudWatch Events?](#) For more information about how to set up rules, see [Getting Started with CloudWatch Events](#).
- 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 is the format of an Athena Query State Change event.

```
{
  "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"
  }
}
```

## Output Properties

The JSON output includes the following properties.

Property	Description
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.

Property	Description
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 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 CloudWatch 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.

## Setting Data Usage Control Limits

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 \(p. 345\)](#).
- 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 choose to take one of the following actions:

- 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 \(p. 346\)](#). 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.
- 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, stopping any further queries from running.

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.

### 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 Developer 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. Choose the **Workgroup** tab.

To create a data usage control for a query in a particular workgroup, you don't need to switch to it and can remain in another workgroup. You do need to select the workgroup from the list and have permissions to edit the workgroup.

3. Select the workgroup from the list, and then choose **View details**. If you have permissions, the workgroup's details display in the **Overview** tab.
4. Choose the **Data usage controls** tab.

The screenshot shows the Amazon Athena console interface. At the top, there is a navigation bar with links: 'Athena', 'Query editor', 'Saved queries', 'History', 'Data sources', and 'Workgroup: TeamA'. The 'Workgroup: TeamA' link is highlighted with a red box. Below the navigation bar, the main content area is titled 'Workgroup: TeamA'. There are four buttons: 'Edit workgroup' (blue), 'Delete workgroup' (grey), 'Disable workgroup' (grey), and 'Enable workgroup' (grey). Below these buttons are four tabs: 'Overview', 'Metrics', 'Data usage controls' (highlighted with a red box), and 'Tags'. The 'Data usage controls' tab is active, showing the 'Per query data usage control' section. This section contains a text description: 'Sets the limit for the maximum amount of data a query is allowed to scan. You can set only one per query workgroup. The limit applies to all queries in the workgroup. [Learn more](#)'. Below this is a form with a 'Data limits' label, a text input field, a unit dropdown menu set to 'Megabytes MB', and a note 'Minimum Limit 10MB per query.'. Below the form is an 'Action' section with the text 'If the query exceeds the limit, it will be cancelled.' and two buttons: 'Delete' (grey) and 'Update' (blue). At the bottom of the console, there is a section titled 'Workgroup data usage controls'.

5. In the **Per query data usage control** section, specify the field values, as follows:
  - For **Data limits**, specify a value between 10000 KB (minimum) and 7000000 TB (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 (KB, MB, GB, or TB).
  - The default **Action** is to cancel the query if it exceeds the limit. This setting cannot be changed.
6. Choose **Create** if you are creating a new limit, or **Update** if you are editing an existing limit. If you are editing an existing limit, refresh the **Overview** tab to see the updated limit.

### To create a per-workgroup 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 control limits per workgroup. If the limit is exceeded, you can choose to take action, such as send an Amazon SNS alarm notification to the specified users.

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Choose the **Workgroup** tab.

To create a data usage control for a particular workgroup, you don't need to switch to it and can remain in another workgroup. You do need to select the workgroup from the list and have permissions to edit the workgroup.

3. Select the workgroup from the list, and then choose **View details**. If you have edit permissions, the workgroup's details display in the **Overview** tab.
4. Choose the **Data usage controls** tab, and scroll down. Then choose **Workgroup data usage controls** to create a new limit or edit an existing limit. The **Create workgroup data usage control** dialog displays.

**Create workgroup data usage control**

Sets the limit for the maximum amount of data queries running in this workgroup are allowed to scan within a specific period. The limit applies to all queries in the workgroup. You can set multiple limits per workgroup, and trigger different actions for each of them. Limits are implemented as [AWS CloudWatch alarms](#), and you can trigger [actions](#) when those alarms are breached. [Learn more](#)

**Data limits**  Terabytes

**Time period** 1 day

**Action** ☒ Send a notification to

[Create SNS topic](#)

5. Specify field values as follows:
  - For **Data limits**, specify a value between 10 MB (minimum) and 7000000 TB (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 time period, choose a time period from the drop-down list.
  - For **Action**, choose an Amazon SNS topic from the drop-down list, if you have one configured. Or, choose **Create an Amazon 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 [Creating an Amazon SNS Topic](#) in the *Amazon Simple Notification Service Getting Started Guide*.
6. Choose **Create** if you are creating a new limit, or **Save** if you are editing an existing limit. If you are editing an existing limit, refresh the **Overview** tab for the workgroup to see the updated limit.



# Tagging 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, workgroups and data catalogs 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. We recommend 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 \(p. 348\)](#)
- [Tag Restrictions \(p. 348\)](#)
- [Working with Tags on Workgroups in the Console \(p. 349\)](#)
- [Using Tag Operations \(p. 350\)](#)
- [Tag-Based IAM Access Control Policies \(p. 353\)](#)

## 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 AWS account have permission to create, edit, remove, or list tags. For more information, see [Tag-Based IAM Access Control Policies \(p. 353\)](#).

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 \(p. 348\)](#).

## 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 AWS account.

## Working with Tags on Workgroups in the Console

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

- [Displaying Tags for Individual Workgroups \(p. 349\)](#)
- [Adding and Deleting Tags on an Individual Workgroup \(p. 349\)](#)

## Displaying Tags for Individual Workgroups

You can display tags for an individual workgroup in the Athena console.

To view a list of tags for a workgroup, select the workgroup, choose **View Details**, and then choose the **Tags** tab. The list of tags for the workgroup displays. You can also view tags on a workgroup if you choose **Edit Workgroup**.

To search for tags, choose the **Tags** tab, and then enter a tag name into the search tool.

## Adding and Deleting 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 creating a new workgroup

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. On the navigation menu, choose the **Workgroups** tab.
3. Choose **Create workgroup** and fill in the values, as needed. For detailed steps, see [Create a Workgroup \(p. 332\)](#).
4. 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 \(p. 348\)](#).
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/>, choose the **Workgroups** tab, and select the workgroup.
2. Choose **View details**.
3. Do one of the following:
  - Choose the **Tags** tab, and then choose **Manage tags**.
  - Choose **Edit workgroup**, and then scroll down to the **Tags** section.
4. Specify the key and value for each tag. For more information, see [Tag Restrictions \(p. 348\)](#).
5. Choose **Save**.

### To delete a tag from an individual workgroup

1. Open the Athena console, and then choose the **Workgroups** tab.
2. In the workgroup list, select the workgroup, and then choose **View details**.
3. Do one of the following:
  - Choose the **Tags** tab, and then choose **Manage tags**.
  - Choose **Edit workgroup**, and then scroll down to the **Tags** section.
4. In the list of tags, select the **delete** button (x) for the tag that you want to delete, and then choose **Save**.

## Using 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.

### Adding Tags When Creating 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.

## Managing Tags Using API Operations

The examples in this section show how to use tag API operations 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();
```

## Managing Tags Using the AWS CLI

The following sections show how to use the AWS CLI to create and manage tags on data catalogs.

### Adding 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 on adding tags when using the `create-data-catalog` command, see [Registering a Catalog: create-data-catalog](#) (p. 50).

### Listing 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"  
    }  
  ]  
}
```

## Removing 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
```

# 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.

## Tag Policy Examples for Workgroups

### Example 1. 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:GetExecutionEngine",
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
        "athena:GetCatalogs",
        "athena:GetNamespaces",
        "athena:GetTables",
        "athena:GetTable"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "athena:StartQueryExecution",
        "athena:GetQueryResults",
        "athena>DeleteNamedQuery",
        "athena:GetNamedQuery",
        "athena:ListQueryExecutions",
        "athena:StopQueryExecution",
        "athena:GetQueryResultsStream",
        "athena:GetQueryExecutions",
        "athena:ListNamedQueries",
        "athena:CreateNamedQuery",
        "athena:GetQueryExecution",
        "athena:BatchGetNamedQuery",
        "athena:BatchGetQueryExecution",
        "athena:GetWorkGroup",
        "athena:TagResource",
        "athena:UntagResource",
        "athena:ListTagsForResource"
      ],
      "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
    }
  ]
}
```

### Example 2: 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`.

```
{
  "Effect": "Deny",
  "Action": [
    "athena:StartQueryExecution",
    "athena:GetQueryResults",
    "athena>DeleteNamedQuery",
    "athena:UpdateWorkGroup",
    "athena:GetNamedQuery",
    "athena:ListQueryExecutions",
    "athena:GetWorkGroup",
    "athena:StopQueryExecution",
    "athena:GetQueryResultsStream",
    "athena:GetQueryExecutions",
  ]
}
```

```
        "athena:ListNamedQueries",
        "athena:CreateNamedQuery",
        "athena:GetQueryExecution",
        "athena:BatchGetNamedQuery",
        "athena:BatchGetQueryExecution",
        "athena:TagResource",
        "athena:UntagResource",
        "athena:ListTagsForResource"
    ],
    "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/stack": "production"
        }
    }
}
```

### Example 3. 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 IAM users to pass in tags as part of a `CreateWorkGroup` operation, make sure that you give the users permissions to the `TagResource` and `CreateWorkGroup` actions.

```
{
  "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 1. 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:ListDataCatalogs",
        "athena:GetExecutionEngine",

```



```
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
        "athena:GetNamespaces",
        "athena:GetTables",
        "athena:GetTable"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "athena:StartQueryExecution",
        "athena:GetQueryResults",
        "athena:DeleteNamedQuery",
        "athena:GetNamedQuery",
        "athena:ListQueryExecutions",
        "athena:StopQueryExecution",
        "athena:GetQueryResultsStream",
        "athena:GetQueryExecutions",
        "athena:ListNamedQueries",
        "athena:CreateNamedQuery",
        "athena:GetQueryExecution",
        "athena:BatchGetNamedQuery",
        "athena:BatchGetQueryExecution",
        "athena:GetWorkGroup",
        "athena:TagResource",
        "athena:UntagResource",
        "athena:ListTagsForResource"
    ],
    "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",
        "athena:TagResource",
        "athena:UntagResource",
        "athena:ListTagsForResource"
    ],
    "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
}
]
```

### Example 2: 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`.

```
{
    "Effect": "Deny",
    "Action": [
        "athena:CreateDataCatalog",
```

```
    "athena:DeleteDataCatalog",
    "athena:GetDataCatalog",
    "athena:GetDatabase",
    "athena:GetTableMetadata",
    "athena:ListDatabases",
    "athena:ListTableMetadata",
    "athena:UpdateDataCatalog",
    "athena:StartQueryExecution",
    "athena:TagResource",
    "athena:UntagResource",
    "athena:ListTagsForResource"
  ],
  "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/stack": "production"
    }
  }
}
```

### Example 3. 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 IAM users to pass in tags as part of a `CreateDataCatalog` operation, make sure that you give the users permissions to the `TagResource` and `CreateDataCatalog` actions.

```
{
  "Effect": "Allow",
  "Action": [
    "athena:CreateDataCatalog",
    "athena:TagResource"
  ],
  "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/costcenter": [
        "1",
        "2",
        "3"
      ]
    }
  }
}
```

# Monitoring Logs and Troubleshooting

Examine Athena requests using CloudTrail logs and troubleshoot queries.

## Topics

- [Logging Amazon Athena API Calls with AWS CloudTrail \(p. 358\)](#)
- [Troubleshooting \(p. 361\)](#)

## Logging 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 also use Athena to query CloudTrail log files for insight. For more information, see [Querying AWS CloudTrail Logs \(p. 203\)](#) and [CloudTrail SerDe \(p. 367\)](#).

## Athena Information in CloudTrail

CloudTrail is enabled on your AWS 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 AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for 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](#).

## Understanding 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.

The following examples demonstrate CloudTrail log entries for:

- [StartQueryExecution \(Successful\)](#) (p. 359)
- [StartQueryExecution \(Failed\)](#) (p. 360)
- [CreateNamedQuery](#) (p. 360)

### 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://athena-johndoe-test/test/"
    },
    "queryString": "Select 10"
  },
  "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":"Select 10"  
  },  
  "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 boto3/1.5.48",  
  "requestParameters":{  
    "name":"johndoetest",  
    "queryString":"select 10",  
    "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"  
  },  
}
```

## Troubleshooting

Use these documentation topics to troubleshoot problems with Amazon Athena.

- [Service Quotas \(p. 438\)](#)
- [Considerations and Limitations for SQL Queries in Amazon Athena \(p. 421\)](#)
- [Unsupported DDL \(p. 400\)](#)
- [Names for Tables, Databases, and Columns \(p. 84\)](#)
- [Data Types in Amazon Athena \(p. 390\)](#)
- [Supported SerDes and Data Formats \(p. 363\)](#)
- [Compression Formats \(p. 388\)](#)
- [Reserved Keywords \(p. 85\)](#)
- [Troubleshooting Workgroups \(p. 336\)](#)

In addition, use the following AWS resources:

- [Athena topics in the AWS Knowledge Center](#)
- [Athena discussion forum](#)
- [Athena posts in the AWS Big Data Blog](#)

# SerDe Reference

Athena supports several SerDe libraries for parsing data from different data formats, such as CSV, JSON, Parquet, and ORC. Athena does not support custom SerDes.

## Topics

- [Using a SerDe \(p. 362\)](#)
- [Supported SerDes and Data Formats \(p. 363\)](#)
- [Compression Formats \(p. 388\)](#)

## Using a SerDe

A SerDe (Serializer/Deserializer) is a way in which Athena interacts with data in various formats.

It is the SerDe you specify, and not the DDL, that defines the table schema. In other words, the SerDe can override the DDL configuration that you specify in Athena when you create your table.

## To Use a SerDe in Queries

To use a SerDe when creating a table in Athena, use one of the following methods:

- Use DDL statements to describe how to read and write data to the table and do not specify a `ROW FORMAT`, as in this example. This omits listing the actual SerDe type and the native `LazySimpleSerDe` is used by default.

In general, Athena uses the `LazySimpleSerDe` if you do not specify a `ROW FORMAT`, or if you specify `ROW FORMAT DELIMITED`.

```
ROW FORMAT
DELIMITED FIELDS TERMINATED BY ','
ESCAPED BY '\\'
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ':'
```

- Explicitly specify the type of SerDe Athena should use when it reads and writes data to the table. Also, specify additional properties in `SERDEPROPERTIES`, as in this example.

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ',',
  'collection.delim' = '|',
  'mapkey.delim' = ':',
  'escape.delim' = '\\'
)
```

## Supported SerDes and Data Formats

Athena supports creating tables and querying data from CSV, TSV, custom-delimited, and JSON formats; data from Hadoop-related formats: ORC, Apache Avro and Parquet; logs from Logstash, AWS CloudTrail logs, and Apache WebServer logs.

### 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 [Creating a Table from Query Results \(CTAS\)](#) (p. 124).

To create tables and query data in these formats in Athena, specify a serializer-deserializer class (SerDe) so that Athena knows which format is used and how to parse the data.

This table lists the data formats supported in Athena and their corresponding SerDe libraries.

A SerDe is a custom library that tells the data catalog used by Athena how to handle the data. You specify a SerDe type by listing it explicitly in the `ROW FORMAT` part of your `CREATE TABLE` statement in Athena. In some cases, you can omit the SerDe name because Athena uses some SerDe types by default for certain types of data formats.

### Supported Data Formats and SerDes

Data Format	Description	SerDe types supported in Athena
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.	<ul style="list-style-type: none"><li>Use the <a href="#">LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files</a> (p. 378) if your data does not include values enclosed in quotes.</li><li>Use the <a href="#">OpenCSVSerDe for Processing CSV</a> (p. 369) when your data includes quotes in values, or different separator or escape characters.</li></ul>
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 <a href="#">LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files</a> (p. 378) and specify the separator character as <code>FIELDS TERMINATED BY '\t'</code> .
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 <a href="#">LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files</a> (p. 378) 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.	<ul style="list-style-type: none"><li>Use the <a href="#">Hive JSON SerDe</a> (p. 375).</li><li>Use the <a href="#">OpenX JSON SerDe</a> (p. 375).</li></ul>
Apache Avro	A format for storing data in Hadoop that uses JSON-based schemas for record values.	Use the <a href="#">Avro SerDe</a> (p. 364).



Data Format	Description	SerDe types supported in Athena
ORC (Optimized Row Columnar)	A format for optimized columnar storage of Hive data.	Use the <a href="#">ORC SerDe (p. 383)</a> and ZLIB compression.
Apache Parquet	A format for columnar storage of data in Hadoop.	Use the <a href="#">Parquet SerDe (p. 386)</a> and SNAPPY compression.
Logstash logs	A format for storing logs in Logstash.	Use the <a href="#">Grok SerDe (p. 372)</a> .
Apache WebServer logs	A format for storing logs in Apache WebServer.	Use the <a href="#">Grok SerDe (p. 372)</a> or <a href="#">Regex SerDe (p. 366)</a> .
CloudTrail logs	A format for storing logs in CloudTrail.	<ul style="list-style-type: none"> <li>• Use the <a href="#">CloudTrail SerDe (p. 367)</a> to query most fields in CloudTrail logs.</li> <li>• Use the <a href="#">OpenX JSON SerDe (p. 375)</a> for a few fields where their format depends on the service. For more information, see <a href="#">CloudTrail SerDe (p. 367)</a>.</li> </ul>

## Topics

- [Avro SerDe \(p. 364\)](#)
- [Regex SerDe \(p. 366\)](#)
- [CloudTrail SerDe \(p. 367\)](#)
- [OpenCSVSerDe for Processing CSV \(p. 369\)](#)
- [Grok SerDe \(p. 372\)](#)
- [JSON SerDe Libraries \(p. 374\)](#)
- [LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files \(p. 378\)](#)
- [ORC SerDe \(p. 383\)](#)
- [Parquet SerDe \(p. 386\)](#)

# Avro SerDe

## SerDe Name

Avro SerDe

## Library Name

`org.apache.hadoop.hive.serde2.avro.AvroSerDe`

## Examples

Athena does not support using `avro.schema.url` to specify table schema for security reasons. Use `avro.schema.literal`. To extract schema from data in the Avro format, use the Apache `avro-tools-<version>.jar` with the `getschema` parameter. This returns a schema that you can use in your `WITH SERDEPROPERTIES` statement. For example:

```
java -jar avro-tools-1.8.2.jar getschema my_data.avro
```

The `avro-tools-<version>.jar` file is located in the `java` subdirectory of your installed Avro release. 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 underlying Avro data stored in Amazon S3. In `ROW FORMAT`, you must specify the Avro SerDe as follows: `ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'`. As demonstrated in the following example, you must specify the schema using the `WITH SERDEPROPERTIES` clause in addition to specifying the column names and corresponding data types for the table.

**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.

## 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.

### SerDe Name

RegexSerDe

### Library Name

RegexSerDe

## Examples

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/`).

[illegible]

# CloudTrail SerDe

AWS CloudTrail is a service that records AWS API calls and events for AWS accounts. CloudTrail generates encrypted logs and stores them in Amazon S3. You can use Athena to query these logs directly from Amazon S3, specifying the `LOCATION` of logs.

To query CloudTrail logs in Athena, create table from the logs and use the CloudTrail SerDe to deserialize the logs data.

In addition to using the CloudTrail SerDe, instances exist where you need to use a different SerDe or to extract data from JSON. Certain fields in CloudTrail logs are STRING values that may have a variable data format, which depends on the service. As a result, the CloudTrail SerDe is unable to predictably deserialize them. To query the following fields, identify the data pattern and then use a different SerDe, such as the [OpenX JSON SerDe \(p. 375\)](#). Alternatively, to get data out of these fields, use `JSON_EXTRACT` functions. For more information, see [Extracting Data From JSON \(p. 184\)](#).

- requestParameters
- responseElements
- additionalEventData
- serviceEventDetails

## SerDe Name

## CloudTrail SerDe

## Library Name

```
com.amazon.emr.hive.serde.CloudTrailSerde
```

## Examples

The following example uses the CloudTrail SerDe on a fictional set of logs to create a table based on them.

In this example, the fields `requestParameters`, `responseElements`, and `additionalEventData` are included as part of `STRUCT` data type used in JSON. To get data out of these fields, use `JSON_EXTRACT` functions. For more information, see [Extracting Data From JSON \(p. 184\)](#).

```
CREATE EXTERNAL TABLE cloudtrail_logs (
  eventversion STRING,
  useridentity STRUCT<
    type:STRING,
    principalid:STRING,
    arn:STRING,
    accountid:STRING,
    invokedby:STRING,
    accesskeyid:STRING,
    userName:STRING,
  sessioncontext:STRUCT<
  attributes:STRUCT<
    mfaauthenticated:STRING,
    creationdate:STRING>,
  sessionIssuer:STRUCT<
    type:STRING,
    principalId:STRING,
    arn:STRING,
    accountId:STRING,
    userName: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
)
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://cloudtrail_bucket_name/AWSLogs/Account_ID/';
```

The following query returns the logins that occurred over a 24-hour period:

```
SELECT
  useridentity.username,
```

```
sourceipaddress,  
eventtime,  
additionaleventdata  
FROM default.cloudtrail_logs  
WHERE eventname = 'ConsoleLogin'  
      AND eventtime >= '2017-02-17T00:00:00Z'  
      AND eventtime < '2017-02-18T00:00:00Z';
```

For more information, see [Querying AWS CloudTrail Logs \(p. 203\)](#).

## OpenCSVSerDe for Processing CSV

When you create a table from CSV data in Athena, determine what types of values it contains:

- If data contains values enclosed in double quotes ("), you can use the [OpenCSV SerDe](#) to deserialize the values in Athena. In the following sections, note the behavior of this SerDe with `STRING` data types.
- If data does not contain values enclosed in double quotes ("), you can omit specifying any SerDe. In this case, Athena uses the default `LazySimpleSerDe`. For information, see [LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files \(p. 378\)](#).

### CSV SerDe (OpenCSVSerDe)

The [OpenCSV SerDe](#) behaves as follows:

- Converts all column type values to `STRING`.
- To recognize data types other than `STRING`, relies on the Presto parser and converts the values from `STRING` into those data types if it can recognize them.
- 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" = "\\")
```

- Cannot escape `\t` or `\n` directly. To escape them, use `"escapeChar" = "\\t"`. See the example in this topic.
- Does not support embedded line breaks in CSV files.
- Does not support empty fields in columns defined as a numeric data type.

#### Note

When you use Athena with `OpenCSVSerDe`, the SerDe converts all column types to `STRING`. Next, the parser in Athena parses the values from `STRING` into actual types based on what it finds. For example, it parses the values into `BOOLEAN`, `BIGINT`, `INT`, and `DOUBLE` data types when it can discern them. If the values are in `TIMESTAMP` in the UNIX format, Athena parses them as `TIMESTAMP`. If the values are in `TIMESTAMP` in Hive format, Athena parses them as `INT`. `DATE` type values are also parsed as `INT`.

To further convert columns to the desired type in a table, you can [create a view \(p. 119\)](#) over the table and use `CAST` to convert to the desired type.

For data types *other* than `STRING`, when the parser in Athena can recognize them, this SerDe behaves as follows:

- Recognizes `BOOLEAN`, `BIGINT`, `INT`, and `DOUBLE` data types and parses them without changes. The parser does not recognize empty or null values in columns defined as a numeric data type, leaving them as the default data type of `STRING`. The workaround is to declare the column as `STRING` and then `CAST` it in a `SELECT` query or view.

- Recognizes the `TIMESTAMP` type if it is specified in the UNIX numeric format, such as 1564610311.
- 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). If you are processing CSV data from Hive, use the UNIX numeric format.
- Recognizes the `DATE` type if it is specified in the UNIX numeric format, such as 1562112000.
- Does not support `DATE` in another format. If you are processing CSV data from Hive, use the UNIX numeric format.

#### Note

For information about using the `TIMESTAMP` and `DATE` columns when they are not specified in the UNIX numeric format, see the article [When I query a table in Amazon Athena, the `TIMESTAMP` result is empty](#) in the [AWS Knowledge Center](#).

#### Example Example: Using the `TIMESTAMP` type and `DATE` type specified in the UNIX numeric format.

Consider the following test data:

```
"unixvalue creationdate 18276 creationdatetime 1579146280000","18276","1579146280000"
```

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://<Location>'
```

Next, run the following query:

```
select * from testtimestamp1
```

The query returns the following result, showing the date and time data:

	profile_id	creationdate	creationdatetime	
1	unixvalue	creationdate 18276	creationdatetime 1579146280000	2020-01-15
	2020-01-16	03:44:40.000		

#### Example Example: Escaping `\t` or `\n`

Consider the following test data:

```
" \\t\\t\\t\\n 123 \\t\\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://user-test-region/dataset/test1/'
```

Next, run the following query:

```
select * from test1;
```

It returns this result, correctly escaping `\t` or `\n`:

f1	s2
<code>\t\t\n 123 \t\t\n</code>	abc
456	xyz

## SerDe Name

### CSV SerDe

### Library Name

To use this SerDe, specify its fully qualified class name in `ROW FORMAT`. Also specify the delimiters inside `SERDEPROPERTIES`, as follows:

```
...
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "~",
  "escapeChar"    = "\\"
)
```

## Example

This example presumes data in CSV saved in `s3://mybucket/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, and reference the `OpenCSVSerDe` class in `ROW FORMAT`, also specifying SerDe properties for character separator, quote character, and escape character, as follows:

```
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://location/of/csv/';
```

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

#### 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.

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. It is required.
- The `input.grokCustomPatterns` expression defines a named custom pattern, which you can subsequently use within the `input.format` expression. It 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

These examples rely on the list of predefined Grok patterns. See [pre-defined patterns](#).

### Example 1

This example uses source data from Postfix maillog entries saved in `s3://mybucket/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://mybucket/groksample/';
```

Start with a simple pattern, such as `%{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:amqListenerContainerPool23[P: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 logs 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://mybucket/samples/';
```

## Example 3

The following example of querying Amazon S3 logs 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://bucket-for-service-logs/s3_access/'
```

## JSON SerDe Libraries

In Athena, you can use two 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.

- The native [Hive JSON SerDe \(p. 375\)](#)
- The [OpenX JSON SerDe \(p. 375\)](#)

## SerDe Names

[Hive-JsonSerDe](#)

[Openx-JsonSerDe](#)

## Library Names

Use one of the following:

[org.apache.hive.hcatalog.data.JsonSerDe](#)

[org.openx.data.jsonserde.JsonSerDe](#)

## Hive JSON SerDe

The Hive JSON SerDe is commonly used to process JSON data like events. These events are represented as blocks of JSON-encoded text separated by a new line. The Hive JSON SerDe does not allow duplicate keys in map or struct key names.

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://myregion.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 (  
    requestbetime 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'  
with serdeproperties ( 'paths'='requestbetime, adid, impressionid, referrer, useragent,  
    usercookie, ip' )  
LOCATION 's3://myregion.elasticmapreduce/samples/hive-ads/tables/impressions';
```

After you create the table, run [MSCK REPAIR TABLE \(p. 415\)](#) to load the table and make it queryable from Athena:

```
MSCK REPAIR TABLE impressions
```

## OpenX JSON SerDe

In addition to the `paths` property that defines the columns in the table, the OpenX JSON SerDe has the following optional 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](#) (p. 85). 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")
```

Like the Hive JSON SerDe, the OpenX JSON SerDe does not allow duplicate keys in `map` or `struct` key names.

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 (
    requestbetime 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.openx.data.jsonserde.JsonSerDe'
with serdeproperties ( 'paths'='requestbegttime, addid, impressionid, referrer, useragent,
    usercookie, ip' )
LOCATION 's3://myregion.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. To parse JSON-encoded data in Athena, make sure that each JSON document is on its own line, separated by a new line.

This example presumes JSON-encoded data that has the following structure:

```
{
  "DocId": "AWS",
  "User": {
    "Id": 1234,
    "Username": "bob1234",
    "Name": "Bob",
  "ShippingAddress": {
    "Address1": "123 Main St.",
    "Address2": null,
    "City": "Seattle",
    "State": "WA"
  },
  "Orders": [
    {
      "ItemId": 6789,
      "OrderDate": "11/11/2017"
    },
    {
      "ItemId": 4352,
      "OrderDate": "12/12/2017"
    }
  ]
}
```

The following `CREATE TABLE` statement uses the [Openx-JsonSerDe](#) with the `struct` and array collection data types to establish groups of objects. Each JSON document is listed on its own line, separated by a new line. To avoid errors, the data being queried does not include duplicate keys in `struct` or map key names.

```
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://mybucket/myjsondata/';
```

## 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.

## LazySimpleSerDe for CSV, TSV, and Custom-Delimited Files

Specifying this SerDe is optional. This is the SerDe for data in CSV, TSV, and custom-delimited formats that Athena uses by default. This SerDe is used if you don't specify any SerDe and only specify `ROW FORMAT DELIMITED`. Use this SerDe if your data does not have values enclosed in quotes.

For reference documentation about the LazySimpleSerDe, see the [Hive SerDe](#) section of the Apache Hive Developer Guide.

## Library Name

The Class library name for the LazySimpleSerDe is `org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`. For information about the LazySimpleSerDe class, see [LazySimpleSerDe](#).

## 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, see the `CREATE TABLE` statements in [Querying Amazon VPC Flow Logs \(p. 216\)](#) and [Querying Amazon CloudFront Logs \(p. 202\)](#).

## Examples

The following examples show how to create tables in Athena from CSV and TSV, using the LazySimpleSerDe. To deserialize custom-delimited files using this SerDe, use the `FIELDS`

TERMINATED BY clause to specify a single-character delimiter, as in the following examples. LazySimpleSerDe does not support multi-character delimiters.

- [CSV Example \(p. 379\)](#)
- [TSV Example \(p. 381\)](#)

**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/`.

**Note**

The flight table data comes from [Flights](#) provided by US Department of Transportation, [Bureau of Transportation Statistics](#). Desaturated from original.

## CSV Example

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,  
  crsarrrtime 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,  
divlairport STRING,  
divlairportid INT,  
divlairportseqid 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;
```

## TSV Example

This example presumes source data in TSV saved in `s3://mybucket/mytsv/`.

Use a `CREATE TABLE` statement to create an Athena table from the TSV data stored in Amazon S3. Notice that this example does not reference any SerDe class in `ROW FORMAT` because it uses the `LazySimpleSerDe`, and it can be omitted. The example specifies SerDe properties for character and line separators, and an escape character:

```
CREATE EXTERNAL TABLE flight_delays_tsv (  
  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,  
crsarrrtime 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 '\t'
  ESCAPED BY '\\'
  LINES TERMINATED BY '\n'
LOCATION 's3://athena-examples-myregion/flight/tsv/';
```

Run `MSCK REPAIR TABLE` to refresh partition metadata each time a new partition is added to this table:

```
MSCK REPAIR TABLE flight_delays_tsv;
```

Query the top 10 routes delayed by more than 1 hour:

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_tsv
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.

## ORC SerDe

### SerDe Name

`OrcSerDe`

### Library Name

This is the SerDe class for data in the ORC format. It passes the object from ORC to the reader and from ORC to the writer: [OrcSerDe](#)

## Examples

#### 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,
  departedelaygroups INT,
  deptimeblk STRING,
  taxiout INT,
  wheelsoff STRING,
  wheelson STRING,
  taxiin INT,
  crsarrrtime 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

## SerDe Name

ParquetHiveSerDe is used for data stored in [Parquet Format](#).

### Note

To convert data into Parquet format, you can use [CREATE TABLE AS SELECT \(CTAS\)](#) (p. 410) queries. For more information, see [Creating a Table from Query Results \(CTAS\)](#) (p. 124), [Examples of CTAS Queries](#) (p. 130) and [Using CTAS and INSERT INTO for ETL and Data Analysis](#) (p. 133).

## Library Name

Athena uses this class when it needs to deserialize data stored in Parquet:  
[org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe](http://org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe).

## Example: Querying 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 in CSV stored in Amazon S3 in Parquet:

```
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,  
crsarrrtime 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.

## Compression Formats

The compression formats listed in this section are used for [CREATE TABLE \(p. 406\)](#) queries. For CTAS queries, Athena supports GZIP and SNAPPY (for data stored in Parquet and ORC). If you omit a format, GZIP is used by default. For more information, see [CREATE TABLE AS \(p. 410\)](#).

Athena supports the following compression formats:

- **SNAPPY** – The default compression format for files in the Parquet data storage format.
- **ZLIB** – The default compression format for files in the ORC data storage format.
- **LZO**
- **GZIP**
- **BZIP2**

## Notes and Resources

- 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 Kinesis Data Firehose logs from Athena, supported formats include GZIP compression or ORC files with SNAPPY compression.
- For more information on using compression, see section 3 ("Compress and split files") of the AWS Big Data Blog post [Top 10 Performance Tuning Tips for Amazon Athena](#).

# SQL Reference for Amazon 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 [Presto 0.172](#).

## Topics

- [Data Types in Amazon Athena \(p. 390\)](#)
- [DML Queries, Functions, and Operators \(p. 391\)](#)
- [DDL Statements \(p. 399\)](#)
- [Considerations and Limitations for SQL Queries in Amazon Athena \(p. 421\)](#)

## Data Types in Amazon Athena

When you run [CREATE TABLE \(p. 406\)](#), you specify column names and the data type that each column can contain. Athena supports the data types listed below. For information about the data type mappings that the JDBC driver supports between Athena, JDBC, and Java, see [Data Types](#) in the *JDBC Driver Installation and Configuration Guide*. For information about the data type mappings that the ODBC driver supports between Athena and SQL, see [Data Types](#) in the *ODBC Driver Installation and Configuration Guide*.

- **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** and **INTEGER** – Athena combines two different implementations of the integer data type, as follows:
  - **INT** – In Data Definition Language (DDL) queries, Athena uses the **INT** data type.
  - **INTEGER** – In DML queries, Athena uses the **INTEGER** data type. **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  $2^{31}-1$ .
    - To ensure compatibility with business analytics applications, the JDBC driver returns the **INTEGER** type.
- **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 double-precision floating point number.
- **FLOAT** – A 32-bit single-precision floating point number. 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 \(p. 454\)](#) release notes).
- **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 definitions: **DECIMAL**(11,5), **DECIMAL**(15).

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](#).

**Note**

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`, as in the following example.

```
substr(cast(col1 as varchar), 1, 4)
```

- **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. For more information, see [STRING Hive Data Type](#).

**Note**

Non-string data types cannot be cast to `STRING` in Athena; cast them to `VARCHAR` instead.

- **BINARY** – Used for data in Parquet.
- **DATE** – A date in ISO format, such as `YYYY-MM-DD`. For example, `DATE '2008-09-15'`.
- **TIMESTAMP** – Date and time instant in a `java.sql.Timestamp` compatible format, such as `yyyy-MM-dd HH:mm:ss[.f...]`. For example, `TIMESTAMP '2008-09-15 03:04:05.324'`. This format uses the session time zone.
- **ARRAY**`<data_type>`
- **MAP**`<primitive_type, data_type>`
- **STRUCT**`<col_name : data_type [COMMENT col_comment] , ...>`

## DML Queries, Functions, and Operators

Athena DML query statements are based on [Presto 0.172](#). For more information about these functions, see [Presto 0.172 Functions and Operators](#) in the open source Presto documentation. We provide links to specific subsections of that documentation in the [Presto Functions \(p. 399\)](#) topic.

Athena does not support all of Presto's features, and there are some significant differences. For more information, see the reference topics for specific statements in this section and [Considerations and Limitations \(p. 421\)](#).

**Topics**

- [SELECT \(p. 391\)](#)
- [INSERT INTO \(p. 396\)](#)
- [Presto Functions in Amazon Athena \(p. 399\)](#)

## 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 \(p. 421\)](#) and [Running SQL Queries Using Amazon Athena \(p. 110\)](#).

## Synopsis

```
[ WITH with_query [, ...] ]
```

```
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ UNION [ ALL | DISTINCT ] union_query ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

### Note

Reserved words in SQL SELECT statements must be enclosed in double quotes. For more information, see [List of Reserved Keywords in SQL SELECT Statements \(p. 85\)](#).

## 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 not supported.

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\_expr

select\_expr determines the rows to be selected.

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.

**[ 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.

The `grouping_expressions` element can be any function, such as `SUM`, `AVG`, or `COUNT`, performed on input columns, or be an ordinal number that selects an output column by position, starting at one.

`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.

These complex grouping operations don't support expressions comprising input columns. Only column names or ordinals 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.

`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.

**[ 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 [ ALL | DISTINCT ] union\_query ]**

Combines the results of more than one `SELECT` statement into a single query. `ALL` or `DISTINCT` control which rows are included in the final result set.

`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. `DISTINCT` is the default.

Multiple `UNION` clauses are processed left to right unless you use parentheses to explicitly define the order of processing.

**[ `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 null ordering is `NULLS LAST`, regardless of ascending or descending sort order.

**`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://awsexamplebucket/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 sampled.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 sampled.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://awsexamplebucket/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?](#).

## 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	<a href="#">Running SQL Queries Using Amazon Athena</a> (p. 110)
Using <code>SELECT</code> to create a table	<a href="#">Creating a Table from Query Results (CTAS)</a> (p. 124)
Inserting data from a <code>SELECT</code> query into another table	<a href="#">INSERT INTO</a> (p. 396)
Using built-in functions in <code>SELECT</code> statements	<a href="#">Presto Functions in Amazon Athena</a> (p. 399)



For Information About This	See This
Using user defined functions in <code>SELECT</code> statements	<a href="#">Querying with User Defined Functions (Preview) (p. 190)</a>
Querying Data Catalog metadata	<a href="#">Querying AWS Glue Data Catalog (p. 221)</a>

## 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.

### Important

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 [Encrypting Query Results Stored in Amazon S3 \(p. 235\)](#). 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.

## 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
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  <b>Note</b> CSV, TSV, and custom-delimited files are supported.

## Bucketed Tables Not Supported

`INSERT INTO` is not supported on bucketed tables. For more information, see [Bucketing vs Partitioning \(p. 129\)](#).

## 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 [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions \(p. 139\)](#).

## 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 [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions \(p. 139\)](#).
- For inserting unpartitioned data into a partitioned table, see [Using CTAS and INSERT INTO for ETL and Data Analysis \(p. 133\)](#).

## 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 [Identifying Query Output Files \(p. 112\)](#).

## Locating Orphaned Files

If a CTAS or `INSERT INTO` statement fails, it is possible that orphaned data are left in the data location. Because Athena does not delete any data (even 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 [Identifying Query Output Files \(p. 112\)](#) 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 \(p. 391\)](#).

## 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 [Working with Query Results, Output Files, and Query History](#) (p. 110).

## 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')
```

## Presto Functions in Amazon Athena

The Athena query engine is based on [Presto 0.172](#). For more information about these functions, see [Presto 0.172 Functions and Operators](#) and the specific sections from Presto documentation referenced below.

Athena does not support all of Presto's features. For information, see [Considerations and Limitations \(p. 421\)](#).

- [Logical Operators](#)
- [Comparison Functions and Operators](#)
- [Conditional Expressions](#)
- [Conversion Functions](#)
- [Mathematical Functions and Operators](#)
- [Bitwise Functions](#)
- [Decimal Functions and Operators](#)
- [String Functions and Operators](#)
- [Binary Functions](#)
- [Date and Time Functions and Operators](#)
- [Regular Expression Functions](#)
- [JSON Functions and Operators](#)
- [URL Functions](#)
- [Aggregate Functions](#)
- [Window Functions](#)
- [Color Functions](#)
- [Array Functions and Operators](#)
- [Map Functions and Operators](#)
- [Lambda Expressions and Functions](#)
- [Teradata Functions](#)

## DDL Statements

Use the following DDL statements directly in Athena.

The Athena query engine is based 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 \(p. 400\)](#).

### Topics

- [Unsupported DDL \(p. 400\)](#)
- [ALTER DATABASE SET DBPROPERTIES \(p. 401\)](#)
- [ALTER TABLE ADD COLUMNS \(p. 402\)](#)
- [ALTER TABLE ADD PARTITION \(p. 402\)](#)

- [ALTER TABLE DROP PARTITION \(p. 404\)](#)
- [ALTER TABLE RENAME PARTITION \(p. 404\)](#)
- [ALTER TABLE SET LOCATION \(p. 405\)](#)
- [ALTER TABLE SET TBLPROPERTIES \(p. 405\)](#)
- [CREATE DATABASE \(p. 406\)](#)
- [CREATE TABLE \(p. 406\)](#)
- [CREATE TABLE AS \(p. 410\)](#)
- [CREATE VIEW \(p. 412\)](#)
- [DESCRIBE TABLE \(p. 413\)](#)
- [DESCRIBE VIEW \(p. 414\)](#)
- [DROP DATABASE \(p. 414\)](#)
- [DROP TABLE \(p. 414\)](#)
- [DROP VIEW \(p. 415\)](#)
- [MSCK REPAIR TABLE \(p. 415\)](#)
- [SHOW COLUMNS \(p. 417\)](#)
- [SHOW CREATE TABLE \(p. 417\)](#)
- [SHOW CREATE VIEW \(p. 418\)](#)
- [SHOW DATABASES \(p. 418\)](#)
- [SHOW PARTITIONS \(p. 418\)](#)
- [SHOW TABLES \(p. 419\)](#)
- [SHOW TBLPROPERTIES \(p. 420\)](#)
- [SHOW VIEWS \(p. 420\)](#)

## Unsupported DDL

The following native Hive DDLs are not supported by Athena:

- ALTER INDEX
- ALTER TABLE `table_name` ARCHIVE PARTITION
- ALTER TABLE `table_name` CLUSTERED BY
- 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` REPLACE COLUMNS
- ALTER TABLE `table_name` `partitionSpec` SET FILEFORMAT
- ALTER TABLE `table_name` RENAME TO
- 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 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)
```

### 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.

## 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 [Partitioning Data \(p. 92\)](#).

In Athena, a table and its partitions must use the same data formats but their schemas may differ. For more information, see [Updates in Tables with Partitions \(p. 149\)](#).

For information about the resource-level permissions required in IAM policies (including `glue:CreatePartition`), see [AWS Glue API Permissions: Actions and Resources Reference](#) and [Fine-Grained Access to Databases and Tables in the AWS Glue Data Catalog \(p. 244\)](#).

## 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 partitions defined by the preceding statement.

## Examples

```
ALTER TABLE orders ADD
PARTITION (dt = '2016-05-14', country = 'IN');
```

```
ALTER TABLE orders ADD
PARTITION (dt = '2016-05-14', country = 'IN')
PARTITION (dt = '2016-05-15', country = 'IN');
```

```
ALTER TABLE orders ADD
PARTITION (dt = '2016-05-14', country = 'IN') LOCATION 's3://mystorage/path/to/
INDIA_14_May_2016/'
PARTITION (dt = '2016-05-15', country = 'IN') LOCATION 's3://mystorage/path/to/
INDIA_15_May_2016/';
```



## 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');
```

## ALTER TABLE RENAME PARTITION

Renames a partition column, `partition_spec`, for the table named `table_name`, to `new_partition_spec`.

For information about partitioning, see [Partitioning Data \(p. 92\)](#).

### Synopsis

```
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');
```

## 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 \(p. 86\)](#).

### Examples

```
ALTER TABLE customers PARTITION (zip='98040', state='WA') SET LOCATION 's3://mystorage/  
custdata/';
```

## ALTER TABLE SET TBLPROPERTIES

Adds custom metadata properties to a table and sets their assigned values.

[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 reset to `property_value`.

### Examples

```
ALTER TABLE orders
```

```
SET TBLPROPERTIES ('notes'="Please don't drop this table.");
```

## CREATE DATABASE

Creates a database. The use of DATABASE and SCHEMA is interchangeable. They mean the same thing.

### Synopsis

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name  
  [COMMENT 'database_comment']  
  [LOCATION 'S3_loc']  
  [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

### 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.

#### [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://myS3location/clickstreams/'  
  WITH DBPROPERTIES ('creator'='Jane D.', 'Dept.'='Marketing analytics');
```

## CREATE TABLE

Creates a table with the name and the parameters that you specify.

### 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], ...)]  
  [ROW FORMAT row_format]  
  [STORED AS file_format]  
  [WITH SERDEPROPERTIES (...)] ]
```

```
[LOCATION 's3://bucket_name/[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. 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]

Causes the error message to be suppressed if a table named `table_name` already exists.

### [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.

### [ ( 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``.

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**. Athena combines two different implementations of the **INTEGER** data type. In Data Definition Language (DDL) queries, Athena uses the **INT** data type. In all other queries, Athena uses the **INTEGER** data type, 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**
- **FLOAT**
- **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)`.

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 and time types
- DATE A date in ISO format, such as `YYYY-MM-DD`. For example, `DATE '2008-09-15'`.
- TIMESTAMP Date and time instant in a `java.sql.Timestamp` compatible format, such as `yyyy-MM-dd HH:mm:ss[.f...]`. For example, `TIMESTAMP '2008-09-15 03:04:05.324'`. This format uses the session time zone.
- 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 \(p. 92\)](#).

**Note**

After you create a table with partitions, run a subsequent query that consists of the [MSCK REPAIR TABLE \(p. 415\)](#) clause to refresh partition metadata, for example, `MSCK REPAIR TABLE cloudfront_logs;`. For partitions that are not Hive compatible, use [ALTER TABLE ADD PARTITION \(p. 402\)](#) to load the partitions so that you can query the data.

**[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`
- `INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname`

**[LOCATION 's3://bucket\_name/[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 [Table Location in Amazon S3 \(p. 86\)](#). For information about data format and permissions, see [Requirements for Tables in Athena and Data in Amazon S3 \(p. 79\)](#).

Use a trailing slash for your folder or bucket. Do not use file names or glob characters.

**Use:**

```
s3://mybucket/
```

```
s3://mybucket/folder/
```

```
s3://mybucket/folder/anotherfolder/
```

**Don't use:**

```
s3://path_to_bucket
```

```
s3://path_to_bucket/*
```

```
s3://path_to-bucket/mydatafile.dat
```

**[TBLPROPERTIES ( ['has\_encrypted\_data'='true | false',] ['classification'='aws\_glue\_classification',] property\_name=property\_value [, ...] ) ]**

Specifies custom metadata key-value pairs for the table definition in addition to predefined table properties, such as `"comment"`.

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 [Configuring Encryption Options \(p. 233\)](#).

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 [Using](#)



### 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.

## 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 \(p. 130\)](#).

**WITH (property\_name = expression [, ...], )**  
**external\_location = [location]**

Optional. The location where Athena saves your CTAS query in Amazon S3, as in the following example:

```
WITH (external_location = 's3://my-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 \(p. 330\)](#), the query fails with an error message. To see the query results location specified for the workgroup, [see the workgroup's details \(p. 334\)](#).

If your workgroup overrides the client-side setting for query results location, Athena creates your table in the following location:

```
s3://<workgroup-query-results-location>/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 \(p. 115\)](#) for the query results location to create your table in the following location:

```
s3://<query-results-location-setting>/<Unsaved-or-query-name>/<year>/<month>/<date>/
tables/<query-id>/

```

**format = [format]**

The data format for the CTAS query results, such as `ORC`, `PARQUET`, `AVRO`, `JSON`, or `TEXTFILE`. For example, `WITH (format = 'PARQUET')`. If omitted, `PARQUET` is used by default. The name of this parameter, `format`, must be listed in lowercase, or your CTAS query will fail.

**partitioned\_by = ARRAY( [col\_name,...])**

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.

**bucketed\_by( [bucket\_name,...])**

An array list of buckets to bucket data. If omitted, Athena does not bucket your data in this query.



**bucket\_count** = [int]

The number of buckets for bucketing your data. If omitted, Athena does not bucket your data.

**orc\_compression** = [format]

The compression type to use for ORC data. For example, `WITH (orc_compression = 'ZLIB')`. If omitted, GZIP compression is used by default for ORC and other data storage formats supported by CTAS.

**parquet\_compression** = [format]

The compression type to use for Parquet data. For example, `WITH (parquet_compression = 'SNAPPY')`. If omitted, GZIP compression is used by default for Parquet and other data storage formats supported by CTAS.

**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.

## Examples

For examples of CTAS queries, consult the following resources.

- [Examples of CTAS Queries \(p. 130\)](#)
- [Using CTAS and INSERT INTO for ETL and Data Analysis \(p. 133\)](#)
- [Use CTAS statements with Amazon Athena to reduce cost and improve performance](#)
- [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions \(p. 139\)](#)

## CREATE VIEW

Creates a new view from a specified `SELECT` query. The 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.

### Note

This topic provides summary information for reference. For more detailed information about using views in Athena, see [Working with Views \(p. 119\)](#).

## 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 [Creating Views \(p. 122\)](#).

## 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;
```

See also [SHOW COLUMNS \(p. 417\)](#), [SHOW CREATE VIEW \(p. 418\)](#), [DESCRIBE VIEW \(p. 414\)](#), and [DROP VIEW \(p. 415\)](#).

## DESCRIBE TABLE

Shows the list of columns, including partition columns, for the named column. This allows you to examine the attributes of a complex column.

### Synopsis

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_spec] [col_name
 ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

### Parameters

#### [EXTENDED | FORMATTED]

Determines the format of the output. If you specify `EXTENDED`, all metadata for the table is output in Thrift serialized form. This is useful primarily for debugging and not for general use. Use `FORMATTED` or omit the clause to show the metadata in tabular format.

#### [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;
```

## DESCRIBE VIEW

Shows the list of columns for the named view. This allows you to examine the attributes of a complex view.

### Synopsis

```
DESCRIBE [view_name]
```

### Example

```
DESCRIBE orders;
```

See also [SHOW COLUMNS](#) (p. 417), [SHOW CREATE VIEW](#) (p. 418), [SHOW VIEWS](#) (p. 420), and [DROP VIEW](#) (p. 415).

## 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;
```

## DROP TABLE

Removes the metadata table definition for the table named `table_name`. When you drop an external table, the underlying data remains intact because all tables in Athena are `EXTERNAL`.

## 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 view. The optional `IF EXISTS` clause causes the error to be suppressed if the view does not exist.

For more information, see [Deleting Views \(p. 124\)](#).

## Synopsis

```
DROP VIEW [ IF EXISTS ] view_name
```

## Examples

```
DROP VIEW orders_by_date
```

```
DROP VIEW IF EXISTS orders_by_date
```

See also [CREATE VIEW \(p. 412\)](#), [SHOW COLUMNS \(p. 417\)](#), [SHOW CREATE VIEW \(p. 418\)](#), [SHOW VIEWS \(p. 420\)](#), and [DESCRIBE VIEW \(p. 414\)](#).

## MSCK REPAIR TABLE

Use the `MSCK REPAIR TABLE` command to update the metadata in the catalog after you add or remove Hive compatible partitions.

The `MSCK REPAIR TABLE` command scans a file system such as Amazon S3 for Hive compatible partitions that were added to or removed from the file system after the table was created. The command updates the metadata in the catalog regarding the partitions and the data associated with them.

When you add or remove partitions, the metadata in the catalog becomes inconsistent with the layout of the data in the file system. For example, after you create a table with partitions, information about the new partitions needs to be added to the catalog. To update the metadata, you run `MSCK REPAIR TABLE` on the table. This enables you to query the data in the new partitions from Athena.

## 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 [Partitioning Data](#) (p. 92).
- For partitions that are not compatible with Hive, use [ALTER TABLE ADD PARTITION](#) (p. 402) 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://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.

## 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:

- Make sure that the AWS Identity and Access Management (IAM) user or role has a policy that allows the `glue:BatchCreatePartition` action.
- Make sure that the IAM user or 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 [Cross-account Access in Athena to Amazon S3 Buckets](#) (p. 251).
- Make sure that the Amazon S3 path is in lower case instead of camel case (for example, `userid` instead of `userId`).

The following sections provide some additional detail.

### Allow `glue:BatchCreatePartition` in the IAM policy

Review the IAM policies attached to the user or role that you're using to run `MSCK REPAIR TABLE`. When you [use the AWS Glue Data Catalog with Athena](#) (p. 16), the IAM policy must allow

the `glue:BatchCreatePartition` action. For an example of an IAM policy that allows the `glue:BatchCreatePartition` action, see [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#).

## Change the Amazon S3 path to lower case

The Amazon S3 path must be in lower case. If the S3 path is in camel case, `MSCK REPAIR TABLE` doesn't add the partitions to the AWS Glue Data Catalog. For example, if your S3 path is `userId`, the following partitions aren't added to the AWS Glue Data Catalog:

```
s3://bucket/path/userId=1/
s3://bucket/path/userId=2/
s3://bucket/path/userId=3/
```

To resolve this issue, use flat case instead of camel case:

```
s3://bucket/path/userid=1/
s3://bucket/path/userid=2/
s3://bucket/path/userid=3/
```

## SHOW COLUMNS

Lists the columns in the schema for a base table or a view.

### Synopsis

```
SHOW COLUMNS IN table_name|view_name
```

### Examples

```
SHOW COLUMNS IN clicks;
```

## 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`;
```

## SHOW CREATE VIEW

Shows the SQL statement that creates the specified view.

## Synopsis

```
SHOW CREATE VIEW view_name
```

## Examples

```
SHOW CREATE VIEW orders_by_date
```

See also [CREATE VIEW \(p. 412\)](#) and [DROP VIEW \(p. 415\)](#).

## SHOW DATABASES

Lists all databases defined in the metastore. You can use `DATABASES` or `SCHEMAS`. They mean the same thing.

## 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 a table.

## Synopsis

```
SHOW PARTITIONS table_name
```

- To show the partitions in a table and list them in a specific order, see the [Listing Partitions for a Specific Table \(p. 222\)](#) section on the [Querying AWS Glue Data Catalog \(p. 221\)](#) page.
- To view the contents of a partition, see the [Query the Data \(p. 94\)](#) section on the [Partitioning Data \(p. 92\)](#) 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 [Partition Projection with Amazon Athena \(p. 96\)](#).

## Examples

```
SHOW PARTITIONS clicks;
```

## SHOW TABLES

Lists all the base tables and views in a database.

## 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.

### ['regular\_expression']

Filters the list of tables to those that match the `regular_expression` you specify. Only the wildcard `*`, which indicates any character, or `|`, which indicates a choice between characters, can be used.

## Examples

### Example – show all of the tables in the database `sampledb`

```
SHOW TABLES IN sampled
```

#### Results

```
alb_logs
cloudfront_logs
elb_logs
flights_2016
flights_parquet
view_2016_flights_dfw
```



**Example – show the names of all tables in `samp1edb` that include the word "flights"**

```
SHOW TABLES IN samp1edb '*flights*'
```

**Results**

```
flights_2016  
flights_parquet  
view_2016_flights_dfw
```

**Example – show the names of all tables in `samp1edb` that end in the word "logs"**

```
SHOW TABLES IN samp1edb '*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 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.

Athena returns a list of `STRING` type values where each value is a view name.

### 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 wildcard `*`, 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](#) (p. 417), [SHOW CREATE VIEW](#) (p. 418), [DESCRIBE VIEW](#) (p. 414), and [DROP VIEW](#) (p. 415).

# 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.
- **Parameterized queries** – Parameterized queries are not supported. However, you can create user-defined functions that you can call in the body of a query. For more information, see [Querying with User Defined Functions \(Preview\)](#) (p. 190).
- **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](#) (p. 410). For a workaround, see [Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions](#) (p. 139).
- **Unsupported statements** – The following statements are not supported:
  - `PREPARED` statements are not supported. You cannot run `EXECUTE` with `USING`.
  - `CREATE TABLE LIKE` is not supported.
  - `DESCRIBE INPUT` and `DESCRIBE OUTPUT` is not supported.
  - `EXPLAIN` statements are not supported.
- **Presto federated connectors** – [Presto federated connectors](#) are not supported. Use Amazon Athena Federated Query (Preview) to connect data sources. For more information, see [Using Amazon Athena Federated Query \(Preview\)](#) (p. 56).
- **Querying Parquet columns with complex data types** – When you query columns with complex data types (`array`, `map`, `struct`), and are using Parquet for storing data, Athena currently reads an entire row of data instead of selectively reading only the specified columns. This is a known issue.
- **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.

- **Cross-region queries** – Cross-region queries are not supported. If you create an Athena table in one AWS Region and attempt to query data in an Amazon S3 bucket in another AWS Region, you may receive the error message `InvalidToken: The provided token is malformed or otherwise invalid`.
- **Amazon S3 GLACIER storage** – When data is moved or transitioned to the [Amazon S3 GLACIER storage class](#), it is no longer readable or queryable by Athena. This is true even after storage class objects are restored. To make the restored objects that you want to query readable by Athena, copy the restored objects back into Amazon S3 to change their storage class.
- **Amazon S3 access points** – You cannot use an Amazon S3 access point in a `LOCATION` clause. However, as long as the Amazon S3 bucket policy does not explicitly deny requests to objects not made through Amazon S3 access points, the objects should be accessible from Athena for requestors that have the right object access permissions.
- **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.

# Code Samples, Service Quotas, and Previous JDBC Driver

Use code samples to create Athena applications based on AWS SDK for Java.

Use the links in this section to use earlier versions of the JDBC driver.

Learn about service quotas.

## Topics

- [Code Samples \(p. 423\)](#)
- [Using Earlier Version JDBC Drivers \(p. 433\)](#)
- [Service Quotas \(p. 438\)](#)

## Code Samples

Use the examples in this topic as a starting point for writing Athena applications using the SDK for Java 2.x. For more information about running the Java code examples, see the [Amazon Athena Java Readme](#) on the [AWS Code Examples Repository](#) on GitHub.

- **Java Code Examples**
  - [Constants \(p. 423\)](#)
  - [Create a Client to Access Athena \(p. 424\)](#)
  - **Working with Query Executions**
    - [Start Query Execution \(p. 424\)](#)
    - [Stop Query Execution \(p. 427\)](#)
    - [List Query Executions \(p. 428\)](#)
  - **Working with Named Queries**
    - [Create a Named Query \(p. 429\)](#)
    - [Delete a Named Query \(p. 430\)](#)
    - [List Query Executions \(p. 428\)](#)

### 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.

## Constants

The `ExampleConstants.java` class demonstrates how to query a table created by the [Getting Started \(p. 8\)](#) 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://mybucket"; //change the bucket
    name to match your environment
    // This example demonstrates how to query a table with a CSV For information, see
    //https://docs.aws.amazon.com/athena/latest/ug/work-with-data.html
    public static final String ATHENA_SAMPLE_QUERY = "SELECT * FROM mydb;"; //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.InstanceProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.AthenaClientBuilder;

/**
 * AthenaClientFactory
 * -----
 * This code shows how to create and configure an Amazon Athena client.
 */
public class AthenaClientFactory {
    /**
     * AthenaClientBuilder to build Athena with the following properties:
     * - Set the region of the client
     * - Use the instance profile from the EC2 instance as the credentials provider
     * - Configure the client to increase the execution timeout.
     */
    private final AthenaClientBuilder builder = AthenaClient.builder()
        .region(Region.US_WEST_2)
        .credentialsProvider(InstanceProfileCredentialsProvider.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;

/**
 * StartQueryExample
 * -----
 * This code shows how to submit a query to Athena for execution, wait till results
 * are available, and then process the results.
 */
public class StartQueryExample {
    public static void main(String[] args) throws InterruptedException {

        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();

        String queryExecutionId = submitAthenaQuery(athenaClient);

        waitForQueryToComplete(athenaClient, queryExecutionId);

        processResultRows(athenaClient, queryExecutionId);
    }

    /**
     * Submits a sample query to 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
            // in S3 and encryption options
            ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                // You can provide encryption options for the output that is written.
                // .withEncryptionConfiguration(encryptionConfiguration)
                .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET).build();

            // Create the StartQueryExecutionRequest to send to Athena which will start the
            // query.
            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 Athena query to complete, fail or to be cancelled. This is done by
 * polling Athena over an
 * interval of time. If a query fails or is cancelled, then it will throw an exception.
 */

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("Query Failed to run with Error Message: " +
            getQueryExecutionResponse
                .queryExecution().status().stateChangeReason());
        } else if (queryState.equals(QueryExecutionState.CANCELLED.toString())) {
            throw new RuntimeException("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("Current Status is: " + queryState);
    }
}

/**
 * This code calls Athena and retrieves the results of a query.
 * The query must be in a completed state before the results can be retrieved and
 * paginated. The first row of results are the column headers.
 */
public static void processResultRows(AthenaClient athenaClient, String
queryExecutionId) {

    try {

        GetQueryResultsRequest getQueryResultsRequest =
        GetQueryResultsRequest.builder()
            // Max Results can be set but if its not set,
            // it will choose the maximum page size
            // As of the writing of this code, the maximum value is 1000
            // .withMaxResults(1000)
            .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) {

    //Write out the data
    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.StopQueryExecutionResponse;
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;

/**
 * StopQueryExecutionExample
 * -----
 * This code runs an example query, immediately stops the query, and checks the status of
 * the query to
 * ensure that it was cancelled.
 */
public class StopQueryExecutionExample {
    public static void main(String[] args) throws Exception {

        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();

        String sampleQueryExecutionId = submitAthenaQuery(athenaClient);
        stopAthenaQuery(athenaClient, sampleQueryExecutionId);
    }

    public static void stopAthenaQuery(AthenaClient athenaClient, String
sampleQueryExecutionId){

        try {
            // Submit the stop query Request
            StopQueryExecutionRequest stopQueryExecutionRequest =
StopQueryExecutionRequest.builder()

```



```

        .queryExecutionId(sampleQueryExecutionId).build();

        StopQueryExecutionResponse stopQueryExecutionResponse =
athenaClient.stopQueryExecution(stopQueryExecutionRequest);

        // Ensure that the query was stopped
        GetQueryExecutionRequest getQueryExecutionRequest =
GetQueryExecutionRequest.builder()
        .queryExecutionId(sampleQueryExecutionId).build();

        GetQueryExecutionResponse getQueryExecutionResponse =
athenaClient.getQueryExecution(getQueryExecutionRequest);
        if (getQueryExecutionResponse.queryExecution()
            .status()
            .state()
            .equals(QueryExecutionState.CANCELLED)) {

            // Query was cancelled.
            System.out.println("Query has been cancelled");
        }

    } catch (AthenaException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

/**
 * Submits an example query and returns a query execution ID of a running query to
stop.
 */
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;

/**
 * ListQueryExecutionsExample
 * -----
 * This code shows how to obtain a list of query execution IDs.
 */
public class ListQueryExecutionsExample {

    public static void main(String[] args) throws Exception {

        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();

        listQueryIds(athenaClient);
    }

    public static void listQueryIds(AthenaClient athenaClient) {

        try {
            // Build the request
            ListQueryExecutionsRequest listQueryExecutionsRequest =
                ListQueryExecutionsRequest.builder().build();

            // Get the list results.
            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;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryResponse;
```

```
/**
 * CreateNamedQueryExample
 * -----
 * This code shows how to create a named query.
 */
public class CreateNamedQueryExample {
    public static void main(String[] args) throws Exception {

        final String USAGE = "\n" +
            "Usage:\n" +
            "    CreateNamedQueryExample <name>\n\n" +
            "Where:\n" +
            "    name - the name of the query \n\n" +
            "Example:\n" +
            "    DescribeTable SampleQuery\n";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        /* Read the name from command args */
        String name = args[0];

        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();

        createNamedQuery(athenaClient, name);
    }

    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();

            // Call Athena to create the named query. If it fails, an exception is thrown.
            CreateNamedQueryResponse createNamedQueryResult =
            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.DeleteNamedQueryResponse;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryResponse;

/**
 * DeleteNamedQueryExample
 * -----
 * This code shows how to delete a named query by using the named query ID.
 */
public class DeleteNamedQueryExample {

    public static void main(String[] args) {

        final String USAGE = "\n" +
            "Usage:\n" +
            "    DeleteNamedQueryExample <name>\n\n" +
            "Where:\n" +
            "    name - the name of the query\n\n" +
            "Example:\n" +
            "    DeleteNamedQueryExample SampleQuery\n";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        /* Read the name from command args */
        String name = args[0];

        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();
        String sampleNamedQueryId = getNamedQueryId(athenaClient, name);
        deleteQueryName(athenaClient, sampleNamedQueryId);
    }

    public static void deleteQueryName(AthenaClient athenaClient, String sampleNamedQueryId)
    {
        try {
            // Create the delete named query request
            DeleteNamedQueryRequest deleteNamedQueryRequest =
                DeleteNamedQueryRequest.builder()
                    .namedQueryId(sampleNamedQueryId).build();

            // Delete the named query
            DeleteNamedQueryResponse deleteNamedQueryResponse =
                athenaClient.deleteNamedQuery(deleteNamedQueryRequest);
        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static String getNamedQueryId(AthenaClient athenaClient, String name) {
        try {
            // Create the NameQuery Request.
            CreateNamedQueryRequest createNamedQueryRequest =
                CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .name(name)
                    .description("Sample Description").build();
        }
    }
}
```

```

        // Create the named query. If it fails, an exception is thrown.
        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;

/**
 * ListNamedQueryExample
 * -----
 * This code shows how to obtain a list of named query IDs.
 */
public class ListNamedQueryExample {

    public static void main(String[] args) throws Exception {
        // Build an Athena client
        AthenaClient athenaClient = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .build();

        listNamedQueries(athenaClient);
    }

    public static void listNamedQueries(AthenaClient athenaClient) {

        try{

            // Build the request
            ListNamedQueriesRequest listNamedQueriesRequest =
ListNamedQueriesRequest.builder().build();

            // Get the list results.
            ListNamedQueriesIterable listNamedQueriesResponses =
athenaClient.listNamedQueriesPaginator(listNamedQueriesRequest);

            // Process the results.
            for (ListNamedQueriesResponse listNamedQueriesResponse :
listNamedQueriesResponses) {
                List<String> namedQueryIds = listNamedQueriesResponse.namedQueryIds();
                // process named query IDs
                System.out.println(namedQueryIds);
            }
        }
    }
}

```

```

    } catch (AthenaException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

```

## Using Earlier Version JDBC Drivers

We recommend that you use the latest version of the JDBC driver. For information, see [Using Athena with the JDBC Driver \(p. 72\)](#). Links to earlier version 2.x drivers and support materials are below if required for your application.

### Earlier Version JDBC Drivers

JDBC Driver Version	Downloads					
<b>2.0.8</b>	JDBC 4.2 and JDK 8.0 Compatible – <a href="#">AthenaJDBC42-2.0.8.jar</a>	<a href="#">Release Notes</a>	<a href="#">License Agreement</a>	<a href="#">Notices</a>	<a href="#">Installation and Configuration Guide (PDF)</a>	<a href="#">Migration Guide(PDF)</a>
	JDBC 4.1 and JDK 7.0 Compatible – <a href="#">AthenaJDBC41-2.0.8.jar</a>					
<b>2.0.7</b>	JDBC 4.2 and JDK 8.0 Compatible – <a href="#">AthenaJDBC42-2.0.7.jar</a>	<a href="#">Release Notes</a>	<a href="#">License Agreement</a>	<a href="#">Notices</a>	<a href="#">Installation and Configuration Guide (PDF)</a>	<a href="#">Migration Guide(PDF)</a>
	JDBC 4.1 and JDK 7.0 Compatible – <a href="#">AthenaJDBC41-2.0.7.jar</a>					
<b>2.0.6</b>	JDBC 4.2 and JDK 8.0 Compatible – <a href="#">AthenaJDBC42-2.0.6.jar</a>	<a href="#">Release Notes</a>	<a href="#">License Agreement</a>	<a href="#">Notices</a>	<a href="#">Installation and Configuration Guide (PDF)</a>	<a href="#">Migration Guide(PDF)</a>
	JDBC 4.1 and JDK 7.0 Compatible – <a href="#">AthenaJDBC41-2.0.6.jar</a>					
<b>2.0.5</b>	JDBC 4.2 and JDK 8.0	<a href="#">Release Notes</a>	<a href="#">License Agreement</a>	<a href="#">Notices</a>	<a href="#">Installation and</a>	<a href="#">Migration Guide(PDF)</a>

JDBC Driver Version	Downloads					
	Compatible – <a href="#">AthenaJDBC42-2.0.5.jar</a>				<a href="#">Configuration Guide (PDF)</a>	
	JDBC 4.1 and JDK 7.0 Compatible – <a href="#">AthenaJDBC41-2.0.5.jar</a>					
<b>2.0.2</b>	JDBC 4.2 and JDK 8.0 Compatible – <a href="#">AthenaJDBC42-2.0.2.jar</a>	<a href="#">Release Notes</a>	<a href="#">License Agreement</a>	<a href="#">Notices</a>	<a href="#">Installation and Configuration Guide (PDF)</a>	<a href="#">Migration Guide(PDF)</a>
	JDBC 4.1 and JDK 7.0 Compatible – <a href="#">AthenaJDBC41-2.0.2.jar</a>					

## Instructions for JDBC Driver version 1.1.0

This section includes a link to download version 1.1.0 of the JDBC driver. We highly recommend that you migrate to the current version of the driver. For information, see the [JDBC Driver Migration Guide](#).

The JDBC driver version 1.0.1 and earlier versions are deprecated.

JDBC driver version 1.1.0 is compatible with JDBC 4.1 and JDK 7.0. Use the following link to download the driver: [AthenaJDBC41-1.1.0.jar](#). Also, download the [driver license](#), and the [third-party licenses](#) for the driver. Use the AWS CLI with the following command: `aws s3 cp s3://path_to_the_driver [local_directory]`, and then use the remaining instructions in this section.

### Note

The following instructions are specific to JDBC version 1.1.0 and earlier.

## JDBC Driver Version 1.1.0: Specify the Connection String

To specify the JDBC driver connection URL in your custom application, use the string in this format:

```
jdbc:awsathena://athena.{REGION}.amazonaws.com:443
```

where {REGION} is a region identifier, such as `us-west-2`. For information on Athena regions see [Regions](#).

## JDBC Driver Version 1.1.0: Specify the JDBC Driver Class Name

To use the driver in custom applications, set up your Java class path to the location of the JAR file that you downloaded from Amazon S3 [https://s3.amazonaws.com/athena-downloads/drivers/JDBC/AthenaJDBC\\_1.1.0/AthenaJDBC41-1.1.0.jar](https://s3.amazonaws.com/athena-downloads/drivers/JDBC/AthenaJDBC_1.1.0/AthenaJDBC41-1.1.0.jar). This makes the classes within the JAR available for use. The main JDBC driver class is `com.amazonaws.athena.jdbc.AthenaDriver`.

## JDBC Driver Version 1.1.0: Provide the JDBC Driver Credentials

To gain access to AWS services and resources, such as Athena and the Amazon S3 buckets, provide JDBC driver credentials to your application.

To provide credentials in the Java code for your application:

1. Use a class which implements the [AWSCredentialsProvider](#).
2. Set the JDBC property, `aws_credentials_provider_class`, equal to the class name, and include it in your classpath.
3. To include constructor parameters, set the JDBC property `aws_credentials_provider_arguments` as specified in the following section about configuration options.

Another method to supply credentials to BI tools, such as SQL Workbench, is to supply the credentials used for the JDBC as AWS access key and AWS secret key for the JDBC properties for user and password, respectively.

Users who connect through the JDBC driver and have custom access policies attached to their profiles need permissions for policy actions in addition to those in the [Amazon Athena API Reference](#).

## Policies for the JDBC Driver Version 1.1.0

You must allow JDBC users to perform a set of policy-specific actions. If the following actions are not allowed, users will be unable to see databases and tables:

- `athena:GetCatalogs`
- `athena:GetExecutionEngine`
- `athena:GetExecutionEngines`
- `athena:GetNamespace`
- `athena:GetNamespaces`
- `athena:GetTable`
- `athena:GetTables`

## JDBC Driver Version 1.1.0: Configure the JDBC Driver Options

You can configure the following options for the version of the JDBC driver version 1.1.0. With this version of the driver, you can also pass parameters using the standard JDBC URL syntax, for example: `jdbc:awsathena://athena.us-west-1.amazonaws.com:443?max_error_retries=20&connection_timeout=20000`.

### Options for the JDBC Driver Version 1.0.1

Property Name	Description	Default Value	Is Required
<code>s3_staging_dir</code>	The S3 location to which your query output is written, for example <code>s3://query-results-bucket/folder/</code> , which is established under <b>Settings</b> in the Athena Console, <a href="https://console.aws.amazon.com/athena/">https://console.aws.amazon.com/athena/</a> . The JDBC driver then asks Athena to read the results and provide rows of data back to the user.	N/A	Yes



Property Name	Description	Default Value	Is Required
<code>query_results_encryption_option</code>	The encryption method to use for the directory specified by <code>s3_staging_dir</code> . If not specified, the location is not encrypted. Valid values are <code>SSE_S3</code> , <code>SSE_KMS</code> , and <code>CSE_KMS</code> .	N/A	No
<code>query_results_aws_kms_key_id</code>	The key ID of the AWS customer master key (CMK) to use if <code>query_results_encryption_option</code> specifies <code>SSE-KMS</code> or <code>CSE-KMS</code> . For example, <code>123abcde-4e56-56f7-g890-1234h5678i9j</code> .	N/A	No
<code>aws_credentials_provider</code>	The credentials provider class name, which implements the <code>AWSCredentialsProvider</code> interface.	N/A	No
<code>aws_credentials_provider_args</code>	Arguments for the credentials provider constructor as comma-separated values.	N/A	No
<code>max_error_retries</code>	The maximum number of retries that the JDBC client attempts to make a request to Athena.	10	No
<code>connection_timeout</code>	The maximum amount of time, in milliseconds, to make a successful connection to Athena before an attempt is terminated.	10,000	No
<code>socket_timeout</code>	The maximum amount of time, in milliseconds, to wait for a socket in order to send data to Athena.	10,000	No
<code>retry_base_delay</code>	Minimum delay amount, in milliseconds, between retrying attempts to connect Athena.	100	No
<code>retry_max_backoff_time</code>	Maximum delay amount, in milliseconds, between retrying attempts to connect to Athena.	1000	No
<code>log_path</code>	Local path of the Athena JDBC driver logs. If no log path is provided, then no log files are created.	N/A	No
<code>log_level</code>	Log level of the Athena JDBC driver logs. Valid values: <code>INFO</code> , <code>DEBUG</code> , <code>WARN</code> , <code>ERROR</code> , <code>ALL</code> , <code>OFF</code> , <code>FATAL</code> , <code>TRACE</code> .	N/A	No

## Examples: Using the 1.1.0 Version of the JDBC Driver with the JDK

The following code examples demonstrate how to use the JDBC driver version 1.1.0 in a Java application. These examples assume that the AWS JAVA SDK is included in your classpath, specifically the `aws-java-sdk-core` module, which includes the authorization packages (`com.amazonaws.auth.*`) referenced in the examples.

### Example Example: Creating a Driver Version 1.0.1

```
Properties info = new Properties();
info.put("user", "AWSAccessKey");
info.put("password", "AWSSecretAccessKey");
info.put("s3_staging_dir", "s3://S3 Bucket Location/");
```

```
info.put("aws_credentials_provider_class", "com.amazonaws.auth.DefaultAWSCredentialsProviderChain");

Class.forName("com.amazonaws.athena.jdbc.AthenaDriver");

Connection connection = DriverManager.getConnection("jdbc:awsathena://athena.us-east-1.amazonaws.com:443/", info);
```

The following examples demonstrate different ways to use a credentials provider that implements the `AWSCredentialsProvider` interface with the previous version of the JDBC driver.

### Example Example: Using a Credentials Provider for JDBC Driver 1.0.1

```
Properties myProps = new Properties();

myProps.put("aws_credentials_provider_class", "com.amazonaws.auth.PropertiesFileCredentialsProvider");
myProps.put("aws_credentials_provider_arguments", "/Users/myUser/.athenaCredentials");
```

In this case, the file `/Users/myUser/.athenaCredentials` should contain the following:

```
accessKey = ACCESSKEY
secretKey = SECRETKEY
```

Replace the right part of the assignments with your account's AWS access and secret keys.

### Example Example: Using a Credentials Provider with Multiple Arguments

This example shows an example credentials provider, `CustomSessionsCredentialsProvider`, that uses an access and secret key in addition to a session token. `CustomSessionsCredentialsProvider` is shown for example only and is not included in the driver. The signature of the class looks like the following:

```
public CustomSessionsCredentialsProvider(String accessId, String secretKey, String token)
{
    //...
}
```

You would then set the properties as follows:

```
Properties myProps = new Properties();

myProps.put("aws_credentials_provider_class", "com.amazonaws.athena.jdbc.CustomSessionsCredentialsProvider");
String providerArgs = "My_Access_Key," + "My_Secret_Key," + "My_Token";
myProps.put("aws_credentials_provider_arguments", providerArgs);
```

#### Note

If you use the [InstanceProfileCredentialsProvider](#), you don't need to supply any credential provider arguments because they are provided using the Amazon EC2 instance profile for the instance on which you are running your application. You would still set the `aws_credentials_provider_class` property to this class name, however.

## Policies for the JDBC Driver Earlier than Version 1.1.0

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
<code>athena:RunQuery</code>	<code>athena:StartQueryExecution</code>
<code>athena:CancelQueryExecution</code>	<code>athena:StopQueryExecution</code>
<code>athena:GetQueryExecutions</code>	<code>athena:ListQueryExecutions</code>

## Service Quotas

### Note

The Service Quotas console provides information about Amazon Athena quotas. Along with viewing the default quotas, you can use the Service Quotas console to [request quota increases](#) for the quotas that are adjustable.

## Queries

Your account has the following default query-related quotas per AWS Region for Amazon Athena:

- **DDL query quota** – 20 DDL active queries. DDL queries include `CREATE TABLE` and `CREATE TABLE ADD PARTITION` queries.
- **DDL query timeout** – The DDL query timeout is 600 minutes.
- **DML query quota** – 20 DML active queries. DML queries include `SELECT` and `CREATE TABLE AS (CTAS)` queries.
- **DML query timeout** – The DML query timeout is 30 minutes.

These are soft quotas; you can use the [Athena Service Quotas](#) console to request a quota increase.

### Note

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.

## 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. Use these [tips \(p. 84\)](#) for naming columns, tables, and databases in Athena.

### Note

If you require a greater query string length, provide feedback at [athena-feedback@amazon.com](mailto:athena-feedback@amazon.com) with the details of your use case, or contact [AWS Support](#).

## 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 tags per workgroup is 50. For more information, see [Tag Restrictions \(p. 348\)](#).

## AWS Glue

- If you are using the AWS Glue Data Catalog with Athena, see [AWS Glue Endpoints and Quotas](#) for service quotas on tables, databases, and partitions.
- If you are not using AWS Glue Data Catalog, the number of partitions per table is 20,000. You can [request a quota increase](#).

### Note

If you have not yet migrated to AWS Glue Data Catalog, see [Upgrading to the AWS Glue Data Catalog Step-by-Step \(p. 29\)](#) for migration instructions.

## Amazon S3 Buckets

When you work with Amazon S3 buckets, remember the following points:

- Amazon S3 has a default service quota of 100 buckets per account.
- Athena requires a separate bucket to log results.
- You can request a quota increase of up to 1,000 Amazon S3 buckets per AWS account.

## Per Account API Call Quotas

Athena APIs have the following default quotas for the number of calls to the API per account (not per query):

API Name	Default Number of Calls per Second	Burst Capacity
BatchGetNamedQuery, ListNamedQueries, ListQueryExecutions	5	up to 10
CreateNamedQuery, DeleteNamedQuery, GetNamedQuery	5	up to 20
BatchGetQueryExecution	20	up to 40
StartQueryExecution, StopQueryExecution	20	up to 80
GetQueryExecution, GetQueryResults	100	up to 200

For example, for `StartQueryExecution`, you can make up to 20 calls per second. In addition, if this API is not called for 4 seconds, your account accumulates a *burst capacity* of up to 80 calls. In this case, your application can make up to 80 calls to this API in burst mode.

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. To request a quota increase, contact AWS Support. Open the [AWS Support Center](#) page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

**Note**

This quota cannot be changed in the Athena Service Quotas console.

# Release Notes

Describes Amazon Athena features, improvements, and bug fixes by release date.

## Release Dates

- [July 29, 2020 \(p. 442\)](#)
- [July 9, 2020 \(p. 442\)](#)
- [June 1, 2020 \(p. 443\)](#)
- [May 21, 2020 \(p. 443\)](#)
- [April 1, 2020 \(p. 443\)](#)
- [March 11, 2020 \(p. 443\)](#)
- [March 6, 2020 \(p. 443\)](#)
- [November 26, 2019 \(p. 444\)](#)
- [November 12, 2019 \(p. 446\)](#)
- [November 8, 2019 \(p. 447\)](#)
- [October 8, 2019 \(p. 447\)](#)
- [September 19, 2019 \(p. 447\)](#)
- [September 12, 2019 \(p. 447\)](#)
- [August 16, 2019 \(p. 448\)](#)
- [August 9, 2019 \(p. 448\)](#)
- [June 26, 2019 \(p. 448\)](#)
- [May 24, 2019 \(p. 448\)](#)
- [March 05, 2019 \(p. 448\)](#)
- [February 22, 2019 \(p. 449\)](#)
- [February 18, 2019 \(p. 450\)](#)
- [November 20, 2018 \(p. 451\)](#)
- [October 15, 2018 \(p. 451\)](#)
- [October 10, 2018 \(p. 452\)](#)
- [September 6, 2018 \(p. 452\)](#)
- [August 23, 2018 \(p. 452\)](#)
- [August 16, 2018 \(p. 453\)](#)
- [August 7, 2018 \(p. 453\)](#)
- [June 5, 2018 \(p. 454\)](#)
- [May 17, 2018 \(p. 454\)](#)
- [April 19, 2018 \(p. 455\)](#)
- [April 6, 2018 \(p. 455\)](#)
- [March 15, 2018 \(p. 455\)](#)
- [February 2, 2018 \(p. 455\)](#)
- [January 19, 2018 \(p. 456\)](#)

- [November 13, 2017 \(p. 456\)](#)
- [November 1, 2017 \(p. 457\)](#)
- [October 19, 2017 \(p. 457\)](#)
- [October 3, 2017 \(p. 457\)](#)
- [September 25, 2017 \(p. 457\)](#)
- [August 14, 2017 \(p. 457\)](#)
- [August 4, 2017 \(p. 457\)](#)
- [June 22, 2017 \(p. 457\)](#)
- [June 8, 2017 \(p. 458\)](#)
- [May 19, 2017 \(p. 458\)](#)
- [April 4, 2017 \(p. 459\)](#)
- [March 24, 2017 \(p. 460\)](#)
- [February 20, 2017 \(p. 460\)](#)

## 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 \(p. 57\)](#), Okta service for authentication, and connections to VPC endpoints.

To download and use the new version of the driver, see [Using Athena with the JDBC Driver \(p. 72\)](#).

## 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 [Using Athena to Query Apache Hudi Datasets \(p. 178\)](#).

### 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 \(p. 55\)](#) 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 \(p. 34\)](#).

## 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 [Partition Projection with Amazon Athena \(p. 96\)](#).

## April 1, 2020

Published on 2020-04-01

In addition to the US East (N. Virginia) Region, the Amazon Athena [federated query \(p. 56\)](#), [user defined functions \(UDFs\) \(p. 190\)](#), [machine learning inference \(p. 189\)](#), and [external Hive metastore \(p. 34\)](#) 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 CloudWatch 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 CloudWatch Events. The event contains information about the query state transition. For more information, see [Monitoring Athena Queries with CloudWatch Events \(p. 342\)](#).

## 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*.

## 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 Redshift](#), [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 [Deploying a Connector and Connecting to a Data Source](#).
- 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 as 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 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 [Using Athena to Query Data Registered With AWS Lake Formation \(p. 275\)](#). 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 [Connecting to Amazon Athena with ODBC \(p. 73\)](#). 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 \(p. 336\)](#) to create and manage workgroups, and [Athena API tag actions \(p. 350\)](#) 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:

- [Using Workgroups for Running Queries \(p. 322\)](#) and [Workgroup Example Policies \(p. 326\)](#).
- [Tagging Resources \(p. 348\)](#) and [Tag-Based IAM Access Control Policies \(p. 353\)](#).

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 [Using Athena with the JDBC Driver \(p. 72\)](#).

## 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 [Tagging Workgroups \(p. 348\)](#).

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 [Working with Tags Using the API Actions \(p. 348\)](#).

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 [Tag-Based IAM Access Control Policies \(p. 353\)](#).

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 [Using Athena with the JDBC Driver \(p. 72\)](#).

## February 18, 2019

Published on 2019-02-18

Added ability to control query costs by running queries in workgroups. For information, see [Using Workgroups to Control Query Access and Costs \(p. 322\)](#). 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 [Using Workgroups for Running Queries \(p. 322\)](#) and [Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events \(p. 338\)](#).

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 *IAM User Guide*. Before you create new workgroups, make sure that you use [workgroup IAM policies \(p. 325\)](#), and the [AmazonAthenaFullAccess Managed Policy \(p. 240\)](#).

You can start using workgroups in the console, with the [workgroup API operations \(p. 336\)](#), or with the JDBC driver. For a high-level procedure, see [Setting up Workgroups \(p. 324\)](#). To download the JDBC driver with workgroup support, see [Using Athena with the JDBC Driver \(p. 72\)](#).

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://<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 [Athena Workgroup APIs \(p. 336\)](#) and [Troubleshooting Workgroups \(p. 336\)](#).

- 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 \(p. 375\)](#).

- 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 "Requirements for Tables in Athena and Data in Amazon S3" \(p. 79\)](#) and [Transitioning to the GLACIER Storage Class \(Object Archival\)](#) in the *Amazon Simple Storage Service Developer 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 "Querying Network Load Balancer Logs" \(p. 214\)](#).

## 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 "Enabling Federated Access to the Athena API" \(p. 268\)](#).

To download the JDBC driver version 2.0.6 and its documentation, see [Using Athena with the JDBC Driver \(p. 72\)](#). For information about this version, see [JDBC Driver Release Notes](#).

To download the ODBC driver version 1.0.4 and its documentation, see [Connecting to Amazon Athena with ODBC \(p. 73\)](#). 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 \(p. 250\)](#).
- 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 [Fine-Grained Access to Databases and Tables in the AWS Glue Data Catalog \(p. 244\)](#).

### Note

Data resides in the Amazon S3 buckets, and access to it is governed by the [Amazon S3 Permissions \(p. 243\)](#). 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 \(p. 124\)](#).

Use CTAS queries to:

- [Create a table from query results \(p. 124\)](#) in one step.
- [Create CTAS queries in the Athena console \(p. 126\)](#), using [Examples \(p. 130\)](#). For information about syntax, see [CREATE TABLE AS \(p. 410\)](#).
- 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 \(p. 124\)](#) and [Columnar Storage Formats \(p. 88\)](#).

## 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 [Connecting to Amazon Athena with ODBC \(p. 73\)](#).

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 [AWSQuicksightAthenaAccess Managed Policy \(p. 242\)](#).

## 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 \(p. 390\)](#).

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 [Using Athena with the JDBC Driver \(p. 72\)](#).

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 [AWSQuicksightAthenaAccess Managed Policy \(p. 242\)](#). 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 [Using the Previous Version of the JDBC Driver \(p. 433\)](#), and 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 [Querying Amazon VPC Flow Logs \(p. 216\)](#) and [Amazon VPC Flow Logs can now be delivered to S3](#).

## June 5, 2018

Published on 2018-06-05

### Topics

- [Support for Views \(p. 454\)](#)
- [Improvements and Updates to Error Messages \(p. 454\)](#)
- [Bug Fixes \(p. 454\)](#)

## Support for Views

Added support for views. You can now use [CREATE VIEW \(p. 412\)](#), [DESCRIBE VIEW \(p. 414\)](#), [DROP VIEW \(p. 415\)](#), [SHOW CREATE VIEW \(p. 418\)](#), and [SHOW VIEWS \(p. 420\)](#) in Athena. The query that defines the view runs each time you reference the view in your query. For more information, see [Working with Views \(p. 119\)](#).

## 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 [Handling Schema Updates \(p. 142\)](#).
- 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 \(p. 438\)](#). 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 [Using Athena with the JDBC Driver \(p. 72\)](#).

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](#).

For information about the previous version of the JDBC driver, see [Using Athena with the Previous Version of the JDBC Driver \(p. 433\)](#).

## 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 [Using the CloudTrail Console to Create an Athena Table for CloudTrail Logs \(p. 205\)](#).

## 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 \(p. 378\)](#) and [OpenCSV SerDe \(p. 369\)](#), 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 a complete list of functions and operators, see [SQL Queries, Functions, and Operators \(p. 391\)](#) in this guide, and [Presto 0.172 Functions](#).

Athena does not support all of Presto's features. For more information, see [Limitations \(p. 421\)](#).

## November 13, 2017

Published on 2017-11-13

Added support for connecting Athena to the ODBC Driver. For information, see [Connecting to Amazon Athena with ODBC \(p. 73\)](#).

## 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 [Querying Geospatial Data \(p. 167\)](#) 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 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 [Integration with AWS Glue \(p. 16\)](#).

## 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 \(p. 372\)](#). 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://s3.amazonaws.com/athena-downloads/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 [Using Athena with the JDBC Driver](#) (p. 72).
- 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
<code>athena:RunQuery</code>	<code>athena:StartQueryExecution</code>
<code>athena:CancelQueryExecution</code>	<code>athena:StopQueryExecution</code>
<code>athena:GetQueryExecutions</code>	<code>athena:ListQueryExecutions</code>

## 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 \(p. 233\)](#).

## 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. For more information, see [Using Athena with the Previous Version of the JDBC Driver \(p. 433\)](#) for the previous version of the driver, and [Connect with the JDBC \(p. 72\)](#), for the most current version.
- 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. For more information, see [CloudTrail SerDe \(p. 367\)](#). For detailed usage examples, see the AWS Big Data Blog post, [Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena](#).

### 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 \(p. 364\)](#)
  - [OpenCSVSerDe for Processing CSV \(p. 369\)](#)
- **US East (Ohio) Region (us-east-2) launch.** You can now run queries in this region.
- You can now use the **Add Table** wizard to define table schema in bulk. Choose **Catalog Manager, Add table**, and then choose **Bulk add columns** as you walk through the steps to define the table.

Athena

Query Editor

Saved Queries

History

Catalog Manager

ACTION

+ Add table

Databases > Add table

Step 1: Name & Location

Step 2: Data Format

Step 3:

Column Name

Column Name

Column name must be single

Column type

string

Type for this column. Certain not exposed in this interface.

Add a column

Bulk add columns

Type name value pairs in the text box and choose **Add**.

Bulk add columns

Define columns in name value pairs, using commas to separate definitions (col1\_name data\_type, col2\_name data\_type, ...). Certain advanced data types (namely, structs) are not supported in this interface, but are supported using DDL statements.

id int, name string

Cancel

Add

## Improvements

- Improved performance on large Parquet tables.

# Document History

**Latest documentation update: October 5, 2020.**

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 documentation for using the JDBC driver with Lake Formation for federated access to Athena.	For more information, see <a href="#">Using Lake Formation and the Athena JDBC and ODBC Drivers for Federated Access to Athena</a> (p. 281) and <a href="#">Tutorial: Configuring Federated Access for Okta Users to Athena Using Lake Formation and JDBC</a> (p. 282).	September 25, 2020
Added documentation for the Amazon Athena Elasticsearch data connector.	For more information, see <a href="#">Amazon Athena Elasticsearch Connector</a> (p. 60).	July 21, 2020
Added documentation for querying Hudi datasets.	For more information, see <a href="#">Using Athena to Query Apache Hudi Datasets</a> (p. 178).	July 9, 2020
Added documentation on querying Apache web server logs and IIS web server logs stored in Amazon S3.	For more information, see <a href="#">Querying Apache Logs Stored in Amazon S3</a> (p. 225) and <a href="#">Querying Internet Information Server (IIS) Logs Stored in Amazon S3</a> (p. 226).	July 8, 2020
The Amazon Athena User Guide is now available in Kindle format.	The Kindle ebook is free of charge. For more information, see <a href="#">Amazon Athena: User Guide Kindle Edition</a> , or choose the <b>Kindle</b> link at the top of any page in the online version of the <a href="#">Amazon Athena User Guide</a> .	June 18, 2020
Added documentation for the general release of the Athena Data Connector for External Hive Metastore.	For more information, see <a href="#">Using Athena Data Connector for External Hive Metastore</a> (p. 34).	June 1, 2020
Added documentation for tagging data catalog resources.	For more information, see <a href="#">Tagging Resources</a> (p. 348).	June 1, 2020

Change	Description	Release Date
Added documentation on partition projection.	For more information, see <a href="#">Partition Projection with Amazon Athena (p. 96)</a> .	May 21, 2020
Updated the Java code examples for Athena.	For more information, see <a href="#">Code Samples (p. 423)</a> .	May 11, 2020
Added a topic on querying Amazon GuardDuty findings.	For more information, see <a href="#">Querying Amazon GuardDuty Findings (p. 213)</a> .	March 19, 2020
Added a topic on using CloudWatch Events to monitor Athena query state transitions.	For more information, see <a href="#">Monitoring Athena Queries with CloudWatch Events (p. 342)</a> .	March 11, 2020
Added a topic on querying AWS Global Accelerator flow logs with Athena.	For more information, see <a href="#">Querying AWS Global Accelerator Flow Logs (p. 211)</a> .	February 6, 2020
<ul style="list-style-type: none"> <li>Added documentation on using CTAS with INSERT INTO to add data from an unpartitioned source to a partitioned destination.</li> <li>Added download links for the 1.1.0 preview version of the ODBC driver for Athena.</li> <li>Description for SHOW DATABASES LIKE regex corrected.</li> <li>Corrected partitioned_by syntax in CTA topic.</li> <li>Other minor fixes.</li> </ul>	<p>Documentation updates include, but are not limited to, the following topics:</p> <ul style="list-style-type: none"> <li><a href="#">Using CTAS and INSERT INTO for ETL and Data Analysis (p. 133)</a></li> <li><a href="#">Connecting to Amazon Athena with ODBC (p. 73)</a> (The 1.1.0 preview features are now included in the 1.1.2 ODBC driver.)</li> <li><a href="#">SHOW DATABASES (p. 418)</a></li> <li><a href="#">CREATE TABLE AS (p. 410)</a></li> </ul>	February 4, 2020

Change	Description	Release Date
Added documentation on using CTAS with INSERT INTO to add data from a partitioned source to a partitioned destination.	For more information, see <a href="#">Using CTAS and INSERT INTO to Create a Table with More Than 100 Partitions (p. 139)</a> .	January 22, 2020
Query results location information updated.	Athena no longer creates a 'default' query results location. For more information, see <a href="#">Specifying a Query Result Location (p. 115)</a> .	January 20, 2020
Added topic on querying the AWS Glue Data Catalog. Updated information on service quotas (formerly "service limits") in Athena.	For more information, see the following topics: <ul style="list-style-type: none"> <li>• <a href="#">Querying AWS Glue Data Catalog (p. 221)</a></li> <li>• <a href="#">Service Quotas (p. 438)</a></li> </ul>	January 17, 2020
Corrected topic on OpenCSVSerDe to note that the <code>TIMESTAMP</code> type should be specified in the UNIX numeric format.	For more information, see <a href="#">OpenCSVSerDe for Processing CSV (p. 369)</a> .	January 15, 2020
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 <a href="#">Supported Amazon S3 Encryption Options (p. 234)</a> .	January 8, 2020
Added information on cross-account access to an Amazon S3 buckets that are encrypted with a custom AWS KMS key.	For more information, see <a href="#">Cross-account Access to a Bucket Encrypted with a Custom AWS KMS Key (p. 251)</a> .	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: <ul style="list-style-type: none"> <li>• <a href="#">Using Amazon Athena Federated Query (Preview) (p. 56)</a></li> <li>• <a href="#">Using Athena Data Source Connectors (p. 59)</a></li> <li>• <a href="#">Using Athena Data Connector for External Hive Metastore (p. 34)</a></li> <li>• <a href="#">Using Machine Learning (ML) with Amazon Athena (Preview) (p. 189)</a></li> <li>• <a href="#">Querying with User Defined Functions (Preview) (p. 190)</a></li> <li>• <a href="#">List of CloudWatch Metrics and Dimensions for Athena (p. 341)</a></li> </ul>	November 26, 2019

Change	Description	Release Date
Added section for new INSERT INTO command and updated query result location information for supporting data manifest files.	For more information, see <a href="#">INSERT INTO</a> (p. 396) and <a href="#">Working with Query Results, Output Files, and Query History</a> (p. 110).	September 18, 2019
Added section for interface VPC endpoints (PrivateLink) support. Updated JDBC drivers. Updated information on enriched VPC flow logs.	For more information, see <a href="#">Connect to Amazon Athena Using an Interface VPC Endpoint</a> (p. 273), <a href="#">Querying Amazon VPC Flow Logs</a> (p. 216), and <a href="#">Using Athena with the JDBC Driver</a> (p. 72).	September 11, 2019
Added section on integrating with AWS Lake Formation.	For more information, see <a href="#">Using Athena to Query Data Registered With AWS Lake Formation</a> (p. 275).	June 26, 2019
Updated Security section for consistency with other AWS services.	For more information, see <a href="#">Amazon Athena Security</a> (p. 232).	June 26, 2019
Added section on querying AWS WAF logs.	For more information, see <a href="#">Querying AWS WAF Logs</a> (p. 218).	May 31, 2019
Released the new version of the ODBC driver with support for Athena workgroups.	To download the ODBC driver version 1.0.5 and its documentation, see <a href="#">Connecting to Amazon Athena with ODBC</a> (p. 73). 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 <a href="#">Athena API workgroup actions</a> (p. 336) to create and manage workgroups, and <a href="#">Athena API tag actions</a> (p. 350) 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 <a href="#">Using Tags for Billing</a> in the <i>AWS Billing and Cost Management User Guide</i> .	February 22, 2019

Change	Description	Release Date
Improved the JSON OpenX SerDe used in Athena.	<p>The improvements include, but are not limited to, the following:</p> <ul style="list-style-type: none"> <li>Support for the <code>ConvertDotsInJsonKeysToUnderscores</code> property. When set to <code>TRUE</code>, 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 <code>"a.b"</code>, you can use this property to define the column name to be <code>"a_b"</code> in Athena. The default is <code>FALSE</code>. By default, Athena does not allow dots in column names.</li> <li>Support for the <code>case.insensitive</code> property. By default, Athena requires that all keys in your JSON dataset use lowercase. Using <code>WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)</code> allows you to use case-sensitive key names in your data. The default is <code>TRUE</code>. When set to <code>TRUE</code>, the SerDe converts all uppercase columns to lowercase.</li> </ul> <p>For more information, see <a href="#">OpenX JSON SerDe (p. 375)</a>.</p>	February 18, 2019
Added support for workgroups.	<p>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 <a href="#">Using Workgroups for Running Queries (p. 322)</a> and <a href="#">Controlling Costs and Monitoring Queries with CloudWatch Metrics and Events (p. 338)</a>.</p>	February 18, 2019
Added support for analyzing logs from Network Load Balancer.	<p>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 <a href="#">the section called "Querying Network Load Balancer Logs" (p. 214)</a>.</p>	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).	<p>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 <a href="#">the section called "Enabling Federated Access to the Athena API" (p. 268)</a>.</p>	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.	<p>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.</p> <p>Additionally, you can encrypt database and table metadata in the Data Catalog, by adding specific policies to Athena.</p> <p>For details, see <a href="#">Fine-Grained Access to Databases and Tables in the AWS Glue Data Catalog</a> (p. 244).</p>	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 <a href="#">Creating a Table from Query Results</a> (p. 124), <a href="#">Considerations and Limitations</a> (p. 124), and <a href="#">Examples</a> (p. 130).	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.	<p>The ODBC driver version 1.0.3 supports streaming results and also includes improvements, bug fixes, and an updated documentation for <i>"Using SSL with a Proxy Server"</i>.</p> <p>For downloading the ODBC driver version 1.0.3 and its documentation, see <a href="#">Connecting to Amazon Athena with ODBC</a> (p. 73).</p>	September 6, 2018
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 documentation.	Released the JDBC driver 2.0.5 with default support for streaming results instead of fetching them in pages. For information, see <a href="#">Using Athena with the JDBC Driver</a> (p. 72).	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.	<p>Updated the documentation for querying Amazon Virtual Private Cloud flow logs, which can be stored directly in Amazon S3 in a GZIP format. For information, see <a href="#">Querying Amazon VPC Flow Logs</a> (p. 216).</p> <p>Updated examples for querying ALB logs. For information, see <a href="#">Querying Application Load Balancer Logs</a> (p. 198).</p>	August 7, 2018

Change	Description	Release Date
Added support for views. Added guidelines for schema manipulations for various data storage formats.	Added support for views. For information, see <a href="#">Working with Views (p. 119)</a> .  Updated this guide with guidance on handling schema updates for various data storage formats. For information, see <a href="#">Handling Schema Updates (p. 142)</a> .	June 5, 2018
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 <a href="#">Service Quotas (p. 438)</a> .	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 <a href="#">Using the Console (p. 15)</a> .	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 <a href="#">Using Athena with the JDBC Driver (p. 72)</a> .	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 <a href="#">Using the CloudTrail Console to Create an Athena Table for CloudTrail Logs (p. 205)</a> .	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 <a href="#">February 2, 2018 (p. 455)</a> .	February 2, 2018
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 <a href="#">January 19, 2018 (p. 456)</a> .	January 19, 2018
Added support for the ODBC Driver.	Added support for connecting Athena to the ODBC Driver. For information, see <a href="#">Connecting to Amazon Athena with ODBC</a> .	November 13, 2017

Change	Description	Release Date
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 <a href="#">Querying Geospatial Data</a> and <a href="#">AWS Regions and Endpoints</a> .	November 1, 2017
Added support for Europe (Frankfurt).	Added support for Europe (Frankfurt). For a list of supported regions, see <a href="#">AWS Regions and Endpoints</a> .	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 <a href="#">AWS::Athena::NamedQuery</a> in the <i>AWS CloudFormation User Guide</i> .	October 3, 2017
Added support for Asia Pacific (Sydney).	Added support for Asia Pacific (Sydney). For a list of supported regions, see <a href="#">AWS Regions and Endpoints</a> .	September 25, 2017
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 <a href="#">Querying AWS Service Logs (p. 198)</a> and for querying different types of data in Athena. For information, see <a href="#">Running SQL Queries Using Amazon Athena (p. 110)</a> .	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 <a href="#">Integration with AWS Glue</a> and <a href="#">AWS Glue</a> .	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 <a href="#">Grok SerDe</a> . Added keyboard shortcuts to scroll through query history using the console (CTRL + ???/??? using Windows, CMD + ???/??? using Mac).	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 <a href="#">AWS Regions and Endpoints</a> .	June 22, 2017
Added support for Europe (Ireland).	Added support for Europe (Ireland). For more information, see <a href="#">AWS Regions and Endpoints</a> .	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
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 <a href="#">Encryption at Rest (p. 233)</a> .	April 4, 2017

Change	Description	Release Date
Added the AWS CloudTrail SerDe.	<p>Added the AWS CloudTrail SerDe, improved performance, fixed partition issues. For more information, see <a href="#">CloudTrail SerDe (p. 367)</a>.</p> <ul style="list-style-type: none"><li>• Improved performance when scanning a large number of partitions.</li><li>• Improved performance on <code>MSCK Repair Table</code> operation.</li><li>• 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.</li></ul>	March 24, 2017
Added support for US East (Ohio).	Added support for <a href="#">Avro SerDe (p. 364)</a> and <a href="#">OpenCSVSerDe for Processing CSV (p. 369)</a> , 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 <i>Amazon Athena User Guide</i> .	November, 2016

# AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.