
AWS Glue

Developer Guide



AWS Glue: Developer Guide

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What Is AWS Glue?	1
When Should I Use AWS Glue?	1
How It Works	3
Serverless ETL Jobs Run in Isolation	3
Concepts	4
AWS Glue Terminology	5
Components	6
AWS Glue Console	7
AWS Glue Data Catalog	7
AWS Glue Crawlers and Classifiers	7
AWS Glue ETL Operations	8
The AWS Glue Jobs System	8
Converting Semi-Structured Schemas to Relational Schemas	8
Getting Started	10
Setting up IAM Permissions for AWS Glue	10
Step 1: Create an IAM Policy for the AWS Glue Service	10
Step 2: Create an IAM Role for AWS Glue	14
Step 3: Attach a Policy to IAM Users That Access AWS Glue	15
Step 4: Create an IAM Policy for Notebook Servers	23
Step 5: Create an IAM Role for Notebook Servers	25
Step 6: Create an IAM Policy for Amazon SageMaker Notebooks	26
Step 7: Create an IAM Role for Amazon SageMaker Notebooks	28
Setting Up DNS in Your VPC	29
Setting Up Your Environment to Access Data Stores	30
Amazon VPC Endpoints for Amazon S3	30
Setting Up a VPC to Connect to JDBC Data Stores	32
Setting Up Your Environment for Development Endpoints	34
Setting Up Your Network for a Development Endpoint	34
Setting Up Amazon EC2 for a Notebook Server	36
Setting Up Encryption	36
Console Workflow Overview	39
Security	41
Data Protection	41
Encryption at Rest	41
Encryption in Transit	47
Key Management	47
Internetwork Traffic Privacy	48
AWS Glue Dependency on Other AWS Services	50
Development Endpoints	50
Identity and Access Management	50
Authentication	51
Access Control Overview	52
Cross-Account Access	61
Resource ARNs	65
Policy Examples	69
API Permissions Reference	84
Logging and Monitoring	101
Compliance Validation	101
Resilience	102
Infrastructure Security	102
Using AWS Glue with VPC Endpoints	102
Shared Amazon VPCs	103
Populating the AWS Glue Data Catalog	104
Defining a Database in Your Data Catalog	105

Working with Databases on the Console	105
Defining Tables in the AWS Glue Data Catalog	106
Table Partitions	106
Updating Manually Created Tables with Crawlers	107
Working with Tables on the Console	107
Adding a Connection to Your Data Store	110
When Is a Connection Used?	110
Defining a Connection in the AWS Glue Data Catalog	110
Connecting to a JDBC Data Store in a VPC	111
Working with Connections on the Console	112
Defining Crawlers	116
Which Data Stores Can I Crawl?	116
What Happens When a Crawler Runs?	117
Crawler Properties	118
Configuring a Crawler	123
Scheduling a Crawler	126
Working with Crawlers on the Console	127
Adding Classifiers to a Crawler	129
When Do I Use a Classifier?	129
Custom Classifiers	129
Built-In Classifiers in AWS Glue	130
Writing Custom Classifiers	132
Working with Classifiers on the Console	142
Working with Data Catalog Settings on the AWS Glue Console	144
Updating the Data Catalog with New Partitions	145
Populating the Data Catalog Using AWS CloudFormation Templates	146
Sample Database	147
Sample Database, Table, Partitions	148
Sample Grok Classifier	151
Sample JSON Classifier	151
Sample XML Classifier	152
Sample Amazon S3 Crawler	153
Sample Connection	154
Sample JDBC Crawler	155
Sample Job for Amazon S3 to Amazon S3	157
Sample Job for JDBC to Amazon S3	158
Sample On-Demand Trigger	159
Sample Scheduled Trigger	160
Sample Conditional Trigger	161
Sample Development Endpoint	161
Authoring Jobs	163
Workflow Overview	164
Adding Jobs	164
Defining Job Properties	165
Adding Python Shell Jobs	168
Built-In Transforms	172
Jobs on the Console	174
Editing Scripts	179
Defining a Script	179
Scripts on the Console	181
Providing Your Own Custom Scripts	181
Triggering Jobs	182
Triggering Jobs Based on Schedules or Events	183
Defining Trigger Types	183
Working with Triggers on the Console	183
Developing Scripts Using Development Endpoints	184
Development Endpoint Workflow	185

Adding a Development Endpoint	186
Viewing Development Endpoint Properties	187
Accessing Your Development Endpoint	189
Creating a Notebook Server Hosted on Amazon EC2	190
Tutorial Prerequisites	192
Tutorial: Local Zeppelin Notebook	195
Tutorial: Amazon EC2 Zeppelin Notebook Server	198
Tutorial: Use a SageMaker Notebook	200
Tutorial: Use a REPL Shell	202
Tutorial: Use PyCharm Professional	203
Managing Notebooks	209
Notebook Server Considerations	210
Working with Notebooks on the Console	217
Running and Monitoring	220
Automated Tools	221
Time-Based Schedules for Jobs and Crawlers	221
Cron Expressions	221
Tracking Processed Data Using Job Bookmarks	223
Using Job Bookmarks	224
Using an AWS Glue Script	225
Using Modification Timestamps	228
AWS Tags	229
Examples	230
Automating with CloudWatch Events	231
Monitoring with the Spark UI	233
Enabling the Spark UI for Jobs	237
Enabling the Spark UI for Dev Endpoints	238
Launching the Spark History Server	239
Monitoring with CloudWatch	242
Using CloudWatch Metrics	242
Setting Up Amazon CloudWatch Alarms on AWS Glue Job Profiles	256
Continuous Logging for AWS Glue Jobs	256
Job Monitoring and Debugging	259
Debugging OOM Exceptions and Job Abnormalities	260
Debugging Demanding Stages and Straggler Tasks	267
Monitoring the Progress of Multiple Jobs	271
Monitoring for DPU Capacity Planning	275
Logging Using CloudTrail	280
AWS Glue Information in CloudTrail	280
Understanding AWS Glue Log File Entries	281
Performing Complex ETL Activities Using Workflows	283
Overview of Workflows	283
Static and Dynamic Workflow Views	284
Workflow Restrictions	284
Creating and Running Workflows	284
Creating and Building Out a Workflow Using the Console	285
Running a Workflow	287
Getting and Setting Workflow Run Properties	287
Querying Workflows Using the AWS Glue API	288
Querying Static Views	288
Querying Dynamic Views	289
ETL Programming	292
General Information	292
Special Parameters	292
Connection Parameters	294
Format Options	299
Managing Partitions	302

Grouping Input Files	303
Reading from JDBC in Parallel	304
Moving Data to and from Amazon Redshift	305
Data Catalog Support for Spark SQL Jobs	306
Excluding Amazon S3 Storage Classes	308
Developing and Testing ETL Scripts Locally	310
ETL Programming in Python	314
Using Python	314
List of Extensions	314
List of Transforms	314
Python Setup	315
Calling APIs	315
Python Libraries	317
Python Samples	318
PySpark Extensions	332
PySpark Transforms	360
ETL Programming in Scala	387
Using Scala	392
Scala API List	393
Matching Records with FindMatches	437
Types of Machine Learning Transforms	437
Find Matches Transform	438
Tuning Machine Learning Transforms	441
Machine Learning Measurements	441
Deciding Between Precision and Recall	442
Deciding Between Accuracy and Cost	442
Teaching the Find Matches Transform	443
Machine Learning Transforms on the Console	444
Transform Properties	444
Adding and Editing Machine Learning Transforms	445
Viewing Transform Details	445
Tutorial: Creating a Machine Learning Transform	447
Step 1: Crawl the Source Data	448
Step 2: Add a Machine Learning Transform	448
Step 3: Teach Your Machine Learning Transform	448
Step 4: Estimate the Quality of Your Machine Learning Transform	449
Step 5: Add and Run a Job with Your Machine Learning Transform	449
Step 6: Verify Output Data from Amazon S3	451
AWS Glue API	452
Security	459
— data types —	459
DataCatalogEncryptionSettings	459
EncryptionAtRest	460
ConnectionPasswordEncryption	460
EncryptionConfiguration	461
S3Encryption	461
CloudWatchEncryption	461
JobBookmarksEncryption	461
SecurityConfiguration	462
— operations —	462
GetDataCatalogEncryptionSettings (get_data_catalog_encryption_settings)	462
PutDataCatalogEncryptionSettings (put_data_catalog_encryption_settings)	463
PutResourcePolicy (put_resource_policy)	463
GetResourcePolicy (get_resource_policy)	464
DeleteResourcePolicy (delete_resource_policy)	465
CreateSecurityConfiguration (create_security_configuration)	465
DeleteSecurityConfiguration (delete_security_configuration)	466

GetSecurityConfiguration (get_security_configuration)	466
GetSecurityConfigurations (get_security_configurations)	467
Catalog	467
Databases	468
Tables	473
Partitions	488
Connections	499
User-Defined Functions	506
Importing an Athena Catalog	511
Crawlers and Classifiers	512
Classifiers	513
Crawlers	522
Scheduler	533
Autogenerating ETL Scripts	535
— data types —	535
CodeGenNode	535
CodeGenNodeArg	536
CodeGenEdge	536
Location	536
CatalogEntry	537
MappingEntry	537
— operations —	537
CreateScript (create_script)	537
GetDataflowGraph (get_dataflow_graph)	538
GetMapping (get_mapping)	539
GetPlan (get_plan)	539
Jobs	540
Jobs	540
Job Runs	551
Triggers	561
Workflows	570
— data types —	570
JobNodeDetails	570
CrawlerNodeDetails	570
TriggerNodeDetails	570
Crawl	571
Node	571
Edge	572
WorkflowGraph	572
WorkflowRun	572
WorkflowRunStatistics	573
Workflow	573
— operations —	574
CreateWorkflow (create_workflow)	574
UpdateWorkflow (update_workflow)	575
DeleteWorkflow (delete_workflow)	576
ListWorkflows (list_workflows)	576
BatchGetWorkflows (batch_get_workflows)	577
GetWorkflowRun (get_workflow_run)	577
GetWorkflowRuns (get_workflow_runs)	578
GetWorkflowRunProperties (get_workflow_run_properties)	579
PutWorkflowRunProperties (put_workflow_run_properties)	579
DevEndpoints	580
— data types —	580
DevEndpoint	580
DevEndpointCustomLibraries	583
— operations —	583

CreateDevEndpoint (create_dev_endpoint)	583
UpdateDevEndpoint (update_dev_endpoint)	587
DeleteDevEndpoint (delete_dev_endpoint)	588
GetDevEndpoint (get_dev_endpoint)	588
GetDevEndpoints (get_dev_endpoints)	589
BatchGetDevEndpoints (batch_get_dev_endpoints)	590
ListDevEndpoints (list_dev_endpoints)	590
Machine Learning	591
— data types —	591
TransformParameters	592
EvaluationMetrics	592
MLTransform	592
FindMatchesParameters	594
FindMatchesMetrics	595
ConfusionMatrix	596
GlueTable	596
TaskRun	597
TransformFilterCriteria	597
TransformSortCriteria	598
TaskRunFilterCriteria	598
TaskRunSortCriteria	599
TaskRunProperties	599
FindMatchesTaskRunProperties	599
ImportLabelsTaskRunProperties	600
ExportLabelsTaskRunProperties	600
LabelingSetGenerationTaskRunProperties	600
SchemaColumn	600
— operations —	601
CreateMLTransform (create_ml_transform)	601
UpdateMLTransform (update_ml_transform)	603
DeleteMLTransform (delete_ml_transform)	605
GetMLTransform (get_ml_transform)	605
GetMLTransforms (get_ml_transforms)	607
StartMLEvaluationTaskRun (start_ml_evaluation_task_run)	608
StartMLLabelingSetGenerationTaskRun (start_ml_labeling_set_generation_task_run)	609
GetMLTaskRun (get_ml_task_run)	609
GetMLTaskRuns (get_ml_task_runs)	611
CancelMLTaskRun (cancel_ml_task_run)	611
StartExportLabelsTaskRun (start_export_labels_task_run)	612
StartImportLabelsTaskRun (start_import_labels_task_run)	613
Tagging APIs	614
— data types —	614
Tag	614
— operations —	614
TagResource (tag_resource)	614
UntagResource (untag_resource)	615
GetTags (get_tags)	616
Common Data Types	616
Tag	616
DecimalNumber	617
ErrorDetail	617
PropertyPredicate	617
ResourceUri	617
String Patterns	618
Exceptions	618
AccessDeniedException	618
AlreadyExistsException	618

ConcurrentModificationException	619
ConcurrentRunsExceededException	619
CrawlerNotRunningException	619
CrawlerRunningException	619
CrawlerStoppingException	619
EntityNotFoundException	620
GlueEncryptionException	620
IdempotentParameterMismatchException	620
InternalServiceException	620
InvalidExecutionEngineException	620
InvalidInputException	621
InvalidTaskStatusTransitionException	621
JobDefinitionErrorException	621
JobRunInTerminalStateException	621
JobRunInValidStateException	621
JobRunNotInTerminalStateException	622
LateRunnerException	622
NoScheduleException	622
OperationTimeoutException	622
ResourceNumberLimitExceededException	622
SchedulerNotRunningException	623
SchedulerRunningException	623
SchedulerTransitioningException	623
UnrecognizedRunnerException	623
ValidationException	623
VersionMismatchException	624
Troubleshooting	625
Gathering AWS Glue Troubleshooting Information	625
Troubleshooting Connection Issues	625
Troubleshooting Errors	626
Error: Resource Unavailable	626
Error: Could Not Find S3 Endpoint or NAT Gateway for subnetId in VPC	627
Error: Inbound Rule in Security Group Required	627
Error: Outbound Rule in Security Group Required	627
Error: Job Run Failed Because the Role Passed Should Be Given Assume Role Permissions for the AWS Glue Service	627
Error: DescribeVpcEndpoints Action Is Unauthorized. Unable to Validate VPC ID vpc-id	627
Error: DescribeRouteTables Action Is Unauthorized. Unable to Validate Subnet Id: subnet-id in VPC id: vpc-id	628
Error: Failed to Call ec2:DescribeSubnets	628
Error: Failed to Call ec2:DescribeSecurityGroups	628
Error: Could Not Find Subnet for AZ	628
Error: Job Run Exception When Writing to a JDBC Target	628
Error: Amazon S3 Timeout	629
Error: Amazon S3 Access Denied	629
Error: Amazon S3 Access Key ID Does Not Exist	629
Error: Job Run Fails When Accessing Amazon S3 with an s3a:// URI	629
Error: Amazon S3 Service Token Expired	630
Error: No Private DNS for Network Interface Found	631
Error: Development Endpoint Provisioning Failed	631
Error: Notebook Server CREATE_FAILED	631
Error: Local Notebook Fails to Start	631
Error: Notebook Usage Errors	631
Error: Running Crawler Failed	632
Error: Upgrading Athena Data Catalog	632
Error: A Job is Reprocessing Data When Job Bookmarks Are Enabled	632
AWS Glue Quotas	633

Known Issues	634
Preventing Cross-Job Data Access	634
AWS Glue Release Notes	636
AWS Glue Versions	636
Document History	637
Earlier Updates	644
AWS Glossary	645

What Is AWS Glue?

AWS Glue is a fully managed ETL (extract, transform, and load) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores. AWS Glue consists of a central metadata repository known as the AWS Glue Data Catalog, an ETL engine that automatically generates Python or Scala code, and a flexible scheduler that handles dependency resolution, job monitoring, and retries. AWS Glue is serverless, so there's no infrastructure to set up or manage.

AWS Glue is designed to work with semi-structured data. It introduces a component called a *dynamic frame*, which you can use in your ETL scripts. A dynamic frame is similar to an Apache Spark dataframe, except that each record is self-describing, so no schema is required initially. With dynamic frames, you get schema flexibility and a set of advanced transformations specifically designed for dynamic frames. You can convert between dynamic frames and Spark dataframes, so that you can take advantage of both AWS Glue and Spark transformations to do the kinds of analysis that you want.

You can use the AWS Glue console to discover data, transform it, and make it available for search and querying. The console calls the underlying services to orchestrate the work required to transform your data. You can also use the AWS Glue API operations to interface with AWS Glue services. Edit, debug, and test your Python or Scala Apache Spark ETL code using a familiar development environment.

For pricing information, see [AWS Glue Pricing](#).

When Should I Use AWS Glue?

You can use AWS Glue to organize, cleanse, validate, and format data for storage in a data warehouse or data lake. You can transform and move AWS Cloud data into your data store. You can also load data from disparate sources into your data warehouse or data lake for regular reporting and analysis. By storing data in a data warehouse or data lake, you integrate information from different parts of your business and provide a common source of data for decision making.

AWS Glue simplifies many tasks when you are building a data warehouse or data lake:

- Discovers and catalogs metadata about your data stores into a central catalog. You can process semi-structured data, such as clickstream or process logs.
- Populates the AWS Glue Data Catalog with table definitions from scheduled crawler programs. Crawlers call classifier logic to infer the schema, format, and data types of your data. This metadata is stored as tables in the AWS Glue Data Catalog and used in the authoring process of your ETL jobs.
- Generates ETL scripts to transform, flatten, and enrich your data from source to target.
- Detects schema changes and adapts based on your preferences.
- Triggers your ETL jobs based on a schedule or event. You can initiate jobs automatically to move your data into your data warehouse or data lake. Triggers can be used to create a dependency flow between jobs.
- Gathers runtime metrics to monitor the activities of your data warehouse or data lake.
- Handles errors and retries automatically.
- Scales resources, as needed, to run your jobs.

You can use AWS Glue when you run serverless queries against your Amazon S3 data lake. AWS Glue can catalog your Amazon Simple Storage Service (Amazon S3) data, making it available for querying with Amazon Athena and Amazon Redshift Spectrum. With crawlers, your metadata stays in sync with

the underlying data. Athena and Redshift Spectrum can directly query your Amazon S3 data lake using the AWS Glue Data Catalog. With AWS Glue, you access and analyze data through one unified interface without loading it into multiple data silos.

You can create event-driven ETL pipelines with AWS Glue. You can run your ETL jobs as soon as new data becomes available in Amazon S3 by invoking your AWS Glue ETL jobs from an AWS Lambda function. You can also register this new dataset in the AWS Glue Data Catalog as part of your ETL jobs.

You can use AWS Glue to understand your data assets. You can store your data using various AWS services and still maintain a unified view of your data using the AWS Glue Data Catalog. View the Data Catalog to quickly search and discover the datasets that you own, and maintain the relevant metadata in one central repository. The Data Catalog also serves as a drop-in replacement for your external Apache Hive Metastore.

AWS Glue: How It Works

AWS Glue uses other AWS services to orchestrate your ETL (extract, transform, and load) jobs to build a data warehouse. AWS Glue calls API operations to transform your data, create runtime logs, store your job logic, and create notifications to help you monitor your job runs. The AWS Glue console connects these services into a managed application, so you can focus on creating and monitoring your ETL work. The console performs administrative and job development operations on your behalf. You supply credentials and other properties to AWS Glue to access your data sources and write to your data warehouse.

AWS Glue takes care of provisioning and managing the resources that are required to run your workload. You don't need to create the infrastructure for an ETL tool because AWS Glue does it for you. When resources are required, to reduce startup time, AWS Glue uses an instance from its warm pool of instances to run your workload.

With AWS Glue, you create jobs using table definitions in your Data Catalog. Jobs consist of scripts that contain the programming logic that performs the transformation. You use triggers to initiate jobs either on a schedule or as a result of a specified event. You determine where your target data resides and which source data populates your target. With your input, AWS Glue generates the code that's required to transform your data from source to target. You can also provide scripts in the AWS Glue console or API to process your data.

AWS Glue is available in several AWS Regions. For more information, see [AWS Regions and Endpoints](#) in the Amazon Web Services General Reference.

Topics

- [Serverless ETL Jobs Run in Isolation \(p. 3\)](#)
- [AWS Glue Concepts \(p. 4\)](#)
- [AWS Glue Components \(p. 6\)](#)
- [Converting Semi-Structured Schemas to Relational Schemas \(p. 8\)](#)

Serverless ETL Jobs Run in Isolation

AWS Glue runs your ETL jobs in an Apache Spark serverless environment. AWS Glue runs these jobs on virtual resources that it provisions and manages in its own service account.

AWS Glue is designed to do the following:

- Segregate customer data.
- Protect customer data in transit and at rest.
- Access customer data only as needed in response to customer requests, using temporary, scoped-down credentials, or with a customer's consent to IAM roles in their account.

During provisioning of an ETL job, you provide input data sources and output data targets in your virtual private cloud (VPC). In addition, you provide the IAM role, VPC ID, subnet ID, and security group that are needed to access data sources and targets. For each tuple (customer account ID, IAM role, subnet

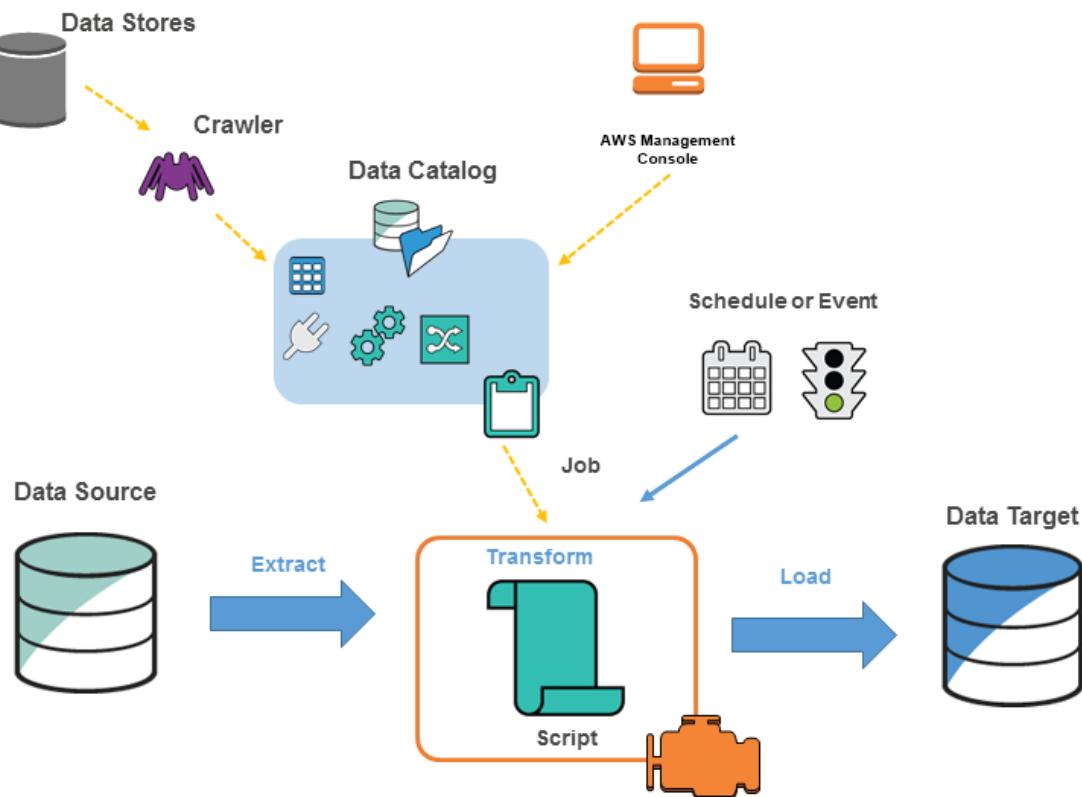
ID, and security group), AWS Glue creates a new Spark environment that is isolated at the network and management level from all other Spark environments inside the AWS Glue service account.

AWS Glue creates elastic network interfaces in your subnet using private IP addresses. Spark jobs use these elastic network interfaces to access your data sources and data targets. Traffic in, out, and within the Spark environment is governed by your VPC and networking policies with one exception: Calls made to AWS Glue libraries can proxy traffic to AWS Glue API operations through the AWS Glue VPC. All AWS Glue API calls are logged; thus, data owners can audit API access by enabling [AWS CloudTrail](#), which delivers audit logs to your account.

AWS Glue managed Spark environments that run your ETL jobs are protected with the same security practices followed by other AWS services. Those practices are listed in the [AWS Access](#) section of the [Introduction to AWS Security Processes](#) whitepaper.

AWS Glue Concepts

The following diagram shows the architecture of an AWS Glue environment.



You define *jobs* in AWS Glue to accomplish the work that's required to extract, transform, and load (ETL) data from a data source to a data target. You typically perform the following actions:

- You define a *crawler* to populate your AWS Glue Data Catalog with metadata table definitions. You point your crawler at a data store, and the crawler creates table definitions in the Data Catalog.

In addition to table definitions, the AWS Glue Data Catalog contains other metadata that is required to define ETL jobs. You use this metadata when you define a job to transform your data.

- AWS Glue can generate a script to transform your data. Or, you can provide the script in the AWS Glue console or API.
- You can run your job on demand, or you can set it up to start when a specified *trigger* occurs. The trigger can be a time-based schedule or an event.

When your job runs, a script extracts data from your data source, transforms the data, and loads it to your data target. The script runs in an Apache Spark environment in AWS Glue.

Important

Tables and databases in AWS Glue are objects in the AWS Glue Data Catalog. They contain metadata; they don't contain data from a data store.

Text-based data, such as CSVs, must be encoded in UTF-8 for AWS Glue to process it successfully. For more information, see [UTF-8](#) in Wikipedia.

AWS Glue Terminology

AWS Glue relies on the interaction of several components to create and manage your extract, transfer, and load (ETL) workflow.

AWS Glue Data Catalog

The persistent metadata store in AWS Glue. Each AWS account has one AWS Glue Data Catalog. It contains table definitions, job definitions, and other control information to manage your AWS Glue environment.

Classifier

Determines the schema of your data. AWS Glue provides classifiers for common file types, such as CSV, JSON, AVRO, XML, and others. It also provides classifiers for common relational database management systems using a JDBC connection. You can write your own classifier by using a grok pattern or by specifying a row tag in an XML document.

Connection

Contains the properties that are required to connect to your data store.

Crawler

A program that connects to a data store (source or target), progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata in the AWS Glue Data Catalog.

Database

A set of associated table definitions organized into a logical group in AWS Glue.

Data store, data source, data target

A *data store* is a repository for persistently storing your data. Examples include Amazon S3 buckets and relational databases. A *data source* is a data store that is used as input to a process or transform. A *data target* is a data store that a process or transform writes to.

Development endpoint

An environment that you can use to develop and test your AWS Glue scripts.

Dynamic Frame

A distributed table that supports nested data such as structures and arrays. Each record is self-describing, designed for schema flexibility with semi-structured data. Each record contains both data and the schema that describes that data. You can use both dynamic frames and Apache Spark dataframes in your ETL scripts, and convert between them. Dynamic frames provide a set of advanced transformations for data cleaning and ETL.

Job

The business logic that is required to perform ETL work. It is composed of a transformation script, data sources, and data targets. Job runs are initiated by triggers that can be scheduled or triggered by events.

Notebook server

A web-based environment that you can use to run your PySpark statements. For more information, see [Apache Zeppelin](#). You can set up a notebook server on a development endpoint to run PySpark statements with AWS Glue extensions.

Script

Code that extracts data from sources, transforms it, and loads it into targets. AWS Glue generates PySpark or Scala scripts. PySpark is a Python dialect for ETL programming.

Table

The metadata definition that represents your data. Whether your data is in an Amazon Simple Storage Service (Amazon S3) file, an Amazon Relational Database Service (Amazon RDS) table, or another set of data, a table defines the schema of your data. A table in the AWS Glue Data Catalog consists of the names of columns, data type definitions, and other metadata about a base dataset. The schema of your data is represented in your AWS Glue table definition. The actual data remains in its original data store, whether it be in a file or a relational database table. AWS Glue catalogs your files and relational database tables in the AWS Glue Data Catalog. They are used as sources and targets when you create an ETL job.

Transform

The code logic that is used to manipulate your data into a different format.

Trigger

Initiates an ETL job. Triggers can be defined based on a scheduled time or an event.

AWS Glue Components

AWS Glue provides a console and API operations to set up and manage your extract, transform, and load (ETL) workload. You can use API operations through several language-specific SDKs and the AWS Command Line Interface (AWS CLI). For information about using the AWS CLI, see [AWS CLI Command Reference](#).

AWS Glue uses the AWS Glue Data Catalog to store metadata about data sources, transforms, and targets. The Data Catalog is a drop-in replacement for the Apache Hive Metastore. The AWS Glue Jobs system provides a managed infrastructure for defining, scheduling, and running ETL operations on your data. For more information about the AWS Glue API, see [AWS Glue API \(p. 452\)](#).

AWS Glue Console

You use the AWS Glue console to define and orchestrate your ETL workflow. The console calls several API operations in the AWS Glue Data Catalog and AWS Glue Jobs system to perform the following tasks:

- Define AWS Glue objects such as jobs, tables, crawlers, and connections.
- Schedule when crawlers run.
- Define events or schedules for job triggers.
- Search and filter lists of AWS Glue objects.
- Edit transformation scripts.

AWS Glue Data Catalog

The AWS Glue Data Catalog is your persistent metadata store. It is a managed service that lets you store, annotate, and share metadata in the AWS Cloud in the same way you would in an Apache Hive metastore.

Each AWS account has one AWS Glue Data Catalog per AWS region. It provides a uniform repository where disparate systems can store and find metadata to keep track of data in data silos, and use that metadata to query and transform the data.

You can use AWS Identity and Access Management (IAM) policies to control access to the data sources managed by the AWS Glue Data Catalog. These policies allow different groups in your enterprise to safely publish data to the wider organization while protecting sensitive information. IAM policies let you clearly and consistently define which users have access to which data, regardless of its location.

For information about how to use the AWS Glue Data Catalog, see [Populating the AWS Glue Data Catalog \(p. 104\)](#). For information about how to program using the Data Catalog API, see [Catalog API \(p. 467\)](#).

Other AWS services and open source projects can use the AWS Glue Data Catalog:

- Amazon Athena – for more information, see [Understanding Tables, Databases, and the Data Catalog](#) in the Amazon Athena User Guide.
- Amazon Redshift Spectrum – for more information, see [Using Amazon Redshift Spectrum to Query External Data](#) in the Amazon Redshift Database Developer Guide.
- Amazon EMR – for more information, see [Use Resource-Based Policies for Amazon EMR Access to AWS Glue Data Catalog](#) in the Amazon EMR Management Guide.
- AWS Glue Data Catalog Client for Apache Hive Metastore – for more information about this GitHub project, see [AWS Glue Data Catalog Client for Apache Hive Metastore](#).

AWS Glue Crawlers and Classifiers

AWS Glue also lets you set up crawlers that can scan data in all kinds of repositories, classify it, extract schema information from it, and store the metadata automatically in the AWS Glue Data Catalog. From there it can be used to guide ETL operations.

For information about how to set up crawlers and classifiers, see [Defining Crawlers \(p. 116\)](#). For information about how to program crawlers and classifiers using the AWS Glue API, see [Crawlers and Classifiers API \(p. 512\)](#).

AWS Glue ETL Operations

Using the metadata in the Data Catalog, AWS Glue can autogenerate Scala or PySpark (the Python API for Apache Spark) scripts with AWS Glue extensions that you can use and modify to perform various ETL operations. For example, you can extract, clean, and transform raw data, and then store the result in a different repository, where it can be queried and analyzed. Such a script might convert a CSV file into a relational form and save it in Amazon Redshift.

For more information about how to use AWS Glue ETL capabilities, see [Programming ETL Scripts \(p. 292\)](#).

The AWS Glue Jobs System

The AWS Glue Jobs system provides managed infrastructure to orchestrate your ETL workflow. You can create jobs in AWS Glue that automate the scripts you use to extract, transform, and transfer data to different locations. Jobs can be scheduled and chained, or they can be triggered by events such as the arrival of new data.

For more information about using the AWS Glue Jobs system, see [Running and Monitoring AWS Glue \(p. 220\)](#). For information about programming using the AWS Glue Jobs system API, see [Jobs API \(p. 540\)](#).

Converting Semi-Structured Schemas to Relational Schemas

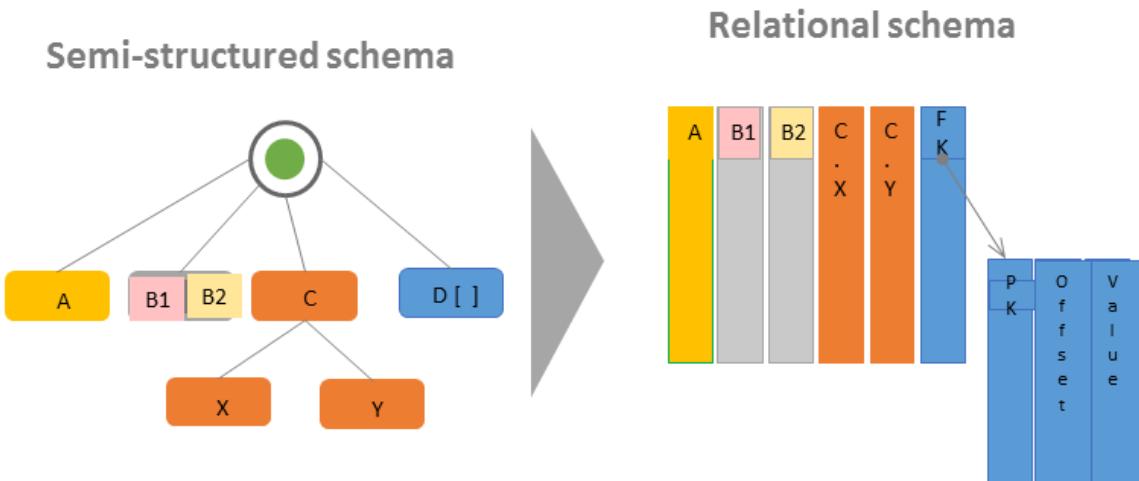
It's common to want to convert semi-structured data into relational tables. Conceptually, you are flattening a hierarchical schema to a relational schema. AWS Glue can perform this conversion for you on-the-fly.

Semi-structured data typically contains mark-up to identify entities within the data. It can have nested data structures with no fixed schema. For more information about semi-structured data, see [Semi-structured data](#) in Wikipedia.

Relational data is represented by tables that consist of rows and columns. Relationships between tables can be represented by a primary key (PK) to foreign key (FK) relationship. For more information, see [Relational database](#) in Wikipedia.

AWS Glue uses crawlers to infer schemas for semi-structured data. It then transforms the data to a relational schema using an ETL (extract, transform, and load) job. For example, you might want to parse JSON data from Amazon Simple Storage Service (Amazon S3) source files to Amazon Relational Database Service (Amazon RDS) tables. Understanding how AWS Glue handles the differences between schemas can help you understand the transformation process.

This diagram shows how AWS Glue transforms a semi-structured schema to a relational schema.



The diagram illustrates the following:

- Single value A converts directly to a relational column.
- The pair of values, B1 and B2, convert to two relational columns.
- Structure C, with children X and Y, converts to two relational columns.
- Array D[] converts to a relational column with a foreign key (FK) that points to another relational table. Along with a primary key (PK), the second relational table has columns that contain the offset and value of the items in the array.

Getting Started Using AWS Glue

The following sections provide an overview and walk you through setting up and using AWS Glue. For information about AWS Glue concepts and components, see [AWS Glue: How It Works \(p. 3\)](#).

Topics

- [Setting up IAM Permissions for AWS Glue \(p. 10\)](#)
- [Setting Up DNS in Your VPC \(p. 29\)](#)
- [Setting Up Your Environment to Access Data Stores \(p. 30\)](#)
- [Setting Up Your Environment for Development Endpoints \(p. 34\)](#)
- [Setting Up Encryption in AWS Glue \(p. 36\)](#)
- [AWS Glue Console Workflow Overview \(p. 39\)](#)

Setting up IAM Permissions for AWS Glue

You use AWS Identity and Access Management (IAM) to define policies and roles that are needed to access resources used by AWS Glue. The following steps lead you through the basic permissions that you need to set up your environment. Depending on your business needs, you might have to add or reduce access to your resources.

1. [Create an IAM Policy for the AWS Glue Service \(p. 10\)](#): Create a service policy that allows access to AWS Glue resources.
2. [Create an IAM Role for AWS Glue \(p. 14\)](#): Create an IAM role, and attach the AWS Glue service policy and a policy for your Amazon Simple Storage Service (Amazon S3) resources that are used by AWS Glue.
3. [Attach a Policy to IAM Users That Access AWS Glue \(p. 15\)](#): Attach policies to any IAM user that signs in to the AWS Glue console.
4. [Create an IAM Policy for Notebooks \(p. 23\)](#): Create a notebook server policy to use in the creation of notebook servers on development endpoints.
5. [Create an IAM Role for Notebooks \(p. 25\)](#): Create an IAM role and attach the notebook server policy.
6. [Create an IAM Policy for Amazon SageMaker Notebooks \(p. 26\)](#): Create an IAM policy to use when creating Amazon SageMaker notebooks on development endpoints.
7. [Create an IAM Role for Amazon SageMaker Notebooks \(p. 28\)](#): Create an IAM role and attach the policy to grant permissions when creating Amazon SageMaker notebooks on development endpoints.

Step 1: Create an IAM Policy for the AWS Glue Service

For any operation that accesses data on another AWS resource, such as accessing your objects in Amazon S3, AWS Glue needs permission to access the resource on your behalf. You provide those permissions by using AWS Identity and Access Management (IAM).

Note

You can skip this step if you use the AWS managed policy **AWSGlueServiceRole**.

In this step, you create a policy that is similar to `AWSGlueServiceRole`. You can find the most current version of `AWSGlueServiceRole` on the IAM console.

To create an IAM policy for AWS Glue

This policy grants permission for some Amazon S3 actions to manage resources in your account that are needed by AWS Glue when it assumes the role using this policy. Some of the resources that are specified

in this policy refer to default names that are used by AWS Glue for Amazon S3 buckets, Amazon S3 ETL scripts, CloudWatch Logs, and Amazon EC2 resources. For simplicity, AWS Glue writes some Amazon S3 objects into buckets in your account prefixed with `aws-glue-*` by default.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. On the **Create Policy** screen, navigate to a tab to edit JSON. Create a policy document with the following JSON statements, and then choose **Review policy**.

Note

Add any permissions needed for Amazon S3 resources. You might want to scope the resources section of your access policy to only those resources that are required.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "s3:GetBucketLocation",
                "s3>ListBucket",
                "s3>ListAllMyBuckets",
                "s3:GetBucketAcl",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2>CreateNetworkInterface",
                "ec2>DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "iam>ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>CreateBucket"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3>DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-/*",
                "arn:aws:s3:::/*aws-glue-/*"
            ]
        }
    ]
}
```

```

},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::crawler-public*",
        "arn:aws:s3:::aws-glue-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs:PutLogEvents",
        "logs:AssociateKmsKey"
    ],
    "Resource": [
        "arn:aws:logs:*::aws-glue/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateTags",
        "ec2>DeleteTags"
    ],
    "Condition": {
        "ForAllValues:StringEquals": {
            "aws:TagKeys": [
                "aws-glue-service-resource"
            ]
        }
    },
    "Resource": [
        "arn:aws:ec2::::network-interface/*",
        "arn:aws:ec2::::security-group/*",
        "arn:aws:ec2::::instance/*"
    ]
}
]
}

```

The following table describes the permissions granted by this policy.

Action	Resource	Description
"glue:/*"	"*"	Grants permission to run all AWS Glue API operations.
"s3:GetBucketLocation", "s3>ListBucket", "s3>ListAllMyBuckets", "s3:GetBucketAcl",	"*"	Allows listing of Amazon S3 buckets from crawlers, jobs, development endpoints, and notebook servers.

Action	Resource	Description
"ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2>CreateNetworkInterface", "ec2>DeleteNetworkInterface", "ec2:DescribeNetworkInterfaces", "ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcAttribute",	"*"	Allows the setup of Amazon EC2 network items, such as virtual private clouds (VPCs) when running jobs, crawlers, and development endpoints.
"iam>ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy"	"*"	Allows listing IAM roles from crawlers, jobs, development endpoints, and notebook servers.
"cloudwatch:PutMetricData"	"*"	Allows writing CloudWatch metrics for jobs.
"s3>CreateBucket"	"arn:aws:s3:::aws-glue-*"	Allows the creation of Amazon S3 buckets in your account from jobs and notebook servers. Naming convention: Uses Amazon S3 folders named aws-glue- .
"s3GetObject", "s3PutObject", "s3DeleteObject"	"arn:aws:s3:::aws-glue-*/*", "arn:aws:s3:::/*aws-glue-*/*"	Allows get, put, and delete of Amazon S3 objects into your account when storing objects such as ETL scripts and notebook server locations. Naming convention: Grants permission to Amazon S3 buckets or folders whose names are prefixed with aws-glue- .
"s3GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	Allows get of Amazon S3 objects used by examples and tutorials from crawlers and jobs. Naming convention: Amazon S3 bucket names begin with crawler-public and aws-glue- .
"logs>CreateLogGroup", "logs>CreateLogStream", "logsPutLogEvents"	"arn:aws:logs:*:*:/aws-glue/*"	Allows writing logs to CloudWatch Logs. Naming convention: AWS Glue writes logs to log groups whose names begin with aws-glue .

Action	Resource	Description
"ec2:CreateTags", "ec2:DeleteTags"	"arn:aws:ec2:*::network-interface/*", "arn:aws:ec2:*::security-group/*", "arn:aws:ec2:*::instance/*"	Allows tagging of Amazon EC2 resources created for development endpoints. Naming convention: AWS Glue tags Amazon EC2 network interfaces, security groups, and instances with aws-glue-service-resource .

- On the **Review Policy** screen, enter your **Policy Name**, for example **GlueServiceRolePolicy**. Enter an optional description, and when you're satisfied with the policy, choose **Create policy**.

Step 2: Create an IAM Role for AWS Glue

You need to grant your IAM role permissions that AWS Glue can assume when calling other services on your behalf. This includes access to Amazon S3 for any sources, targets, scripts, and temporary directories that you use with AWS Glue. Permission is needed by crawlers, jobs, and development endpoints.

You provide those permissions by using AWS Identity and Access Management (IAM). Add a policy to the IAM role that you pass to AWS Glue.

To create an IAM role for AWS Glue

- Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
- In the left navigation pane, choose **Roles**.
- Choose **Create role**.
- For role type, choose **AWS Service**, find and choose **Glue**, and choose **Next: Permissions**.
- On the **Attach permissions policy** page, choose the policies that contain the required permissions; for example, the AWS managed policy **AWSGlueServiceRole** for general AWS Glue permissions and the AWS managed policy **AmazonS3FullAccess** for access to Amazon S3 resources. Then choose **Next: Review**.

Note

Ensure that one of the policies in this role grants permissions to your Amazon S3 sources and targets. You might want to provide your own policy for access to specific Amazon S3 resources. Data sources require s3>ListBucket and s3:GetObject permissions. Data targets require s3>ListBucket, s3>PutObject, and s3>DeleteObject permissions. For more information about creating an Amazon S3 policy for your resources, see [Specifying Resources in a Policy](#). For an example Amazon S3 policy, see [Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket](#).

If you plan to access Amazon S3 sources and targets that are encrypted with SSE-KMS, attach a policy that allows AWS Glue crawlers, jobs, and development endpoints to decrypt the data. For more information, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\)](#).

The following is an example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject"
      ]
    }
  ]
}
```

```
        "kms:Decrypt"
    ],
    "Resource": [
        "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
    ]
}
}
```

6. For **Role name**, enter a name for your role; for example, **AWSGlueServiceRoleDefault**. Create the role with the name prefixed with the string **AWSGlueServiceRole** to allow the role to be passed from console users to the service. AWS Glue provided policies expect IAM service roles to begin with **AWSGlueServiceRole**. Otherwise, you must add a policy to allow your users the **iam:PassRole** permission for IAM roles to match your naming convention. Choose **Create Role**.

Step 3: Attach a Policy to IAM Users That Access AWS Glue

Any IAM user that signs in to the AWS Glue console or AWS Command Line Interface (AWS CLI) must have permissions to access specific resources. You provide those permissions by using AWS Identity and Access Management (IAM), through policies.

When you finish this step, your IAM user has the following policies attached:

- The AWS managed policy **AWSGlueConsoleFullAccess** or the custom policy **GlueConsoleAccessPolicy**
- **AWSGlueConsoleSageMakerNotebookFullAccess**
- **CloudWatchLogsReadOnlyAccess**
- **AWSCloudFormationReadOnlyAccess**
- **AmazonAthenaFullAccess**

To attach an inline policy and embed it in an IAM user

You can attach an AWS managed policy or an inline policy to an IAM user to access the AWS Glue console. Some of the resources specified in this policy refer to default names that are used by AWS Glue for Amazon S3 buckets, Amazon S3 ETL scripts, CloudWatch Logs, AWS CloudFormation, and Amazon EC2 resources. For simplicity, AWS Glue writes some Amazon S3 objects into buckets in your account prefixed with `aws-glue-*` by default.

Note

You can skip this step if you use the AWS managed policy **AWSGlueConsoleFullAccess**.

Important

AWS Glue needs permission to assume a role that is used to perform work on your behalf. **To accomplish this, you add the `iam:PassRole` permissions to your AWS Glue users.** This policy grants permission to roles that begin with `AWSGlueServiceRole` for AWS Glue service roles, and `AWSGlueServiceNotebookRole` for roles that are required when you create a notebook server. You can also create your own policy for `iam:PassRole` permissions that follows your naming convention.

In this step, you create a policy that is similar to `AWSGlueConsoleFullAccess`. You can find the most current version of `AWSGlueConsoleFullAccess` on the IAM console.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.
3. In the list, choose the name of the user to embed a policy in.

4. Choose the **Permissions** tab and, if necessary, expand the **Permissions policies** section.
5. Choose the **Add Inline policy** link.
6. On the **Create Policy** screen, navigate to a tab to edit JSON. Create a policy document with the following JSON statements, and then choose **Review policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "redshift:DescribeClusters",
                "redshift:DescribeClusterSubnetGroups",
                "iam>ListRoles",
                "iam>ListRolePolicies",
                "iam>GetRole",
                "iam>GetRolePolicy",
                "iam>ListAttachedRolePolicies",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeInstances",
                "rds:DescribeDBInstances",
                "s3>ListAllMyBuckets",
                "s3>ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "cloudformation:DescribeStacks",
                "cloudformation:GetTemplateSummary",
                "dynamodb>ListTables",
                "kms>ListAliases",
                "kms:DescribeKey",
                "cloudwatch:GetMetricData",
                "cloudwatch>ListDashboards"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*/*",
                "arn:aws:s3:::/*aws-glue-*/*",
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

```

},
{
    "Effect": "Allow",
    "Action": [
        "s3:CreateBucket"
    ],
    "Resource": [
        "arn:aws:s3:::aws-glue-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:GetLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:*:*/aws-glue/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation>CreateStack",
        "cloudformation>DeleteStack"
    ],
    "Resource": "arn:aws:cloudformation:*:stack/aws-glue/*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:RunInstances"
    ],
    "Resource": [
        "arn:aws:ec2:*:instance/*",
        "arn:aws:ec2:*:key-pair/*",
        "arn:aws:ec2:*:image/*",
        "arn:aws:ec2:*:security-group/*",
        "arn:aws:ec2:*:network-interface/*",
        "arn:aws:ec2:*:subnet/*",
        "arn:aws:ec2:*:volume/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:TerminateInstances",
        "ec2:CreateTags",
        "ec2:DeleteTags"
    ],
    "Resource": [
        "arn:aws:ec2:*:instance/*"
    ],
    "Condition": {
        "StringLike": {
            "ec2:ResourceTag/aws:cloudformation:stack-id": "arn:aws:cloudformation:*:stack/aws-glue-*"
        },
        "StringEquals": {
            "ec2:ResourceTag/aws:cloudformation:logical-id": "ZeppelinInstance"
        }
    }
},
{
    "Action": [
        "iam:PassRole"
    ],

```

```
"Effect": "Allow",
"Resource": "arn:aws:iam::*:role/AWSGlueServiceRole*",
"Condition": {
    "StringLike": {
        "iam:PassedToService": [
            "glue.amazonaws.com"
        ]
    }
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/AWSGlueServiceNotebookRole*",
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "ec2.amazonaws.com"
            ]
        }
    }
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "glue.amazonaws.com"
            ]
        }
    }
}
]
```

The following table describes the permissions granted by this policy.

Action	Resource	Description
"glue:/*"	"*"	<p>Grants permission to run all AWS Glue API operations.</p> <p>If you had previously created your policy without the "glue:/*" action, you must add the following individual permissions to your policy:</p> <ul style="list-style-type: none"> • "glue>ListCrawlers" • "glue>BatchGetCrawlers" • "glue>ListTriggers" • "glue>BatchGetTriggers" • "glue>ListDevEndpoints" • "glue>BatchGetDevEndpoints" • "glue>ListJobs" • "glue>BatchGetJobs"
"redshift:DescribeClusters", "redshift:DescribeClusterSubnetGroups"	"*"	Allows creation of connections to Amazon Redshift.
"iam>ListRoles", "iam>ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy", "iam>ListAttachedRolePolicies"	"*"	Allows listing IAM roles when working with crawlers, jobs, development endpoints, and notebook servers.
"ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcs", "ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:DescribeVpcAttribute", "ec2:DescribeKeyPairs", "ec2:DescribeInstances"	"*"	Allows setup of Amazon EC2 network items, such as VPCs, when running jobs, crawlers, and development endpoints.
"rds:DescribeDBInstances"	"*"	Allows creation of connections to Amazon RDS.
"s3>ListAllMyBuckets", "s3>ListBucket", "s3:GetBucketAcl", "s3:GetBucketLocation"	"*"	Allows listing of Amazon S3 buckets when working with crawlers, jobs, development endpoints, and notebook servers.
"dynamodb>ListTables"	"*"	Allows listing of DynamoDB tables.
"kms>ListAliases", "kms>DescribeKey"	"*"	Allows working with KMS keys.

Action	Resource	Description
"cloudwatch:GetMetricData", "cloudwatch>ListDashboards"	"*"	Allows working with CloudWatch metrics.
"s3:GetObject", "s3:PutObject"	"arn:aws:s3::: aws-glue-*/*", "arn:aws:s3::: */*aws-glue-*/*", "arn:aws:s3::: aws-glue-*"	Allows get and put of Amazon S3 objects into your account when storing objects such as ETL scripts and notebook server locations. Naming convention: Grants permission to Amazon S3 buckets or folders whose names are prefixed with aws-glue- .
"tag:GetResources"	"*"	Allows retrieval of AWS tags.
"s3>CreateBucket"	"arn:aws:s3::: aws-glue-*"	Allows creation of an Amazon S3 bucket into your account when storing objects such as ETL scripts and notebook server locations. Naming convention: Grants permission to Amazon S3 buckets whose names are prefixed with aws-glue- .
"logs:GetLogEvents"	"arn:aws:logs:*::*: /aws-glue/*"	Allows retrieval of CloudWatch Logs. Naming convention: AWS Glue writes logs to log groups whose names begin with aws-glue- .
"cloudformation>CreateStack", "cloudformation>DeleteStack"	"arn:aws:cloudformation:*::*:stack/aws-glue*/*"	Allows managing AWS CloudFormation stacks when working with notebook servers. Naming convention: AWS Glue creates stacks whose names begin with aws-glue- .
"ec2:RunInstances"	"arn:aws:ec2:::*:instance/*", "arn:aws:ec2:::*:key-pair/*", "arn:aws:ec2:::*:image/*", "arn:aws:ec2:::*:security-group/*", "arn:aws:ec2:::*:network-interface/*", "arn:aws:ec2:::*:subnet/*", "arn:aws:ec2:::*:volume/*"	Allows running of development endpoints and notebook servers.

Action	Resource	Description
"ec2:TerminateInstances", "ec2>CreateTags", "ec2>DeleteTags"	"arn:aws:ec2::*:instance/*"	Allows manipulating development endpoints and notebook servers. Naming convention: AWS Glue AWS CloudFormation stacks with a name that is prefixed with aws-glue- and logical-id ZeppelinInstance .
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceRole*"	Allows AWS Glue to assume PassRole permission for roles that begin with AWSGlueServiceRole .
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceNotebookRole*"	Allows Amazon EC2 to assume PassRole permission for roles that begin with AWSGlueServiceNotebookRole .
"iam:PassRole"	"arn:aws:iam::*:role/ service-role/ AWSGlueServiceRole*"	Allows AWS Glue to assume PassRole permission for roles that begin with service-role/AWSGlueServiceRole .

- On the **Review policy** screen, enter a name for the policy, for example **GlueConsoleAccessPolicy**. When you're satisfied with the policy, choose **Create policy**. Ensure that no errors appear in a red box at the top of the screen. Correct any that are reported.

Note

If **Use autoformatting** is selected, the policy is reformatted whenever you open a policy or choose **Validate Policy**.

To attach the **AWSGlueConsoleFullAccess** managed policy

You can attach the **AWSGlueConsoleFullAccess** policy to provide permissions that are required by the AWS Glue console user.

Note

You can skip this step if you created your own policy for AWS Glue console access.

- Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
- In the navigation pane, choose **Policies**.
- In the list of policies, select the check box next to the **AWSGlueConsoleFullAccess**. You can use the **Filter** menu and the search box to filter the list of policies.
- Choose **Policy actions**, and then choose **Attach**.
- Choose the user to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user to attach the policy to, choose **Attach policy**.

To attach the **AWSGlueConsoleSageMakerNotebookFullAccess** managed policy

You can attach the **AWSGlueConsoleSageMakerNotebookFullAccess** policy to a user to manage Amazon SageMaker notebooks created on the AWS Glue console. In addition to other required AWS Glue console permissions, this policy grants access to resources needed to manage Amazon SageMaker notebooks.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the check box next to the **AWSGlueConsoleSageMakerNotebookFullAccess**. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.
5. Choose the user to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user to attach the policy to, choose **Attach policy**.

To attach the **CloudWatchLogsReadOnlyAccess** managed policy

You can attach the **CloudWatchLogsReadOnlyAccess** policy to a user to view the logs created by AWS Glue on the CloudWatch Logs console.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the check box next to the **CloudWatchLogsReadOnlyAccess**. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.
5. Choose the user to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user to attach the policy to, choose **Attach policy**.

To attach the **AWSCloudFormationReadOnlyAccess** managed policy

You can attach the **AWSCloudFormationReadOnlyAccess** policy to a user to view the AWS CloudFormation stacks used by AWS Glue on the AWS CloudFormation console.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the check box next to the **AWSCloudFormationReadOnlyAccess**. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.
5. Choose the user to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user to attach the policy to, choose **Attach policy**.

To attach the **AmazonAthenaFullAccess** managed policy

You can attach the **AmazonAthenaFullAccess** policy to a user to view Amazon S3 data in the Athena console.

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

2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the check box next to the **AmazonAthenaFullAccess**. You can use the **Filter** menu and the search box to filter the list of policies.
4. Choose **Policy actions**, and then choose **Attach**.
5. Choose the user to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities. After choosing the user to attach the policy to, choose **Attach policy**.

Step 4: Create an IAM Policy for Notebook Servers

If you plan to use notebooks with development endpoints, you must specify permissions when you create the notebook server. You provide those permissions by using AWS Identity and Access Management (IAM).

This policy grants permission for some Amazon S3 actions to manage resources in your account that are needed by AWS Glue when it assumes the role using this policy. Some of the resources that are specified in this policy refer to default names used by AWS Glue for Amazon S3 buckets, Amazon S3 ETL scripts, and Amazon EC2 resources. For simplicity, AWS Glue defaults writing some Amazon S3 objects into buckets in your account prefixed with aws-glue-*.

Note

You can skip this step if you use the AWS managed policy **AWSGlueServiceNotebookRole**.

In this step, you create a policy that is similar to **AWSGlueServiceNotebookRole**. You can find the most current version of **AWSGlueServiceNotebookRole** on the IAM console.

To create an IAM policy for notebooks

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. On the **Create Policy** screen, navigate to a tab to edit JSON. Create a policy document with the following JSON statements, and then choose **Review policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:CreatePartition",
                "glue: CreateTable",
                "glue:DeleteDatabase",
                "glue:DeletePartition",
                "glue:DeleteTable",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:GetTable",
                "glue:GetTableVersions",
                "glue:GetTables",
                "glue:UpdateDatabase",
                "glue:UpdatePartition",
                "glue:UpdateTable",
                "glue>CreateBookmark",
                "glue:GetBookmark",
                "glue:PutBookmark"
            ]
        }
    ]
}
```

```
        "glue:UpdateBookmark",
        "glue:GetMetric",
        "glue:PutMetric",
        "glue>CreateConnection",
        "glue>CreateJob",
        "glue>DeleteConnection",
        "glue>DeleteJob",
        "glue:GetConnection",
        "glue:GetConnections",
        "glue:GetDevEndpoint",
        "glue:GetDevEndpoints",
        "glue:GetJob",
        "glue:GetJobs",
        "glue:UpdateJob",
        "glue:BatchDeleteConnection",
        "glue:UpdateConnection",
        "glue:GetUserDefinedFunction",
        "glue:UpdateUserDefinedFunction",
        "glue GetUserDefinedFunctions",
        "glue:DeleteUserDefinedFunction",
        "glue:CreateUserDefinedFunction",
        "glue:BatchGetPartition",
        "glue:BatchDeletePartition",
        "glue:BatchCreatePartition",
        "glue:BatchDeleteTable",
        "glue:UpdateDevEndpoint",
        "s3:GetBucketLocation",
        "s3>ListBucket",
        "s3>ListAllMyBuckets",
        "s3:GetBucketAcl"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3.GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::crawler-public*",
        "arn:aws:s3:::aws-glue*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:DeleteObject"
    ],
    "Resource": [
        "arn:aws:s3:::aws-glue*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateTags",
        "ec2>DeleteTags"
    ],
    "Condition": {
        "ForAllValues:StringEquals": {
            "aws:TagKeys": [
                "aws-glue-service-resource"
            ]
        }
    }
}
```

```

        },
        "Resource": [
            "arn:aws:ec2::::network-interface/*",
            "arn:aws:ec2::::security-group/*",
            "arn:aws:ec2::::instance/*"
        ]
    }
}

```

The following table describes the permissions granted by this policy.

Action	Resource	Description
"glue:*	"*"	Grants permission to run all AWS Glue API operations.
"s3:GetBucketLocation", "s3>ListBucket", "s3>ListAllMyBuckets", "s3:GetBucketAcl"	"*"	Allows listing of Amazon S3 buckets from notebook servers.
"s3GetObject"	"arn:aws:s3::::crawler-public*", "arn:aws:s3::::aws-glue-*"	Allows get of Amazon S3 objects used by examples and tutorials from notebooks. Naming convention: Amazon S3 bucket names begin with crawler-public and aws-glue- .
"s3PutObject", "s3DeleteObject"	"arn:aws:s3::::aws-glue*"	Allows put and delete of Amazon S3 objects into your account from notebooks. Naming convention: Uses Amazon S3 folders named aws-glue .
"ec2CreateTags", "ec2DeleteTags"	"arn:aws:ec2::::network-interface/*", "arn:aws:ec2::::security-group/*", "arn:aws:ec2::::instance/*"	Allows tagging of Amazon EC2 resources created for notebook servers. Naming convention: AWS Glue tags Amazon EC2 instances with aws-glue-service-resource .

- On the **Review Policy** screen, enter your **Policy Name**, for example **GlueServiceNotebookPolicyDefault**. Enter an optional description, and when you're satisfied with the policy, choose **Create policy**.

Step 5: Create an IAM Role for Notebook Servers

If you plan to use notebooks with development endpoints, you need to grant the IAM role permissions. You provide those permissions by using AWS Identity and Access Management IAM, through an IAM role.

Note

When you create an IAM role using the IAM console, the console creates an instance profile automatically and gives it the same name as the role to which it corresponds.

To create an IAM role for notebooks

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For role type, choose **AWS Service**, find and choose **EC2**, and choose the **EC2** use case, then choose **Next: Permissions**.
5. On the **Attach permissions policy** page, choose the policies that contain the required permissions; for example, **AWSGlueServiceNotebookRole** for general AWS Glue permissions and the AWS managed policy **AmazonS3FullAccess** for access to Amazon S3 resources. Then choose **Next: Review**.

Note

Ensure that one of the policies in this role grants permissions to your Amazon S3 sources and targets. Also confirm that your policy allows full access to the location where you store your notebook when you create a notebook server. You might want to provide your own policy for access to specific Amazon S3 resources. For more information about creating an Amazon S3 policy for your resources, see [Specifying Resources in a Policy](#).

If you plan to access Amazon S3 sources and targets that are encrypted with SSE-KMS, attach a policy that allows notebooks to decrypt the data. For more information, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\)](#).

The following is an example.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:Decrypt"  
            ],  
            "Resource": [  
                "arn:aws:kms:*:account-id-without-hyphens:key/key-id"  
            ]  
        }  
    ]  
}
```

6. For **Role name**, enter a name for your role. Create the role with the name prefixed with the string **AWSGlueServiceNotebookRole** to allow the role to be passed from console users to the notebook server. AWS Glue provided policies expect IAM service roles to begin with **AWSGlueServiceNotebookRole**. Otherwise you must add a policy to your users to allow the **iam:PassRole** permission for IAM roles to match your naming convention. For example, enter **AWSGlueServiceNotebookRoleDefault**. Then choose **Create role**.

Step 6: Create an IAM Policy for Amazon SageMaker Notebooks

If you plan to use Amazon SageMaker notebooks with development endpoints, you must specify permissions when you create the notebook. You provide those permissions by using AWS Identity and Access Management (IAM).

To create an IAM policy for Amazon SageMaker notebooks

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. On the **Create Policy** page, navigate to a tab to edit the JSON. Create a policy document with the following JSON statements. Edit `bucket-name`, `region-code`, and `account-id` for your environment.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "s3>ListBucket"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::bucket-name"  
            ]  
        },  
        {  
            "Action": [  
                "s3GetObject"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::bucket-name*"  
            ]  
        },  
        {  
            "Action": [  
                "logs>CreateLogStream",  
                "logs>DescribeLogStreams",  
                "logs>PutLogEvents",  
                "logs>CreateLogGroup"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/*",  
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/*:log-  
stream:aws-glue-*"  
            ]  
        },  
        {  
            "Action": [  
                "glue>UpdateDevEndpoint",  
                "glue>GetDevEndpoint",  
                "glue>GetDevEndpoints"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:glue:region-code:account-id:devEndpoint/*"  
            ]  
        },  
        {  
            "Action": [  
                "sagemaker>ListTags"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:sagemaker:region-code:account-id:notebook-instance/*"  
            ]  
        }  
    ]  
}
```

```

        ]
    }
}
```

Then choose **Review policy**.

The following table describes the permissions granted by this policy.

Action	Resource	Description
"s3>ListBucket*"	"arn:aws:s3:::: <i>bucket-name</i> "	Grants permission to list Amazon S3 buckets.
"s3GetObject"	"arn:aws:s3:::: <i>bucket-name</i> *"	Grants permission to get Amazon S3 objects that are used by Amazon SageMaker notebooks.
"logs>CreateLogStream", "logs:DescribeLogStreams", "logs:PutLogEvents", "logs>CreateLogGroup"	"arn:aws:logs: <i>region-code:account-id</i> :log-group:/aws/sagemaker/*", "arn:aws:logs: <i>region-code:account-id</i> :log-group:/aws/sagemaker/*:log-stream:aws-glue-*"	Grants permission to write logs to Amazon CloudWatch Logs from notebooks. Naming convention: Writes to log groups whose names begin with aws-glue .
"glue:UpdateDevEndpoint", "glue:GetDevEndpoint", "glue:GetEndpoints"	"arn:aws:glue: <i>region-code:account-id</i> :devEndpoint/*"	Grants permission to use a development endpoint from Amazon SageMaker notebooks.
"sagemaker>ListTags"	"arn:aws:sagemaker: <i>region-code:account-id</i> :notebook-instance/*"	Grants permission to return tags for an Amazon SageMaker resource. The aws-glue-dev-endpoint tag is required on the Amazon SageMaker notebook for connecting the notebook to a development endpoint.

5. On the **Review Policy** screen, enter your **Policy Name**, for example **AWSGlueSageMakerNotebook**. Enter an optional description, and when you're satisfied with the policy, choose **Create policy**.

Step 7: Create an IAM Role for Amazon SageMaker Notebooks

If you plan to use Amazon SageMaker notebooks with development endpoints, you need to grant the IAM role permissions. You provide those permissions by using AWS Identity and Access Management (IAM), through an IAM role.

To create an IAM role for Amazon SageMaker notebooks

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Choose **Create role**.

4. For role type, choose **AWS Service**, find and choose **SageMaker**, and then choose the **SageMaker - Execution** use case. Then choose **Next: Permissions**.
5. On the **Attach permissions policy** page, choose the policies that contain the required permissions; for example, **AmazonSageMakerFullAccess**. Choose **Next: Review**.

If you plan to access Amazon S3 sources and targets that are encrypted with SSE-KMS, attach a policy that allows notebooks to decrypt the data, as shown in the following example. For more information, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
      ]
    }
  ]
}
```

6. For **Role name**, enter a name for your role. To allow the role to be passed from console users to Amazon SageMaker, use a name that is prefixed with the string **AWSGlueServiceSageMakerNotebookRole**. AWS Glue provided policies expect IAM roles to begin with **AWSGlueServiceSageMakerNotebookRole**. Otherwise you must add a policy to your users to allow the **iam:PassRole** permission for IAM roles to match your naming convention.

For example, enter **AWSGlueServiceSageMakerNotebookRole-Default**, and then choose **Create role**.

7. After you create the role, attach the policy that allows additional permissions required to create Amazon SageMaker notebooks from AWS Glue.

Open the role that you just created, **AWSGlueServiceSageMakerNotebookRole-Default**, and choose **Attach policies**. Attach the policy that you created named **AWSGlueSageMakerNotebook** to the role.

Setting Up DNS in Your VPC

Domain Name System (DNS) is a standard by which names used on the internet are resolved to their corresponding IP addresses. A DNS hostname uniquely names a computer and consists of a host name and a domain name. DNS servers resolve DNS hostnames to their corresponding IP addresses.

To set up DNS in your VPC, ensure that DNS hostnames and DNS resolution are both enabled in your VPC. The VPC network attributes `enableDnsHostnames` and `enableDnsSupport` must be set to `true`. To view and modify these attributes, go to the VPC console at <https://console.aws.amazon.com/vpc/>.

For more information, see [Using DNS with your VPC](#).

Note

If you are using Route 53, confirm that your configuration does not override DNS network attributes.

Setting Up Your Environment to Access Data Stores

To run your extract, transform, and load (ETL) jobs, AWS Glue must be able to access your data stores. If a job doesn't need to run in your virtual private cloud (VPC) subnet—for example, transforming data from Amazon S3 to Amazon S3—no additional configuration is needed.

If a job needs to run in your VPC subnet—for example, transforming data from a JDBC data store in a private subnet—AWS Glue sets up [elastic network interfaces](#) that enable your jobs to connect securely to other resources within your VPC. Each elastic network interface is assigned a private IP address from the IP address range within the subnet you specified. No public IP addresses are assigned. Security groups specified in the AWS Glue connection are applied on each of the elastic network interfaces. For more information, see [Setting Up a VPC to Connect to JDBC Data Stores \(p. 32\)](#).

All JDBC data stores that are accessed by the job must be available from the VPC subnet. To access Amazon S3 from within your VPC, a [VPC endpoint \(p. 30\)](#) is required. If your job needs to access both VPC resources and the public internet, the VPC needs to have a Network Address Translation (NAT) gateway inside the VPC.

A job or development endpoint can only access one VPC (and subnet) at a time. If you need to access data stores in different VPCs, you have the following options:

- Use VPC peering to access the data stores. For more about VPC peering, see [VPC Peering Basics](#)
- Use an Amazon S3 bucket as an intermediary storage location. Split the work into two jobs, with the Amazon S3 output of job 1 as the input to job 2.

For JDBC data stores, you create a connection in AWS Glue with the necessary properties to connect to your data stores. For more information about the connection, see [Adding a Connection to Your Data Store \(p. 110\)](#).

Note

Make sure you set up your DNS environment for AWS Glue. For more information, see [Setting Up DNS in Your VPC \(p. 29\)](#).

Topics

- [Amazon VPC Endpoints for Amazon S3 \(p. 30\)](#)
- [Setting Up a VPC to Connect to JDBC Data Stores \(p. 32\)](#)

Amazon VPC Endpoints for Amazon S3

For security reasons, many AWS customers run their applications within an Amazon Virtual Private Cloud environment (Amazon VPC). With Amazon VPC, you can launch Amazon EC2 instances into a virtual private cloud, which is logically isolated from other networks—including the public internet. With an Amazon VPC, you have control over its IP address range, subnets, routing tables, network gateways, and security settings.

Note

If you created your AWS account after 2013-12-04, you already have a default VPC in each AWS Region. You can immediately start using your default VPC without any additional configuration. For more information, see [Your Default VPC and Subnets](#) in the Amazon VPC User Guide.

Many customers have legitimate privacy and security concerns about sending and receiving data across the public internet. Customers can address these concerns by using a virtual private network (VPN) to

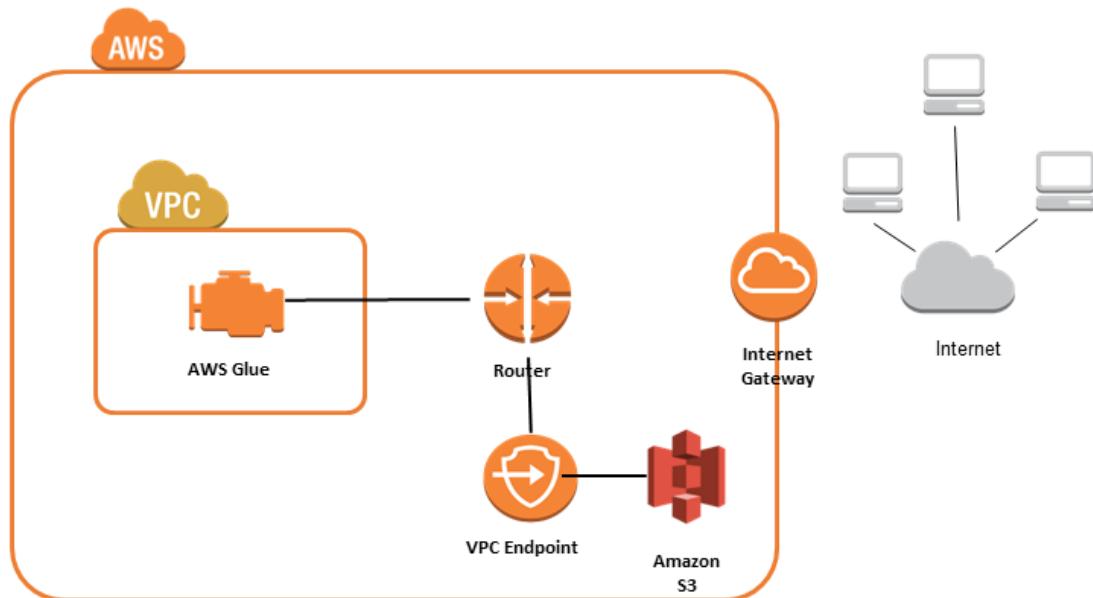
route all Amazon S3 network traffic through their own corporate network infrastructure. However, this approach can introduce bandwidth and availability challenges.

VPC endpoints for Amazon S3 can alleviate these challenges. A VPC endpoint for Amazon S3 enables AWS Glue to use private IP addresses to access Amazon S3 with no exposure to the public internet. AWS Glue does not require public IP addresses, and you don't need an internet gateway, a NAT device, or a virtual private gateway in your VPC. You use endpoint policies to control access to Amazon S3. Traffic between your VPC and the AWS service does not leave the Amazon network.

When you create a VPC endpoint for Amazon S3, any requests to an Amazon S3 endpoint within the Region (for example, `s3.us-west-2.amazonaws.com`) are routed to a private Amazon S3 endpoint within the Amazon network. You don't need to modify your applications running on EC2 instances in your VPC—the endpoint name remains the same, but the route to Amazon S3 stays entirely within the Amazon network, and does not access the public internet.

For more information about VPC endpoints, see [VPC Endpoints](#) in the Amazon VPC User Guide.

The following diagram shows how AWS Glue can use a VPC endpoint to access Amazon S3.



To set up access for Amazon S3

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the left navigation pane, choose **Endpoints**.
3. Choose **Create Endpoint**, and follow the steps to create an Amazon S3 endpoint in your VPC.

Setting Up a VPC to Connect to JDBC Data Stores

To enable AWS Glue components to communicate, you must set up access to your data stores, such as Amazon Redshift and Amazon RDS. To enable AWS Glue to communicate between its components, specify a security group with a self-referencing inbound rule for all TCP ports. By creating a self-referencing rule, you can restrict the source to the same security group in the VPC, and it's not open to all networks. The default security group for your VPC might already have a self-referencing inbound rule for ALL Traffic.

To set up access for Amazon Redshift data stores

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. In the left navigation pane, choose **Clusters**.
3. Choose the cluster name that you want to access from AWS Glue.
4. In the **Cluster Properties** section, choose a security group in **VPC security groups** to allow AWS Glue to use. Record the name of the security group that you chose for future reference. Choosing the security group opens the Amazon EC2 console **Security Groups** list.
5. Choose the security group to modify and navigate to the **Inbound** tab.
6. Add a self-referencing rule to allow AWS Glue components to communicate. Specifically, add or confirm that there is a rule of **Type All TCP**, **Protocol** is **TCP**, **Port Range** includes all ports, and whose **Source** is the same security group name as the **Group ID**.

The inbound rule looks similar to the following:

Type	Protocol	Port Range	Source
All TCP	TCP	0–65535	database-security-group

For example:

The screenshot shows the AWS EC2 Security Groups Inbound Rules tab. A red circle highlights the 'Source' field in the first row of the table, which contains the value 'sg-ba764ac6'. Below the table, an 'Edit' button is visible. Another red circle highlights the 'Source' field in the 'Edit' modal, which also contains the value 'sg-ba764ac6'.

Type	Protocol	Port Range	Source
ALL TCP	TCP (6)	ALL	sg-ba764ac6

7. Add a rule for outbound traffic also. Either open outbound traffic to all ports, for example:

Type	Protocol	Port Range	Destination
All Traffic	ALL	ALL	0.0.0.0/0

Or create a self-referencing rule where **Type** All TCP, **Protocol** is TCP, **Port Range** includes all ports, and whose **Destination** is the same security group name as the **Group ID**. If using an Amazon S3 VPC endpoint, also add an HTTPS rule for Amazon S3 access. The *s3-prefix-list-id* is required in the security group rule to allow traffic from the VPC to the Amazon S3 VPC endpoint.

For example:

Type	Protocol	Port Range	Destination
All TCP	TCP	0–65535	<i>security-group</i>
HTTPS	TCP	443	<i>s3-prefix-list-id</i>

To set up access for Amazon RDS data stores

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the left navigation pane, choose **Instances**.
3. Choose the Amazon RDS **Engine** and **DB Instance** name that you want to access from AWS Glue.
4. From **Instance Actions**, choose **See Details**. On the **Details** tab, find the **Security Groups** name you will access from AWS Glue. Record the name of the security group for future reference.
5. Choose the security group to open the Amazon EC2 console.
6. Confirm that your **Group ID** from Amazon RDS is chosen, then choose the **Inbound** tab.
7. Add a self-referencing rule to allow AWS Glue components to communicate. Specifically, add or confirm that there is a rule of **Type** All TCP, **Protocol** is TCP, **Port Range** includes all ports, and whose **Source** is the same security group name as the **Group ID**.

The inbound rule looks similar to this:

Type	Protocol	Port Range	Source
All TCP	TCP	0–65535	<i>database-security-group</i>

For example:

Type	Protocol	Port Range	Source
ALL TCP	TCP (6)	ALL	sg-ba764ac6

8. Add a rule for outbound traffic also. Either open outbound traffic to all ports, for example:

Type	Protocol	Port Range	Destination
All Traffic	ALL	ALL	0.0.0.0/0

Or create a self-referencing rule where **Type** All TCP, **Protocol** is TCP, **Port Range** includes all ports, and whose **Destination** is the same security group name as the **Group ID**. If using an Amazon S3 VPC endpoint, also add an HTTPS rule for Amazon S3 access. The *s3-prefix-list-id* is required in the security group rule to allow traffic from the VPC to the Amazon S3 VPC endpoint.

For example:

Type	Protocol	Port Range	Destination
All TCP	TCP	0-65535	<i>security-group</i>
HTTPS	TCP	443	<i>s3-prefix-list-id</i>

Setting Up Your Environment for Development Endpoints

To run your extract, transform, and load (ETL) scripts with AWS Glue, you sometimes develop and test your scripts using a development endpoint. When you set up a development endpoint, you specify a virtual private cloud (VPC), subnet, and security groups.

Note

Make sure you set up your DNS environment for AWS Glue. For more information, see [Setting Up DNS in Your VPC \(p. 29\)](#).

Setting Up Your Network for a Development Endpoint

To enable AWS Glue to access required resources, add a row in your subnet route table to associate a prefix list for Amazon S3 to the VPC endpoint. A prefix list ID is required for creating an outbound security group rule that allows traffic from a VPC to access an AWS service through a VPC endpoint. To ease connecting to a notebook server that is associated with this development endpoint, from your local machine, add a row to the route table to add an internet gateway ID. For more information, see [VPC Endpoints](#). Update the subnet routes table to be similar to the following table:

Destination	Target		
10.0.0.0/16	local		
pl-id for Amazon S3	vpce-id		
0.0.0.0/0	igw-xxxx		

To enable AWS Glue to communicate between its components, specify a security group with a self-referencing inbound rule for all TCP ports. By creating a self-referencing rule, you can restrict the source

to the same security group in the VPC, and it's not open to all networks. The default security group for your VPC might already have a self-referencing inbound rule for ALL Traffic.

To set up a security group

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the left navigation pane, choose **Security Groups**.
3. Either choose an existing security group from the list, or **Create Security Group** to use with the development endpoint.
4. In the security group pane, navigate to the **Inbound** tab.
5. Add a self-referencing rule to allow AWS Glue components to communicate. Specifically, add or confirm that there is a rule of **Type All TCP**, **Protocol** is **TCP**, **Port Range** includes all ports, and whose **Source** is the same security group name as the **Group ID**.

The inbound rule looks similar to this:

Type	Protocol	Port Range	Source
All TCP	TCP	0-65535	<i>security-group</i>

The following shows an example of a self-referencing inbound rule:

The screenshot shows the AWS Management Console interface for managing security groups. At the top, a security group named 'sg-ba764ac6' is selected, indicated by a red oval around its name. Below the navigation tabs (Summary, Inbound Rules, Outbound Rules, Tags), there is an 'Edit' button. Under the 'Inbound Rules' tab, a table lists a single rule. The table has columns: Type, Protocol, Port Range, and Source. The 'Source' column contains the value 'sg-ba764ac6', which is also highlighted with a red oval.

Type	Protocol	Port Range	Source
ALL TCP	TCP (6)	ALL	sg-ba764ac6

6. Add a rule to for outbound traffic also. Either open outbound traffic to all ports, or create a self-referencing rule of **Type All TCP**, **Protocol** is **TCP**, **Port Range** includes all ports, and whose **Source** is the same security group name as the **Group ID**.

The outbound rule looks similar to one of these rules:

Type	Protocol	Port Range	Destination
All TCP	TCP	0-65535	<i>security-group</i>
All Traffic	ALL	ALL	0.0.0.0/0

Setting Up Amazon EC2 for a Notebook Server

With a development endpoint, you can create a notebook server to test your ETL scripts with Zeppelin notebooks. To enable communication to your notebook, specify a security group with inbound rules for both HTTPS (port 443) and SSH (port 22). Ensure that the rule's source is either 0.0.0.0/0 or the IP address of the machine that is connecting to the notebook.

To set up a security group

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the left navigation pane, choose **Security Groups**.
3. Either choose an existing security group from the list, or **Create Security Group** to use with your notebook server. The security group that is associated with your development endpoint is also used to create your notebook server.
4. In the security group pane, navigate to the **Inbound** tab.
5. Add inbound rules similar to this:

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

The following shows an example of the inbound rules for the security group:

The screenshot shows the AWS Management Console interface for managing security groups. The title bar says "Security Group: sg-19e1b768". Below it, there are tabs: "Description", "Inbound" (which is highlighted in orange), "Outbound", and "Tags". Under the "Inbound" tab, there is an "Edit" button. The main area displays a table of inbound rules:

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

Setting Up Encryption in AWS Glue

The following example workflow highlights the options to configure when you use encryption with AWS Glue. The example demonstrates the use of specific AWS Key Management Service (AWS KMS) keys, but you might choose other settings based on your particular needs. This workflow highlights only the options that pertain to encryption when setting up AWS Glue.

1. If the user of the AWS Glue console doesn't use a permissions policy that allows all AWS Glue API operations (for example, "glue:*"), confirm that the following actions are allowed:
 - "glue:GetDataCatalogEncryptionSettings"
 - "glue:PutDataCatalogEncryptionSettings"
 - "glue>CreateSecurityConfiguration"

- "glue:GetSecurityConfiguration"
- "glue:GetSecurityConfigurations"
- "glue>DeleteSecurityConfiguration"

2. Any client that accesses or writes to an encrypted catalog—that is, any console user, crawler, job, or development endpoint—needs the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt",
      "kms:Encrypt"
    ],
    "Resource": "<key-arns-used-for-data-catalog>"
  }
}
```

3. Any user or role that accesses an encrypted connection password needs the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "<key-arns-used-for-password-encryption>"
  }
}
```

4. The role of any extract, transform, and load (ETL) job that writes encrypted data to Amazon S3 needs the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:Encrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "<key-arns-used-for-s3>"
  }
}
```

5. Any ETL job or crawler that writes encrypted Amazon CloudWatch Logs requires the following permissions in the key policy (not the IAM policy).

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.region.amazonaws.com"
  },
  "Action": [
    "kms:Encrypt*",
    "kms:Decrypt*",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ]
}
```

```

    "kms:Describe"
],
"Resource": "<arn of key used for ETL/crawler cloudwatch encryption>"
}

```

For more information about key policies, see [Using Key Policies in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

6. Any ETL job that uses an encrypted job bookmark needs the following permissions.

```

{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt",
            "kms:Encrypt"
        ],
        "Resource": "<key-arns-used-for-job-bookmark-encryption>"
    }
]
}

```

7. On the AWS Glue console, choose **Settings** in the navigation pane.
 - a. On the **Data catalog settings** page, encrypt your Data Catalog by selecting **Metadata encryption**. This option encrypts all the objects in the Data Catalog with the AWS KMS key that you choose.
 - b. For **AWS KMS key**, choose **aws/glue**. You can also choose a customer master key (CMK) that you created.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key** list displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

When encryption is enabled, the client that is accessing the Data Catalog must have AWS KMS permissions.

8. In the navigation pane, choose **Security configurations**. A security configuration is a set of security properties that can be used to configure AWS Glue processes. Then choose **Add security configuration**. In the configuration, choose any of the following options:
 - a. Select **S3 encryption**. For **Encryption mode**, choose **SSE-KMS**. For the **AWS KMS key**, choose **aws/s3** (ensure that the user has permission to use this key). This enables data written by the job to Amazon S3 to use the AWS managed AWS KMS key.
 - b. Select **CloudWatch logs encryption**, and choose a CMK. (Ensure that the user has permission to use this key). For more information, see [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key** list displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

- c. Choose **Advanced properties**, and select **Job bookmark encryption**. For the **AWS KMS key**, choose **aws/glue** (ensure that the user has permission to use this key). This enables encryption of job bookmarks written to Amazon S3 with the AWS Glue AWS KMS key.

9. In the navigation pane, choose **Connections**.

- a. Choose **Add connection** to create a connection to the Java Database Connectivity (JDBC) data store that is the target of your ETL job.
- b. To enforce that Secure Sockets Layer (SSL) encryption is used, select **Require SSL connection**, and test your connection.

10In the navigation pane, choose **Jobs**.

- a. Choose **Add job** to create a job that transforms data.
- b. In the job definition, choose the security configuration that you created.

11On the AWS Glue console, run your job on demand. Verify that any Amazon S3 data written by the job, the CloudWatch Logs written by the job, and the job bookmarks are all encrypted.

AWS Glue Console Workflow Overview

With AWS Glue, you store metadata in the AWS Glue Data Catalog. You use this metadata to orchestrate ETL jobs that transform data sources and load your data warehouse. The following steps describe the general workflow and some of the choices that you make when working with AWS Glue.

1. Populate the AWS Glue Data Catalog with table definitions.

In the console, you can add a crawler to populate the AWS Glue Data Catalog. You can start the **Add crawler** wizard from the list of tables or the list of crawlers. You choose one or more data stores for your crawler to access. You can also create a schedule to determine the frequency of running your crawler.

Optionally, you can provide a custom classifier that infers the schema of your data. You can create custom classifiers using a grok pattern. However, AWS Glue provides built-in classifiers that are automatically used by crawlers if a custom classifier does not recognize your data. When you define a crawler, you don't have to select a classifier. For more information about classifiers in AWS Glue, see [Adding Classifiers to a Crawler \(p. 129\)](#).

Crawling some types of data stores requires a connection that provides authentication and location information. If needed, you can create a connection that provides this required information in the AWS Glue console.

The crawler reads your data store and creates data definitions and named tables in the AWS Glue Data Catalog. These tables are organized into a database of your choosing. You can also populate the Data Catalog with manually created tables. With this method, you provide the schema and other metadata to create table definitions in the Data Catalog. Because this method can be a bit tedious and error prone, it's often better to have a crawler create the table definitions.

For more information about populating the AWS Glue Data Catalog with table definitions, see [Defining Tables in the AWS Glue Data Catalog \(p. 106\)](#).

2. Define a job that describes the transformation of data from source to target.

Generally, to create a job, you have to make the following choices:

- Pick a table from the AWS Glue Data Catalog to be the source of the job. Your job uses this table definition to access your data store and interpret the format of your data.
- Pick a table or location from the AWS Glue Data Catalog to be the target of the job. Your job uses this information to access your data store.
- Tell AWS Glue to generate a PySpark script to transform your source to target. AWS Glue generates the code to call built-in transforms to convert data from its source schema to target schema format. These transforms perform operations such as copy data, rename columns, and filter data to transform data as necessary. You can modify this script in the AWS Glue console.

For more information about defining jobs in AWS Glue, see [Authoring Jobs in AWS Glue \(p. 163\)](#).

3. Run your job to transform your data.

You can run your job on demand, or start it based on one of these trigger types:

- A trigger that is based on a cron schedule.

- A trigger that is event-based; for example, the successful completion of another job can start an AWS Glue job.
- A trigger that starts a job on demand.

For more information about triggers in AWS Glue, see [Triggering Jobs in AWS Glue \(p. 182\)](#).

4. Monitor your scheduled crawlers and triggered jobs.

Use the AWS Glue console to view the following:

- Job run details and errors.
- Crawler run details and errors.
- Any notifications about AWS Glue activities

For more information about monitoring your crawlers and jobs in AWS Glue, see [Running and Monitoring AWS Glue \(p. 220\)](#).

Security in AWS Glue

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to AWS Glue, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Glue. The following topics show you how to configure AWS Glue to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Glue resources.

Topics

- [Data Protection in AWS Glue \(p. 41\)](#)
- [Identity and Access Management in AWS Glue \(p. 50\)](#)
- [Logging and Monitoring in AWS Glue \(p. 101\)](#)
- [Compliance Validation for AWS Glue \(p. 101\)](#)
- [Resilience in AWS Glue \(p. 102\)](#)
- [Infrastructure Security in AWS Glue \(p. 102\)](#)

Data Protection in AWS Glue

AWS Glue offers several features that are designed to help protect your data.

Topics

- [Encryption at Rest \(p. 41\)](#)
- [Encryption in Transit \(p. 47\)](#)
- [Key Management \(p. 47\)](#)
- [Internetwork Traffic Privacy \(p. 48\)](#)
- [AWS Glue Dependency on Other AWS Services \(p. 50\)](#)
- [Development Endpoints \(p. 50\)](#)

Encryption at Rest

AWS Glue supports data encryption at rest for [Authoring Jobs in AWS Glue \(p. 163\)](#) and [Developing Scripts Using Development Endpoints \(p. 184\)](#). You can configure extract, transform, and load (ETL)

jobs and development endpoints to use [AWS Key Management Service \(AWS KMS\)](#) keys to write encrypted data at rest. You can also encrypt the metadata stored in the [AWS Glue Data Catalog \(p. 7\)](#) using keys that you manage with AWS KMS. Additionally, you can use AWS KMS keys to encrypt job bookmarks and the logs generated by [crawlers](#) and ETL jobs.

You can encrypt metadata objects in your AWS Glue Data Catalog in addition to the data written to Amazon Simple Storage Service (Amazon S3) and Amazon CloudWatch Logs by jobs, crawlers, and development endpoints. When you create jobs, crawlers, and development endpoints in AWS Glue, you can provide encryption settings by attaching a security configuration. Security configurations contain Amazon S3-managed server-side encryption keys (SSE-S3) or customer master keys (CMKs) stored in AWS KMS (SSE-KMS). You can create security configurations using the AWS Glue console.

You can also enable encryption of the entire Data Catalog in your account. You do so by specifying CMKs stored in AWS KMS.

Important

AWS Glue supports only symmetric CMKs. For more information, see [Customer Master Keys \(CMKs\) in the AWS Key Management Service Developer Guide](#).

With encryption enabled, when you add Data Catalog objects, run crawlers, run jobs, or start development endpoints, SSE-S3 or SSE-KMS keys are used to write data at rest. In addition, you can configure AWS Glue to only access Java Database Connectivity (JDBC) data stores through a trusted Secure Sockets Layer (SSL) protocol.

In AWS Glue, you control encryption settings in the following places:

- The settings of your Data Catalog.
- The security configurations that you create.
- The server-side encryption setting (SSE-S3 or SSE-KMS) that is passed as a parameter to your AWS Glue ETL (extract, transform, and load) job.

For more information about how to set up encryption, see [Setting Up Encryption in AWS Glue \(p. 36\)](#).

Topics

- [Encrypting Your Data Catalog \(p. 42\)](#)
- [Encrypting Connection Passwords \(p. 43\)](#)
- [Encrypting Data Written by Crawlers, Jobs, and Development Endpoints \(p. 44\)](#)

Encrypting Your Data Catalog

You can enable encryption of your AWS Glue Data Catalog objects in the **Settings** of the Data Catalog on the AWS Glue console. You can enable or disable encryption settings for the entire Data Catalog. In the process, you specify an AWS KMS key that is automatically used when objects, such as tables, are written to the Data Catalog. The encrypted objects include the following:

- Databases
- Tables
- Partitions
- Table versions
- Connections
- User-defined functions

You can set this behavior using the AWS Management Console or AWS Command Line Interface (AWS CLI).

To enable encryption using the console

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Settings** in the navigation pane.
3. On the **Data catalog settings** page, select **Metadata encryption**, and choose an AWS KMS key.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key list** displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

When encryption is enabled, all future Data Catalog objects are encrypted. The default key is the AWS Glue AWS KMS key that is created for your account by AWS. If you clear this setting, objects are no longer encrypted when they are written to the Data Catalog. Any encrypted objects in the Data Catalog can continue to be accessed with the key.

To enable encryption using the SDK or AWS CLI

- Use the `PutDataCatalogEncryptionSettings` API operation. If no key is specified, the default AWS Glue encryption key for the customer account is used.

Important

The AWS KMS key must remain available in the AWS KMS key store for any objects that are encrypted with it in the Data Catalog. If you remove the key, the objects can no longer be decrypted. You might want this in some scenarios to prevent access to Data Catalog metadata.

When encryption is enabled, the client that is accessing the Data Catalog must have the following AWS KMS permissions in its policy:

- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKey`

For example, when you define a crawler or a job, the IAM role that you provide in the definition must have these AWS KMS permissions.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:Decrypt",  
                "kms:Encrypt",  
                "kms:GenerateDataKey"  
            ],  
            "Resource": "ARN-of-key-used-to-encrypt-data-catalog"  
        }  
    ]  
}
```

Encrypting Connection Passwords

You can retrieve connection passwords in the AWS Glue Data Catalog by using the `GetConnection` and `GetConnections` API operations. These passwords are stored in the Data Catalog connection and are

used when AWS Glue connects to a Java Database Connectivity (JDBC) data store. When the connection was created or updated, an option in the Data Catalog settings determined whether the password was encrypted, and if so, what AWS Key Management Service (AWS KMS) key was specified.

On the AWS Glue console, you can enable this option on the **Data catalog settings** page.

To encrypt connection passwords

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Settings** in the navigation pane.
3. On the **Data catalog settings** page, select **Encrypt connection passwords**, and choose an AWS KMS key.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key** list displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

For more information, see [Working with Data Catalog Settings on the AWS Glue Console \(p. 144\)](#).

Encrypting Data Written by Crawlers, Jobs, and Development Endpoints

A *security configuration* is a set of security properties that can be used by AWS Glue. You can use a security configuration to encrypt data at rest. The following scenarios show some of the ways that you can use a security configuration.

- Attach a security configuration to an AWS Glue crawler to write encrypted Amazon CloudWatch Logs.
- Attach a security configuration to an extract, transform, and load (ETL) job to write encrypted Amazon Simple Storage Service (Amazon S3) targets and encrypted CloudWatch Logs.
- Attach a security configuration to an ETL job to write its jobs bookmarks as encrypted Amazon S3 data.
- Attach a security configuration to a development endpoint to write encrypted Amazon S3 targets.

Important

Currently, a security configuration overrides any server-side encryption (SSE-S3) setting that is passed as an ETL job parameter. Thus, if both a security configuration and an SSE-S3 parameter are associated with a job, the SSE-S3 parameter is ignored.

For more information about security configurations, see [Working with Security Configurations on the AWS Glue Console \(p. 46\)](#).

Topics

- [Setting Up AWS Glue to Use Security Configurations \(p. 44\)](#)
- [Creating a Route to AWS KMS for VPC Jobs and Crawlers \(p. 45\)](#)
- [Working with Security Configurations on the AWS Glue Console \(p. 46\)](#)

Setting Up AWS Glue to Use Security Configurations

Follow these steps to set up your AWS Glue environment to use security configurations.

1. Create or update your AWS Key Management Service (AWS KMS) keys to grant AWS KMS permissions to the IAM roles that are passed to AWS Glue crawlers and jobs to encrypt CloudWatch

Logs. For more information, see [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#) in the [Amazon CloudWatch Logs User Guide](#).

In the following example, "`role1`", "`role2`", and "`role3`" are IAM roles that are passed to crawlers and jobs.

```
{  
    "Effect": "Allow",  
    "Principal": { "Service": "logs.region.amazonaws.com",  
    "AWS": [  
        "role1",  
        "role2",  
        "role3"  
    ] },  
    "Action": [  
        "kms:Encrypt*",  
        "kms:Decrypt*",  
        "kms:ReEncrypt*",  
        "kms:GenerateDataKey*",  
        "kms:Describe*"  
    ],  
    "Resource": "*"  
}
```

The Service statement, shown as "Service": "logs.`region`.amazonaws.com", is required if you use the key to encrypt CloudWatch Logs.

2. Ensure that the AWS KMS key is **ENABLED** before it is used.
3. Ensure that the AWS Glue job includes the following code for the security setting to take effect.

```
job = Job(glueContext)  
job.init(args['JOB_NAME'], args)
```

Creating a Route to AWS KMS for VPC Jobs and Crawlers

You can connect directly to AWS KMS through a private endpoint in your virtual private cloud (VPC) instead of connecting over the internet. When you use a VPC endpoint, communication between your VPC and AWS KMS is conducted entirely within the AWS network.

You can create an AWS KMS VPC endpoint within a VPC. Without this step, your jobs or crawlers might fail with a `kms` timeout on jobs or an `internal service` exception on crawlers. For detailed instructions, see [Connecting to AWS KMS Through a VPC Endpoint](#) in the [AWS Key Management Service Developer Guide](#).

As you follow these instructions, on the [VPC console](#), you must do the following:

- Select **Enable Private DNS name**.
- Choose the **Security group** (with self-referencing rule) that you use for your job or crawler that accesses Java Database Connectivity (JDBC). For more information about AWS Glue connections, see [Adding a Connection to Your Data Store \(p. 110\)](#).

When you add a security configuration to a crawler or job that accesses JDBC data stores, AWS Glue must have a route to the AWS KMS endpoint. You can provide the route with a network address translation (NAT) gateway or with an AWS KMS VPC endpoint. To create a NAT gateway, see [NAT Gateways](#) in the [Amazon VPC User Guide](#).

Working with Security Configurations on the AWS Glue Console

A *security configuration* in AWS Glue contains the properties that are needed when you write encrypted data. You create security configurations on the AWS Glue console to provide the encryption properties that are used by crawlers, jobs, and development endpoints.

To see a list of all the security configurations that you have created, open the AWS Glue console at <https://console.aws.amazon.com/glue/> and choose **Security configurations** in the navigation pane.

The **Security configurations** list displays the following properties about each configuration:

Name

The unique name you provided when you created the configuration.

S3 encryption mode

If enabled, the Amazon Simple Storage Service (Amazon S3) encryption mode such as SSE-KMS or SSE-S3.

CloudWatch logs encryption mode

If enabled, the Amazon S3 encryption mode such as SSE-KMS.

Job bookmark encryption mode

If enabled, the Amazon S3 encryption mode such as CSE-KMS.

Date created

The date and time (UTC) that the configuration was created.

You can add or delete configurations in the **Security configurations** section on the console. To see more details for a configuration, choose the configuration name in the list. Details include the information that you defined when you created the configuration.

Adding a Security Configuration

To add a security configuration using the AWS Glue console, on the **Security configurations** page, choose **Add security configuration**. The wizard guides you through setting the required properties.

To set up encryption of data and metadata with AWS Key Management Service (AWS KMS) keys on the AWS Glue console, add a policy to the console user. This policy must specify the allowed resources as key Amazon Resource Names (ARNs) that are used to encrypt Amazon S3 data stores, as in the following example.

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt",
            "kms:Encrypt"
        ],
        "Resource": "arn:aws:kms:region:account-id:key/key-id"
    }
}
```

Important

When a security configuration is attached to a crawler or job, the IAM role that is passed must have AWS KMS permissions. For more information, see [Encrypting Data Written by Crawlers, Jobs, and Development Endpoints \(p. 44\)](#).

When you define a configuration, you can provide values for the following properties:

S3 encryption

When you are writing Amazon S3 data, you use either server-side encryption with Amazon S3 managed keys (SSE-S3) or server-side encryption with AWS KMS managed keys (SSE-KMS). This field is optional. To enable access to Amazon S3, choose an AWS KMS key, or choose **Enter a key ARN** and provide the ARN for the key. Enter the ARN in the form `arn:aws:kms:region:account-id:key/key-id`. You can also provide the ARN as a key alias, such as `arn:aws:kms:region:account-id:alias/alias-name`.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key list** displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

CloudWatch Logs encryption

Server-side (SSE-KMS) encryption is used to encrypt CloudWatch Logs. This field is optional. To enable it, choose an AWS KMS key, or choose **Enter a key ARN** and provide the ARN for the key. Enter the ARN in the form `arn:aws:kms:region:account-id:key/key-id`. You can also provide the ARN as a key alias, such as `arn:aws:kms:region:account-id:alias/alias-name`.

Job bookmark encryption

Client-side (CSE-KMS) encryption is used to encrypt job bookmarks. This field is optional. The bookmark data is encrypted before it is sent to Amazon S3 for storage. To enable it, choose an AWS KMS key, or choose **Enter a key ARN** and provide the ARN for the key. Enter the ARN in the form `arn:aws:kms:region:account-id:key/key-id`. You can also provide the ARN as a key alias, such as `arn:aws:kms:region:account-id:alias/alias-name`.

For more information, see the following topics in the *Amazon Simple Storage Service Developer Guide*:

- For information about SSE-S3, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#).
- For information about SSE-KMS, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\)](#).
- For information about CSE-KMS, see [Using an AWS KMS-Managed Customer Master Key \(CMK\)](#).

Encryption in Transit

AWS provides Secure Sockets Layer (SSL) encryption for data in motion. You can configure encryption settings for crawlers, ETL jobs, and development endpoints using [security configurations](#) in AWS Glue. You can enable AWS Glue Data Catalog encryption via the settings for the Data Catalog.

As of September 4, 2018, AWS KMS (*bring your own key and server-side encryption*) for AWS Glue ETL and the AWS Glue Data Catalog is supported.

Key Management

You can use AWS Identity and Access Management (IAM) with AWS Glue to define users, AWS resources, groups, roles and fine-grained policies regarding access, denial, and more.

You can define the access to the metadata using both resource-based and identity-based policies, depending on your organization's needs. Resource-based policies list the principals that are allowed or denied access to your resources, allowing you to set up policies such as cross-account access. Identity policies are specifically attached to users, groups, and roles within IAM.

For a step-by-step example, see [Restrict access to your AWS Glue Data Catalog with resource-level IAM permissions and resource-based policies](#) on the AWS Big Data Blog.

The fine-grained access portion of the policy is defined within the `Resource` clause. This portion defines both the AWS Glue Data Catalog object that the action can be performed on, and what resulting objects get returned by that operation.

A *development endpoint* is an environment that you can use to develop and test your AWS Glue scripts. You can add, delete, or rotate the SSH key of a development endpoint.

As of September 4, 2018, AWS KMS (*bring your own key and server-side encryption*) for AWS Glue ETL and the AWS Glue Data Catalog is supported.

Internetwork Traffic Privacy

AWS Glue uses other AWS services to orchestrate your extract, transform, and load (ETL) jobs to build a data warehouse. AWS Glue calls API operations to transform your data, create runtime logs, store your job logic, and create notifications to help you monitor your job runs. For more information, see [AWS Glue Dependency on Other AWS Services \(p. 50\)](#).

The AWS Glue console connects these services into a managed application. The console performs administrative and job development operations on your behalf. You supply credentials and other properties to AWS Glue to access your data sources and write to your data warehouse. AWS Glue takes care of provisioning and managing resources that are required to run your workload. When resources are required, to reduce startup time, AWS Glue uses an instance from its warm pool of instances to run your workload.

To start using AWS Glue, sign into the AWS Management Console. Under the **Analytics** category, choose **Glue**. AWS Glue automatically discovers and profiles your data via the AWS Glue Data Catalog. It recommends and generates ETL code to transform your source data into target schemas. It then runs the ETL jobs on a fully managed, scale-out Apache Spark environment to load your data into its destination. It also enables you to set up, orchestrate, and monitor complex data flows.

You can create jobs using table definitions in your Data Catalog. Jobs consist of scripts that contain programming logic that performs the transformation. You use triggers to initiate jobs either on a schedule or as a result of a specified event. You determine where your target data resides and which source data populates your target. With your input, AWS Glue generates the code that's required to transform your data from source to target. You can also provide scripts in the AWS Glue console or API to process your data.

Topics

- [AWS Glue Console \(p. 48\)](#)
- [AWS Glue Data Catalog \(p. 49\)](#)
- [AWS Glue Crawlers and Classifiers \(p. 49\)](#)
- [AWS Glue ETL Operations \(p. 49\)](#)
- [AWS Glue Jobs System \(p. 49\)](#)

AWS Glue Console

Use the AWS Glue console to define and orchestrate your ETL workflow. The console calls several API operations in the AWS Glue Data Catalog and AWS Glue jobs system to perform the following tasks:

- Define AWS Glue objects such as jobs, tables, crawlers, and connections
- Schedule when crawlers run
- Define events or schedules for job triggers
- Search and filter lists of AWS Glue objects

- Edit transformation scripts AWS Glue Data Catalog

AWS Glue Data Catalog

The AWS Glue Data Catalog is your persistent metadata store. It is a managed service that lets you store, annotate, and share metadata in the AWS Cloud in the same way you would in an Apache Hive metastore.

Each AWS account has one AWS Glue Data Catalog. It provides a uniform repository where disparate systems can store and find metadata to keep track of data in data silos, and use that metadata to query and transform the data.

AWS Identity and Access Management (IAM) policies control access to data sources managed by the AWS Glue Data Catalog. These policies allow different groups in your enterprise to safely publish data to the wider organization while protecting sensitive information. IAM policies let you clearly and consistently define which users have access to which data, regardless of its location.

The Data Catalog also provides comprehensive audit and governance capabilities, with schema change tracking, lineage of data, and data access controls. You can audit changes to data schemas and track the movement of data across systems. This helps ensure that data is not inappropriately modified or inadvertently shared.

For information about how to use the AWS Glue Data Catalog, see [Populating the AWS Glue Data Catalog \(p. 104\)](#). For information about how to program using the Data Catalog API, see [Catalog API \(p. 467\)](#).

AWS Glue Crawlers and Classifiers

AWS Glue sets up crawlers that scan data in all kinds of repositories. They classify the data, extract schema information from it, and store the metadata automatically in the AWS Glue Data Catalog. From there, it can be used to guide ETL operations.

For information about how to set up crawlers and classifiers, see [Defining Crawlers \(p. 116\)](#). For information about how to program crawlers and classifiers using the AWS Glue API, see [Crawlers and Classifiers API \(p. 512\)](#).

AWS Glue ETL Operations

Using metadata in the Data Catalog, AWS Glue generates Scala or PySpark (the Python API for Apache Spark) scripts with AWS Glue extensions. You can use and modify these scripts to perform various ETL operations. You can extract, clean, and transform raw data, and then store the result in a different repository, where it can be queried and analyzed. Such a script might convert a CSV file into a relational form and save it in Amazon Redshift.

For more information about how to use AWS Glue ETL capabilities, see [Programming ETL Scripts \(p. 292\)](#).

AWS Glue Jobs System

The AWS Glue jobs system provides managed infrastructure to orchestrate your ETL workflow. You create jobs in AWS Glue that automate scripts used to extract, transform, and transfer data to different locations. Jobs can be scheduled and chained, or they can be triggered by events such as the arrival of new data.

For more information about using the AWS Glue jobs system, see [Running and Monitoring AWS Glue \(p. 220\)](#). For information about programming using the AWS Glue jobs system API, see [Jobs API \(p. 540\)](#).

AWS Glue Dependency on Other AWS Services

For a user to work with the AWS Glue console, that user must have a minimum set of permissions that allows them to work with the AWS Glue resources for their AWS account. In addition to these AWS Glue permissions, the console requires permissions from the following services:

- Amazon CloudWatch Logs permissions to display logs.
- AWS Identity and Access Management (IAM) permissions to list and pass roles.
- Amazon CloudFront permissions to work with stacks.
- Amazon Elastic Compute Cloud (Amazon EC2) permissions to list virtual private clouds (VPCs), subnets, security groups, instances, and other objects (to set up Amazon EC2 items such as VPCs when running jobs, crawlers, and creating development endpoints).
- Amazon Simple Storage Service (Amazon S3) permissions to list buckets and objects, and to retrieve and save scripts.
- Amazon Redshift permissions to work with clusters.
- Amazon Relational Database Service (Amazon RDS) permissions to list instances.

Development Endpoints

A development endpoint is an environment that you can use to develop and test your AWS Glue scripts. You can use AWS Glue to create, edit, and delete development endpoints. The **Dev Endpoints** tab on the AWS Glue console lists all the development endpoints that are created. You can add, delete, or rotate the SSH key of a development endpoint. You can also create notebooks that use the development endpoint.

You provide configuration values to provision the development environments. These values tell AWS Glue how to set up the network so that you can access the development endpoint securely, and so that your endpoint can access your data stores. Then, you can create a notebook that connects to the development endpoint. You use your notebook to author and test your ETL script.

Use an AWS Identity and Access Management (IAM) role with permissions similar to the IAM role that you use to run AWS Glue ETL jobs. Use a virtual private cloud (VPC), a subnet, and a security group to create a development endpoint that can connect to your data resources securely. You generate an SSH key pair to connect to the development environment using SSH.

You can create development endpoints for Amazon S3 data and within a VPC that you can use to access datasets using JDBC.

You can install an Apache Zeppelin notebook on your local machine and use it to debug and test ETL scripts on a development endpoint. Or, you can host the Zeppelin notebook on an Amazon EC2 instance. A notebook server is a web-based environment that you can use to run your PySpark statements.

AWS Glue tags Amazon EC2 instances with a name that is prefixed with `aws-glue-dev-endpoint`.

You can set up a notebook server on a development endpoint to run PySpark statements with AWS Glue extensions. For more information about Zeppelin notebooks, see [Apache Zeppelin](#).

Identity and Access Management in AWS Glue

Access to AWS Glue requires credentials. Those credentials must have permissions to access AWS resources, such as an AWS Glue table or an Amazon Elastic Compute Cloud (Amazon EC2) instance. The following sections provide details on how you can use [AWS Identity and Access Management \(IAM\)](#) and AWS Glue to help secure access to your resources.

Topics

- [Authentication \(p. 51\)](#)
- [Managing Access Permissions for AWS Glue Resources \(p. 52\)](#)
- [Granting Cross-Account Access \(p. 61\)](#)
- [Specifying AWS Glue Resource ARNs \(p. 65\)](#)
- [AWS Glue Access Control Policy Examples \(p. 69\)](#)
- [AWS Glue API Permissions: Actions and Resources Reference \(p. 84\)](#)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An [IAM user](#) is an identity within your AWS account that has specific custom permissions (for example, permissions to create a table in AWS Glue). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. AWS Glue supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the [AWS General Reference](#).

- **IAM role** – An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:
 - **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the [IAM User Guide](#).
 - **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but

many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances](#) in the *IAM User Guide*.

Managing Access Permissions for AWS Glue Resources

You can have valid credentials to authenticate your requests, but unless you have the appropriate permissions, you can't create or access an AWS Glue resource such as a table in the AWS Glue Data Catalog.

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services (such as AWS Glue and Amazon S3) also support attaching permissions policies to the resources themselves.

Note

An *account administrator* (or administrator user) is a user who has administrative privileges. For more information, see [IAM Best Practices](#) in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

- [Using Permissions Policies to Manage Access to Resources \(p. 52\)](#)
- [AWS Glue Resources and Operations \(p. 53\)](#)
- [Understanding Resource Ownership \(p. 53\)](#)
- [Managing Access to Resources \(p. 53\)](#)
- [Specifying Policy Elements: Actions, Effects, and Principals \(p. 54\)](#)
- [Specifying Conditions in a Policy \(p. 55\)](#)
- [Identity-Based Policies \(IAM Policies\) for Access Control \(p. 55\)](#)
- [AWS Glue Resource Policies for Access Control \(p. 59\)](#)

Using Permissions Policies to Manage Access to Resources

A *permissions policy* is defined by a JSON object that describes who has access to what. The syntax of the JSON object is largely defined by AWS Identity and Access Management (IAM). For more information, see [IAM JSON Policy Reference](#) in the *IAM User Guide*.

Note

This section discusses using IAM in the context of AWS Glue, but it does not provide detailed information about the IAM service. For more information, see [What Is IAM?](#) in the *IAM User Guide*.

For a list showing all of the AWS Glue API operations and the resources that they apply to, see [AWS Glue API Permissions: Actions and Resources Reference \(p. 84\)](#).

To learn more about IAM policy syntax and descriptions, see [IAM JSON Policy Reference](#) in the *IAM User Guide*.

AWS Glue supports two kinds of policies:

- [Identity-Based Policies \(IAM Policies\) for Access Control \(p. 55\)](#)
- [AWS Glue Resource Policies for Access Control \(p. 59\)](#)

By supporting both identity-based and resource policies, AWS Glue gives you fine-grained control over who can access what metadata.

For more examples, see [AWS Glue Resource-Based Access Control Policy Examples \(p. 79\)](#).

AWS Glue Resources and Operations

AWS Glue provides a set of operations to work with AWS Glue resources. For a list of available operations, see [AWS Glue API \(p. 452\)](#).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the [principal entity](#) (that is, the AWS account root user, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the AWS account root user credentials of your AWS account to create a table, your AWS account is the owner of the resource (in AWS Glue, the resource is a table).
- If you create an IAM user in your AWS account and grant permissions to create a table to that user, the user can create a table. However, your AWS account, which the user belongs to, owns the table resource.
- If you create an IAM role in your AWS account with permissions to create a table, anyone who can assume the role can create a table. Your AWS account, to which the user belongs, owns the table resource.

Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

Note

This section discusses using IAM in the context of AWS Glue. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see [What Is IAM?](#) in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see [IAM JSON Policy Reference](#) in the *IAM User Guide*.

Policies that are attached to an IAM identity are referred to as *identity-based* policies (IAM policies). Policies that are attached to a resource are referred to as *resource-based* policies.

Topics

- [Identity-Based Policies \(IAM Policies\) \(p. 54\)](#)
- [Resource-Based Policies \(p. 54\)](#)

Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create an AWS Glue resource, such as a table, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
 1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
 2. Account A administrator attaches a trust policy to the role identifying account B as the principal who can assume the role.
 3. Account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see [Access Management](#) in the *IAM User Guide*.

The following is an example identity-based policy that grants permissions for one AWS Glue action (`GetTables`). The wildcard character (*) in the `Resource` value means that you are granting permission to this action to obtain names and details of all the tables in a database in the Data Catalog. If the user also has access to other catalogs through a resource policy, it is given access to these resources too.

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

For more information about using identity-based policies with AWS Glue, see [Identity-Based Policies \(IAM Policies\) for Access Control \(p. 55\)](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket.

Specifying Policy Elements: Actions, Effects, and Principals

For each AWS Glue resource, the service defines a set of API operations. To grant permissions for these API operations, AWS Glue defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more

information about resources and API operations, see [AWS Glue Resources and Operations \(p. 53\)](#) and [AWS Glue AWS Glue API \(p. 452\)](#).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see [AWS Glue Resources and Operations \(p. 53\)](#).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `create` to allow users to create a table.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). AWS Glue doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see [IAM JSON Policy Reference](#) in the *IAM User Guide*.

For a list showing all of the AWS Glue API operations and the resources that they apply to, see [AWS Glue API Permissions: Actions and Resources Reference \(p. 84\)](#).

Specifying Conditions in a Policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Condition](#) in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are AWS-wide condition keys and AWS Glue-specific keys that you can use as appropriate. For a complete list of AWS-wide keys, see [Available Keys for Conditions](#) in the *IAM User Guide*.

Identity-Based Policies (IAM Policies) for Access Control

Identity-based policies are attached to an IAM identity (user, group, role, or service). This type of policy grants permissions for that IAM identity to access specified resources.

AWS Glue supports identity-based policies (IAM policies) for all AWS Glue operations. By attaching a policy to a user or a group in your account, you can grant them permissions to create, access, or modify an AWS Glue resource, such as a table in the AWS Glue Data Catalog.

By attaching a policy to an IAM role, you can grant cross-account access permissions to IAM identities in other AWS accounts. For more information, see [Granting Cross-Account Access \(p. 61\)](#).

The following is an example identity-based policy that grants permissions for AWS Glue actions (`glue:GetTable`, `GetTables`, `GetDatabase`, and `GetDatabases`). The wildcard character (*) in the Resource value means that you are granting permission to these actions to obtain names and details of all the tables and databases in the Data Catalog. If the user also has access to other catalogs through a resource policy, then it is given access to these resources too.

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

```

        "Sid": "GetTables",
        "Effect": "Allow",
        "Action": [
            "glue:GetTable",
            "glue:GetTables",
            "glue:GetDatabase",
            "glue:GetDataBases"
        ],
        "Resource": "*"
    }
]
}

```

Here is another example, targeting the us-west-2 Region and using a placeholder for the specific AWS account number.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesActionOnBooks",
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/books"
            ]
        }
    ]
}
```

This policy grants read-only permission to a table named books in the database named db1. Notice that to grant Get permission to a table that permission to the catalog and database resources is also required.

To deny access to a table, requires that you create a policy to deny a user access to the table, or its parent database or catalog. This allows you to easily deny access to a specific resource that cannot be circumvented with a subsequent allow permission. For example, if you deny access to table books in database db1, then if you grant access to database db1, access to table books is still denied. The following is an example identity-based policy that denies permissions for AWS Glue actions (glue:GetTables and glue:GetTable) to database db1 and all of the tables within it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyGetTablesToDb1",
            "Effect": "Deny",
            "Action": [
                "glue:GetTables",
                "glue:GetTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:database/db1"
            ]
        }
    ]
}
```

For more policy examples, see [Identity-Based Policy Examples \(p. 69\)](#).

Identity-Based Policies (IAM Policies) with Tags

You can also control access to certain types of AWS Glue resources using AWS tags. For more information about tags in AWS Glue, see [AWS Tags \(p. 229\)](#).

You can use the `Condition` element along with the `glue:resourceTag` context key in an IAM user policy to allow or deny access based on keys associated with crawlers, jobs, triggers, and development endpoints. For example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "glue:*",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "glue:resourceTag/Name": "Tom"  
                }  
            }  
        }  
    ]  
}
```

Important

The condition context keys apply only to those AWS Glue API actions on crawlers, jobs, triggers, and development endpoints. For more information about which APIs are affected, see [AWS Glue API Permissions: Actions and Resources Reference \(p. 84\)](#).

For information about how to control access using tags, see [AWS Glue Identity-Based \(IAM\) Access Control Policy with Tags Examples \(p. 74\)](#).

Resource-Level Permissions Only Apply to Specific AWS Glue Objects

You can only define fine-grained control for specific objects in AWS Glue. Therefore you must write your client's IAM policy so that API operations that allow Amazon Resource Names (ARNs) for the `Resource` statement are not mixed with API operations that don't allow ARNs. For example, the following IAM policy allows API operations for `GetClassifier` and `GetJobRun`. It defines the `Resource` as `*` because AWS Glue doesn't allow ARNs for classifiers and job runs. Because ARNs are allowed for specific API operations such as `GetDatabase` and `GetTable`, ARNs can be specified in the second half of the policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetClassifier*",  
                "glue:GetJobRun*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue:Get*"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-east-1:123456789012:catalog",  
                "arn:aws:glue:us-east-1:123456789012:database/  
            ]  
        }  
    ]  
}
```

```
        "arn:aws:glue:us-east-1:123456789012:database/default",
        "arn:aws:glue:us-east-1:123456789012:table/default/e*1*",
        "arn:aws:glue:us-east-1:123456789012:connection/connection2"
    ]
}
}
```

For a list of AWS Glue objects that allow ARNs, see [Resource ARNs \(p. 65\)](#).

Permissions Required to Use the AWS Glue Console

For a user to work with the AWS Glue console, that user must have a minimum set of permissions that allows them to work with the AWS Glue resources for their AWS account. In addition to these AWS Glue permissions, the console requires permissions from the following services:

- Amazon CloudWatch Logs permissions to display logs.
- AWS Identity and Access Management (IAM) permissions to list and pass roles.
- AWS CloudFormation permissions to work with stacks.
- Amazon Elastic Compute Cloud (Amazon EC2) permissions to list VPCs, subnets, security groups, instances, and other objects.
- Amazon Simple Storage Service (Amazon S3) permissions to list buckets and objects, and to retrieve and save scripts.
- Amazon Redshift permissions to work with clusters.
- Amazon Relational Database Service (Amazon RDS) permissions to list instances.

For more information about the permissions that users require to view and work with the AWS Glue console, see [Step 3: Attach a Policy to IAM Users That Access AWS Glue \(p. 15\)](#).

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the AWS Glue console, also attach the `AWSGlueConsoleFullAccess` managed policy to the user, as described in [AWS Managed \(Predefined\) Policies for AWS Glue \(p. 58\)](#).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS Glue API.

AWS Managed (Predefined) Policies for AWS Glue

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see [AWS Managed Policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to AWS Glue and are grouped by use case scenario:

- **AWSGlueConsoleFullAccess** – Grants full access to AWS Glue resources when using the AWS Management Console. If you follow the naming convention for resources specified in this policy, users have full console capabilities. This policy is typically attached to users of the AWS Glue console.
- **AWSGlueServiceRole** – Grants access to resources that various AWS Glue processes require to run on your behalf. These resources include AWS Glue, Amazon S3, IAM, CloudWatch Logs, and Amazon EC2. If you follow the naming convention for resources specified in this policy, AWS Glue processes have the required permissions. This policy is typically attached to roles specified when defining crawlers, jobs, and development endpoints.
- **AWSGlueServiceNotebookRole** – Grants access to resources required when creating a notebook server. These resources include AWS Glue, Amazon S3, and Amazon EC2. If you follow the naming

convention for resources specified in this policy, AWS Glue processes have the required permissions. This policy is typically attached to roles specified when creating a notebook server on a development endpoint.

- **AWSGlueConsoleSageMakerNotebookFullAccess** – Grants full access to AWS Glue and Amazon SageMaker resources when using the AWS Management Console. If you follow the naming convention for resources specified in this policy, users have full console capabilities. This policy is typically attached to users of the AWS Glue console who manage Amazon SageMaker notebooks.

Note

You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for AWS Glue actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

AWS Glue Resource Policies for Access Control

A *resource policy* is a policy that is attached to a resource rather than to an IAM identity. For example, in Amazon Simple Storage Service (Amazon S3), a resource policy is attached to an Amazon S3 bucket. AWS Glue supports using resource policies to control access to Data Catalog resources. These resources include databases, tables, connections, and user-defined functions, along with the Data Catalog APIs that interact with these resources.

An AWS Glue resource policy can only be used to manage permissions for Data Catalog resources. You can't attach it to any other AWS Glue resources such as jobs, triggers, development endpoints, crawlers, or classifiers.

A resource policy is attached to a *catalog*, which is a virtual container for all the kinds of Data Catalog resources mentioned previously. Each AWS account owns a single catalog in an AWS Region whose catalog ID is the same as the AWS account ID. A catalog cannot be deleted or modified.

A resource policy is evaluated for all API calls to the catalog where the caller principal is included in the "Principal" block of the policy document.

Note

Currently, only one resource policy is allowed per catalog, and its size is limited to 10 KB.

You use a policy document written in JSON format to create or modify a resource policy. The policy syntax is the same as for an IAM policy (see [IAM JSON Policy Reference](#)), with the following exceptions:

- A "Principal" or "NotPrincipal" block is required for each policy statement.
- The "Principal" or "NotPrincipal" must identify valid existing AWS root users or IAM users, roles, or groups. Wildcard patterns (like `arn:aws:iam::account-id:user/*`) are not allowed.
- The "Resource" block in the policy requires all resource ARNs to match the following regular expression syntax (where the first %s is the `region` and the second %s is the `account-id`):

```
*arn:aws:glue:%s:%s:(\*|[a-zA-Z\*]+\/\?.*)
```

For example, both `arn:aws:glue:us-west-2:account-id:*` and `arn:aws:glue:us-west-2:account-id:database/default` are allowed, but `*` is not allowed.

- Unlike identity-based policies, an AWS Glue resource policy must only contain Amazon Resource Names (ARNs) of resources belonging to the catalog to which the policy is attached. Such ARNs always start with `arn:aws:glue:..`.
- A policy cannot cause the identity creating it to be locked out of further policy creation or modification.
- A resource-policy JSON document cannot exceed 10 KB in size.

As an example, suppose that the following policy is attached to the catalog in Account A. It grants to the IAM identity `dev` in Account A permission to create any table in database `db1` in Account A. It also grants the same permission to the root user in Account B.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-A-id:user/dev",
        "arn:aws:iam::account-B-id:root"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:table/db1/*",
        "arn:aws:glue:us-east-1:account-A-id:database/db1",
        "arn:aws:glue:us-east-1:account-A-id:catalog"
      ]
    }
  ]
}
```

The following are some examples of resource policy documents that are *not* valid.

For example, a policy is not valid if it specifies a user that does not exist in the account of the catalog to which it is attached.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-A-id:user/(non-existent-user)"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:table/db1/tbl1",
        "arn:aws:glue:us-east-1:account-A-id:database/db1",
        "arn:aws:glue:us-east-1:account-A-id:catalog"
      ]
    }
  ]
}
```

A policy is not valid if it contains a resource ARN for a resource in a different account than the catalog to which it is attached. In the following example, this is an incorrect policy if attached to account-A.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-A-id:user/dev"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-B-id:table/db1/*"
      ]
    }
  ]
}
```

```

        ],
      "Resource": [
        "arn:aws:glue:us-east-1:account-B-id:table/db1/tbl1",
        "arn:aws:glue:us-east-1:account-B-id:database/db1",
        "arn:aws:glue:us-east-1:account-B-id:catalog"
      ]
    }
}

```

A policy is not valid if it contains a resource ARN for a resource that is not an AWS Glue resource (in this case, an Amazon S3 bucket).

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-A-id:user/dev"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:table/db1/tbl1",
        "arn:aws:glue:us-east-1:account-A-id:database/db1",
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:s3:::bucket/my-bucket"
      ]
    }
  ]
}

```

AWS Glue Resource Policy APIs

You can use the following AWS Glue Data Catalog APIs to create, retrieve, modify, and delete an AWS Glue resource policy:

- [PutResourcePolicy \(put_resource_policy\) \(p. 463\)](#)
- [GetResourcePolicy \(get_resource_policy\) \(p. 464\)](#)
- [DeleteResourcePolicy \(delete_resource_policy\) \(p. 465\)](#)

You can also use the AWS Glue console to view and edit a resource policy. For more information, see [Working with Data Catalog Settings on the AWS Glue Console \(p. 144\)](#).

Granting Cross-Account Access

There are two ways in AWS to grant cross-account access to a resource:

- Use a resource policy
- Use an IAM role

To use a resource policy to grant cross-account access

1. An administrator (or other authorized identity) in Account A attaches a resource policy to the Data Catalog in Account A. This policy grants Account B specific cross-account permissions to perform operations on a resource in Account A's catalog.

2. An administrator in Account B attaches an IAM policy to a user or other IAM identity in Account B that delegates the permissions received from Account A.
3. The user or other identity in Account B now has access to the specified resource in Account A.

The user needs permission from *both* the resource owner (Account A) *and* their parent account (Account B) to be able to access the resource.

To use an IAM role to grant cross-account access

1. An administrator (or other authorized identity) in the account that owns the resource (Account A) creates an IAM role.
2. The administrator in Account A attaches a policy to the role that grants cross-account permissions for access to the resource in question.
3. The administrator in Account A attaches a trust policy to the role that identifies an IAM identity in a different account (Account B) as the principal who can assume the role.

The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permission to assume the role.

4. An administrator in Account B now delegates permissions to one or more IAM identities in Account B so that they can assume that role. Doing so gives those identities in Account B access to the resource in account A.

For more information about using IAM to delegate permissions, see [Access Management in the IAM User Guide](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

For a comparison of these two approaches, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*. AWS Glue supports both options, with the restriction that a resource policy can grant access only to Data Catalog resources.

For example, to give user Bob in Account B access to database db1 in Account A, attach the following resource policy to the catalog in Account A.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Principal": {"AWS": [
          "arn:aws:iam::account-B-id:user/Bob"
        ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}
```

In addition, Account B would have to attach the following IAM policy to Bob before he would actually get access to db1 in Account A.

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

```

"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "glue:GetDatabase"
        ],
        "Resource": [
            "arn:aws:glue:us-east-1:account-A-id:catalog",
            "arn:aws:glue:us-east-1:account-A-id:database/db1"
        ]
    }
]
}

```

Making a Cross-Account API Call

All AWS Glue Data Catalog operations have a `CatalogId` field. If the required permissions have been granted to enable cross-account access, a caller can make Data Catalog API calls across accounts. The caller does this by passing the target AWS account ID in `CatalogId` so as to access the resource in that target account.

If no `CatalogId` value is provided, AWS Glue uses the caller's own account ID by default, and the call is not cross-account.

Making a Cross-Account ETL Call

Some AWS Glue PySpark and Scala APIs have a catalog ID field. If all the required permissions have been granted to enable cross-account access, an ETL job can make PySpark and Scala calls to API operations across accounts by passing the target AWS account ID in the catalog ID field to access Data Catalog resources in a target account.

If no catalog ID value is provided, AWS Glue uses the caller's own account ID by default, and the call is not cross-account.

For PySpark APIs that support `catalog_id`, see [GlueContext Class \(p. 352\)](#). For Scala APIs that support `catalogId`, see [AWS Glue Scala GlueContext APIs \(p. 415\)](#).

The following example shows the permissions required by the grantee to run an ETL job. In this example, `grantee-account-id` is the catalog-id of the client running the job and `grantor-account-id` is the owner of the resource. This example grants permission to all catalog resources in the grantor's account. To limit the scope of resources granted, you can provide specific ARNs for the catalog, database, table, and connection.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetConnection",
                "glue:GetDatabase",
                "glue:GetTable",
                "glue:GetPartition"
            ],
            "Principal": {"AWS": ["arn:aws:iam:grantee-account-id:root"]},
            "Resource": [
                "arn:aws:glue:us-east-1:grantor-account-id:*"
            ]
        }
    ]
}

```

Note

If a table in the grantor's account points to an Amazon S3 location that is also in the grantor's account, the IAM role used to run an ETL job in the grantee's account must have permission to list and get objects from the grantor's account.

Given that the client in Account A already has permission to create and run ETL jobs, the following are the basic steps to set up an ETL job for cross-account access:

1. Allow cross-account data access (skip this step if Amazon S3 cross-account access is already set up).
 - a. Update the Amazon S3 bucket policy in Account B to allow cross-account access from Account A.
 - b. Update the IAM policy in Account A to allow access to the bucket in Account B.
2. Allow cross-account Data Catalog access.
 - a. Create or update the resource policy attached to the Data Catalog in Account B to allow access from Account A.
 - b. Update the IAM policy in Account A to allow access to the Data Catalog in Account B.

AWS Glue Resource Ownership and Operations

Your AWS account owns the AWS Glue Data Catalog resources that are created in that account, regardless of who created them. Specifically, the owner of a resource is the AWS account of the **principal entity** (that is, the AWS account root user, the IAM user, or the IAM role) that *authenticated* the creation request for that resource; for example:

- If you use your AWS account root user credentials to create a table in your Data Catalog, your AWS account is the owner of the resource.
- If you create an IAM user in your AWS account and grant permissions to that user to create a table, every table that the user creates is owned by your AWS account, to which the user belongs.
- If you create an IAM role in your AWS account with permissions to create a table, anyone who can assume the role can create a table. But again, your AWS account owns the table resources that are created using that role.

For each AWS Glue resource, the service defines a set of API operations that apply to it. To grant permissions for these API operations, AWS Glue defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation.

Cross-Account Resource Ownership and Billing

When a user in one AWS account (Account A) creates a new resource such as a database in a different account (Account B), that resource is then owned by Account B, the account where it was created. An administrator in Account B automatically gets full permissions to access the new resource, including reading, writing, and granting access permissions to a third account. The user in Account A can access the resource that they just created only if they have the appropriate permissions granted by Account B.

Storage costs and other costs that are directly associated with the new resource are billed to Account B, the resource owner. The cost of requests from the user who created the resource are billed to the requester's account, Account A.

For more information about AWS Glue billing and pricing, see [How AWS Pricing Works](#).

Cross-Account Access Limitations

AWS Glue cross-account access has the following limitations:

- Cross-account access to AWS Glue is not allowed if the resource owner account has not migrated the Amazon Athena data catalog to AWS Glue. You can find the current migration status using the [GetCatalogImportStatus \(get_catalog_import_status\) \(p. 512\)](#). For more details on how to migrate an Athena catalog to AWS Glue, see [Upgrading to the AWS Glue Data Catalog Step-by-Step](#) in the [Amazon Athena User Guide](#).
- Cross-account access is *only* supported for Data Catalog resources, including databases, tables, user-defined functions, and connections.
- Cross-account access to the Data Catalog is not supported when using an AWS Glue crawler, or Amazon Athena.

Specifying AWS Glue Resource ARNs

In AWS Glue, you can control access to resources using an AWS Identity and Access Management (IAM) policy. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. Not all resources in AWS Glue support ARNs.

Topics

- [Data Catalog ARNs \(p. 65\)](#)
- [ARNs for Non-Catalog Objects \(p. 67\)](#)
- [Access Control for AWS Glue Non-Catalog Singular API Operations \(p. 67\)](#)
- [Access Control for AWS Glue Non-Catalog API Operations That Retrieve Multiple Items \(p. 68\)](#)
- [Access Control for AWS Glue Non-Catalog Batch Get API Operations \(p. 69\)](#)

Data Catalog ARNs

Data Catalog resources have a hierarchical structure, with `catalog` as the root.

```
arn:aws:glue:<region>:<account-id>:catalog
```

Each AWS account has a single Data Catalog in an AWS Region with the 12-digit account ID as the catalog ID. Resources have unique ARNs associated with them, as shown in the following table.

Resource Type	ARN Format
Catalog	<code>arn:aws:glue:<region>:<account-id>:catalog</code> For example: <code>arn:aws:glue:us-east-1:123456789012:catalog</code>
Database	<code>arn:aws:glue:<region>:<account-id>:database/<database name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:database/db1</code>
Table	<code>arn:aws:glue:<region>:<account-id>:table/<database name>/<table name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:table/db1/tbl1</code>
User-defined function	<code>arn:aws:glue:<region>:<account-id>:userDefinedFunction/<database name>/<user-defined function name></code>

Resource Type	ARN Format
	For example: arn:aws:glue:us-east-1:123456789012:userDefinedFunction/db1/func1
Connection	arn:aws:glue:<i>region</i>:<i>account-id</i>:connection/<i>connection name</i> For example: arn:aws:glue:us-east-1:123456789012:connection/connection1

To enable fine-grained access control, you can use these ARNs in your IAM policies and resource policies to grant and deny access to specific resources. Wildcards are allowed in the policies. For example, the following ARN matches all tables in database default.

```
arn:aws:glue:us-east-1:123456789012:table/default/*
```

Important

All operations performed on a Data Catalog resource require permission on the resource and all the ancestors of that resource. For example, to create a partition for a table requires permission on the table, database, and catalog where the table is located. The following example shows the permission required to create partitions on table `PrivateTable` in database `PrivateDatabase` in the Data Catalog.

```
{
    "Sid": "GrantCreatePartitions",
    "Effect": "Allow",
    "Action": [
        "glue:BatchCreatePartitions"
    ],
    "Resource": [
        "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/PrivateTable",
        "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
        "arn:aws:glue:us-east-1:123456789012:catalog"
    ]
}
```

In addition to permission on the resource and all its ancestors, all delete operations require permission on all children of that resource. For example, deleting a database requires permission on all the tables and user-defined functions in the database, in addition to the database and the catalog where the database is located. The following example shows the permission required to delete database `PrivateDatabase` in the Data Catalog.

```
{
    "Sid": "GrantDeleteDatabase",
    "Effect": "Allow",
    "Action": [
        "glue>DeleteDatabase"
    ],
    "Resource": [
        "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/*",
        "arn:aws:glue:us-east-1:123456789012:userDefinedFunction/PrivateDatabase/*",
        "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
        "arn:aws:glue:us-east-1:123456789012:catalog"
    ]
}
```

In summary, actions on Data Catalog resources follow these permission rules:

- Actions on the catalog require permission on the catalog only.

- Actions on a database require permission on the database and catalog.
- Delete actions on a database require permission on the database and catalog plus all tables and user-defined functions in the database.
- Actions on a table, partition, or table version require permission on the table, database, and catalog.
- Actions on a user-defined function require permission on the user-defined function, database, and catalog.
- Actions on a connection require permission on the connection and catalog.

ARNs for Non-Catalog Objects

Some AWS Glue resources allow resource-level permissions to control access using an ARN. You can use these ARNs in your IAM policies to enable fine-grained access control. The following table lists the resources that can contain resource ARNs.

Resource Type	ARN Format
Crawler	<code>arn:aws:glue:<i>region</i>:<i>account-id</i>:crawler/<i>crawler-name</i></code> For example: <code>arn:aws:glue:us-east-1:123456789012:crawler/mycrawler</code>
Job	<code>arn:aws:glue:<i>region</i>:<i>account-id</i>:job/<i>job-name</i></code> For example: <code>arn:aws:glue:us-east-1:123456789012:job/testjob</code>
Trigger	<code>arn:aws:glue:<i>region</i>:<i>account-id</i>:trigger/<i>trigger-name</i></code> For example: <code>arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger</code>
Development endpoint	<code>arn:aws:glue:<i>region</i>:<i>account-id</i>:devEndpoint/<i>development-endpoint-name</i></code> For example: <code>arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint</code>

Access Control for AWS Glue Non-Catalog Singular API Operations

AWS Glue non-catalog *singular* API operations act on a single item (development endpoint). Examples are `GetDevEndpoint`, `CreateUpdateDevEndpoint`, and `UpdateDevEndpoint`. For these operations, a policy must put the API name in the "action" block and the resource ARN in the "resource" block.

Suppose that you want to allow a user to call the `GetDevEndpoint` operation. The following policy grants the minimum necessary permissions to an endpoint named `myDevEndpoint-1`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Minimum permissions",
            "Effect": "Allow",
            "Action": "glue:GetDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-1"
    ]}
```

```
        ]
    }
}
```

The following policy allows `UpdateDevEndpoint` access to resources that match `myDevEndpoint-` with a wildcard (*).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Permission with wildcard",
            "Effect": "Allow",
            "Action": "glue:UpdateDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-*"
        }
    ]
}
```

You can combine the two policies as in the following example. You might see `EntityNotFoundException` for any development endpoint whose name begins with A. However, an access denied error is returned when you try to access other development endpoints.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Combined permissions",
            "Effect": "Allow",
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint"
            ],
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/A*"
        }
    ]
}
```

Access Control for AWS Glue Non-Catalog API Operations That Retrieve Multiple Items

Some AWS Glue API operations retrieve multiple items (such as multiple development endpoints); for example, `GetEndpoints`. For this operation, you can specify only a wildcard (*) resource, and not specific ARNs.

For example, to include `GetEndpoints` in the policy, the resource must be scoped to the wildcard (*). The singular operations (`GetEndpoint`, `CreateEndpoint`, and `DeleteEndpoint`) are also scoped to all (*) resources in the example.

```
{
    "Sid": "Plural API included",
    "Effect": "Allow",
    "Action": [
        "glue:GetEndpoints",
        "glue:GetEndpoint",
        "glue>CreateEndpoint",
        "glue:UpdateEndpoint"
    ],
    "Resource": [
        "*"
    ]
}
```

```
        ]  
    }
```

Access Control for AWS Glue Non-Catalog Batch Get API Operations

Some AWS Glue API operations retrieve multiple items (such as multiple development endpoints); for example, `BatchGetDevEndpoints`. For this operation, you can specify an ARN to limit the scope of resources that can be accessed.

For example, to allow access to a specific development endpoint, include `BatchGetDevEndpoints` in the policy with its resource ARN.

```
{  
    "Sid": "BatchGet API included",  
    "Effect": "Allow",  
    "Action": [  
        "glue:BatchGetDevEndpoints"  
    ],  
    "Resource": [  
        "arn:aws:glue:us-east-1:123456789012:devEndpoint/de1"  
    ]  
}
```

With this policy, you can successfully access the development endpoint named `de1`. However, if you try to access the development endpoint named `de2`, an error is returned.

```
An error occurred (AccessDeniedException) when calling the BatchGetDevEndpoints operation:  
No access to any requested resource.
```

Important

For alternative approaches to setting up IAM policies, such as using `List` and `BatchGet` API operations, see [AWS Glue Identity-Based \(IAM\) Access Control Policy with Tags Examples \(p. 74\)](#).

AWS Glue Access Control Policy Examples

This section contains examples of both identity-based (IAM) access control policies and AWS Glue resource policies.

Topics

- [AWS Glue Identity-Based \(IAM\) Access Control Policy Examples \(p. 69\)](#)
- [AWS Glue Identity-Based \(IAM\) Access Control Policy with Tags Examples \(p. 74\)](#)
- [AWS Glue Resource-Based Access Control Policy Examples \(p. 79\)](#)

AWS Glue Identity-Based (IAM) Access Control Policy Examples

This section contains example AWS Identity and Access Management (IAM) policies that grant permissions for various AWS Glue actions and resources. You can copy these examples and edit them on the IAM console. Then you can attach them to IAM identities such as users, roles, and groups.

Note

These examples all use the `us-west-2` Region. You can replace this with whatever AWS Region you are using.

Example 1: Grant Read-Only Permission to a Table

The following policy grants read-only permission to a books table in database db1. For more information about resource Amazon Resource Names (ARNs), see [Data Catalog ARNs \(p. 65\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "GetTablesActionOnBooks",  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetTables",  
                "glue:GetTable"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-west-2:123456789012:catalog",  
                "arn:aws:glue:us-west-2:123456789012:database/db1",  
                "arn:aws:glue:us-west-2:123456789012:table/db1/books"  
            ]  
        }  
    ]  
}
```

This policy grants read-only permission to a table named books in the database named db1. Notice that to grant Get permission to a table that permission to the catalog and database resources is also required.

The following policy grants the minimum necessary permissions to create table tb1 in database db1:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue>CreateTable"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-west-2:123456789012:table/db1/tb1",  
                "arn:aws:glue:us-west-2:123456789012:database/db1",  
                "arn:aws:glue:us-west-2:123456789012:catalog"  
            ]  
        }  
    ]  
}
```

Example 2: Filter Tables by GetTables Permission

Assume that there are three tables—customers, stores, and store_sales—in database db1. The following policy grants GetTables permission to stores and store_sales, but not to customers. When you call GetTables with this policy, the result contains only the two authorized tables (the customers table is not returned).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "GetTablesExample",  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetTables"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-west-2:123456789012:table/db1/stores",  
                "arn:aws:glue:us-west-2:123456789012:table/db1/store_sales"  
            ]  
        }  
    ]  
}
```

```

        "glue:GetTables"
    ],
    "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/store_sales",
        "arn:aws:glue:us-west-2:123456789012:table/db1/stores"
    ]
}
]
```

You can simplify the preceding policy by using `store*` to match any table names that start with `store`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesExample2",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/store*"
            ]
        }
    ]
}
```

Similarly, using `/db1/*` to match all tables in db1, the following policy grants GetTables access to all the tables in db1.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesReturnAll",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
        }
    ]
}
```

If no table ARN is provided, a call to GetTables succeeds, but it returns an empty list.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesEmptyResults",
            "Effect": "Allow",
            "Action": [

```

```

        "glue:GetTables"
    ],
    "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1"
    ]
}
]
}

```

If the database ARN is missing in the policy, a call to GetTables fails with an AccessDeniedException.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesAccessDeny",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
        }
    ]
}

```

Example 3: Grant Full Access to a Table and All Partitions

The following policy grants all permissions on a table named books in database db1. This includes read and write permissions on the table itself, on archived versions of it, and on all its partitions.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:DeleteTableVersion",
                "glue:BatchDeleteTableVersion",
                "glue>CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:UpdatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",

```

```

        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
    ]
}
]
```

The preceding policy can be simplified in practice.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:*Table*",
                "glue:*Partition*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/books"
            ]
        }
    ]
}
```

Notice that the minimum granularity of fine-grained access control is at the table level. This means that you can't grant a user access to some partitions in a table but not others, or to some table columns but not to others. A user either has access to all of a table, or to none of it.

Example 4: Control Access by Name Prefix and Explicit Denial

In this example, suppose that the databases and tables in your AWS Glue Data Catalog are organized using name prefixes. The databases in the development stage have the name prefix dev-, and those in production have the name prefix prod-. You can use the following policy to grant developers full access to all databases, tables, UDFs, and so on, that have the dev- prefix. But you grant read-only access to everything with the prod- prefix.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DevAndProdFullAccess",
            "Effect": "Allow",
            "Action": [
                "glue:*Database*",
                "glue:*Table*",
                "glue:*Partition*",
                "glue:*UserDefinedFunction*",
                "glue:*Connection*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/dev-*",
                "arn:aws:glue:us-west-2:123456789012:database/prod-*",
                "arn:aws:glue:us-west-2:123456789012:table/dev-/*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/*/*",
                "arn:aws:glue:us-west-2:123456789012:table/prod-/*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/dev-/*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/*"
            ]
        }
    ]
}
```

```

        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/dev-*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/*prod-*",
        "arn:aws:glue:us-west-2:123456789012:connection/dev-*",
        "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
    ],
},
{
    "Sid": "ProdWriteDeny",
    "Effect": "Deny",
    "Action": [
        "glue:*Create*",
        "glue:*Update*",
        "glue:*Delete*"
    ],
    "Resource": [
        "arn:aws:glue:us-west-2:123456789012:database/prod-*",
        "arn:aws:glue:us-west-2:123456789012:table/prod-*/*",
        "arn:aws:glue:us-west-2:123456789012:table/*/*prod-*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/*prod-*",
        "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
    ]
}
]
}

```

The second statement in the preceding policy uses explicit deny. You can use explicit deny to overwrite any allow permissions that are granted to the principal. This lets you lock down access to critical resources and prevent another policy from accidentally granting access to them.

In the preceding example, even though the first statement grants full access to prod- resources, the second statement explicitly revokes write access to them, leaving only read access to prod- resources.

AWS Glue Identity-Based (IAM) Access Control Policy with Tags Examples

This section contains example AWS Identity and Access Management (IAM) policies with tags to control permissions for various AWS Glue actions and resources. You can copy these examples and edit them on the IAM console. Then you can attach them to IAM identities such as users, roles, and groups.

You can control access to crawlers, jobs, triggers, and development endpoints by attaching tags and specifying `resourceTag` conditions in IAM policies.

Example Access Control Using Tags

For example, suppose that you want to limit access to a trigger t2 to a specific user named Tom in your account. All other users, including Sam, have access to trigger t1. The triggers t1 and t2 have the following properties.

```

aws glue get-triggers
{
    "Triggers": [
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t1",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
        },
        {
            "State": "ENABLED",
            "Type": "SCHEDULED",
            "Name": "t2",
            "Actions": [
                {
                    "JobName": "j2"
                }
            ],
        }
    ]
}

```

```

        "Schedule": "cron(0 0/1 * * ? *)"
    },
    {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j1"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
]
}

```

The AWS Glue administrator attached a tag value Tom (`glue:resourceTag/Name`: "Tom") to trigger t2. The AWS Glue administrator also gave Tom an IAM policy with a condition statement based on the tag. As a result, Tom can only use an AWS Glue operation that acts on resources with the tag value Tom.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "glue:resourceTag/Name": "Tom"
                }
            }
        }
    ]
}

```

When Tom tries to access the trigger t1, he receives an access denied message. Meanwhile, he can successfully retrieve trigger t2.

```

aws glue get-trigger --name t1

An error occurred (AccessDeniedException) when calling the GetTrigger operation:
User: Tom is not authorized to perform: glue:GetTrigger on resource: arn:aws:glue:us-east-1:123456789012:trigger/t1

aws glue get-trigger --name t2
{
    "Trigger": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j1"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

```

Tom can't use the plural `GetTriggers` API to list triggers because this API operation doesn't support filtering on tags.

To give Tom access to `GetTriggers`, the AWS Glue administrator creates a policy that splits the permissions into two sections. One section allows Tom access to all triggers with the `GetTriggers` API operation. The second section allows Tom access to API operations that are tagged with the value `Tom`. With this policy, Tom is allowed both `GetTriggers` and `GetTrigger` access to trigger `t2`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:GetTriggers",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "glue:/*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "glue:resourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

Example Access Control Using Tags with Deny

Another approach to write a resource policy is to explicitly deny access to resources instead of granting access to a user. For example, if Tom is considered a special user of a team, the administrator can deny access to Tom's resources to Sam and everyone else on the team. As a result, Sam can access almost every resource except those tagged with the tag value `Tom`. Here is the resource policy that AWS Glue administrator grants to Sam. In the first section of the policy, all AWS Glue API operations are allowed for all resources. However, in the second section, those resources tagged with `Tom` are denied access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:/*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "glue:/*"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "glue:resourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

Using the same triggers as the previous example, Sam can access trigger `t1`, but not trigger `t2`. The following example shows the results when Sam tries to access `t1` and `t2`.

```
aws glue get-trigger --name t1
{
    "Trigger": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t1",
        "Actions": [
            {
                "JobName": "j1"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

aws glue get-trigger --name t2

An error occurred (AccessDeniedException) when calling the GetTrigger operation:
User: Sam is not authorized to perform: glue:GetTrigger on resource: arn:aws:glue:us-
east-1:123456789012:trigger/t2 with an explicit deny
```

Important

An explicit denial policy does not work for plural APIs. Even with the attachment of tag value `Tom` to trigger `t2`, Sam can still call `GetTriggers` to view trigger `t2`. Because of this, the administrator might not want to allow access to the `GetTriggers` API operations. The following example shows the results when Sam runs the `GetTriggers` API.

```
aws glue get-triggers
{
    "Triggers": [
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t1",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        },
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t2",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        }
    ]
}
```

Example Access Control Using Tags with List and Batch API Operations

A third approach to writing a resource policy is to allow access to resources using a `List` API operation to list out resources for a tag value. Then, use the corresponding `Batch` API operation to allow access to details of specific resources. With this approach, the administrator doesn't need to allow access to the plural `GetCrawlers`, `GetDevEndpoints`, `GetJobs`, or `GetTriggers` API operations. Instead, you can allow the ability to list the resources with the following API operations:

- `ListCrawlers`
- `ListDevEndpoints`
- `ListJobs`
- `ListTriggers`

And, you can allow the ability to get details about individual resources with the following API operations:

- `BatchGetCrawlers`
- `BatchGetDevEndpoints`
- `BatchGetJobs`
- `BatchGetTriggers`

As an administrator, to use this approach, you can do the following:

1. Add tags to your crawlers, development endpoints, jobs, and triggers.
2. Deny user access to Get API operations such as `GetCrawlers`, `GetDevEndpoints`, `GetJobs`, and `GetTriggers`.
3. To enable users to find out which tagged resources they have access to, allow user access to List API operations such as `ListCrawlers`, `ListDevEndpoints`, `ListJobs`, and `ListTriggers`.
4. Deny user access to AWS Glue tagging APIs, such as `TagResource` and `UntagResource`.
5. Allow user access to resource details with BatchGet API operations such as `BatchGetCrawlers`, `BatchGetDevEndpoints`, `BatchGetJobs`, and `BatchGetTriggers`.

For example, when calling the `ListCrawlers` operation, provide a tag value to match the user name. Then the result is a list of crawlers that match the provided tag values. Provide the list of names to the `BatchGetCrawlers` to get details about each crawler with the given tag.

For example, if Tom should only be able to retrieve details of triggers that are tagged with Tom, the administrator can add tags to triggers for Tom, deny access to the `GetTriggers` API operation to all users and allow access to all users to `ListTriggers` and `BatchGetTriggers`. The following is the resource policy that the AWS Glue administrator grants to Tom. In the first section of the policy, AWS Glue API operations are denied for `GetTriggers`. In the second section of the policy, `ListTriggers` is allowed for all resources. However, in the third section, those resources tagged with Tom are allowed access with the `BatchGetTriggers` access.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": "glue:GetTriggers",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue>ListTriggers"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue:BatchGetTriggers"  
            ]  
        }  
    ]  
}
```

```

        "glue:BatchGetTriggers"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "glue:resourceTag/Name": "Tom"
        }
    }
}
]
```

Using the same triggers as the previous example, Tom can access trigger t2, but not trigger t1. The following example shows the results when Tom tries to access t1 and t2 with BatchGetTriggers.

```

aws glue batch-get-triggers --trigger-names t2
{
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

aws glue batch-get-triggers --trigger-names t1

An error occurred (AccessDeniedException) when calling the BatchGetTriggers operation: No
access to any requested resource.

```

The following example shows the results when Tom tries to access both trigger t2 and trigger t3 (which does not exist) in the same BatchGetTriggers call. Notice that because Tom has access to trigger t2 and it exists, only t2 is returned. Although Tom is allowed to access trigger t3, trigger t3 does not exist, so t3 is returned in the response in a list of "TriggersNotFound": [].

```

aws glue batch-get-triggers --trigger-names t2 t3
{
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
            }
        ],
        "TriggersNotFound": ["t3"],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

```

AWS Glue Resource-Based Access Control Policy Examples

This section contains example resource policies, including policies that grant cross-account access.

Important

By changing an AWS Glue resource policy, you might accidentally revoke permissions for existing AWS Glue users in your account and cause unexpected disruptions. Try these examples only in development or test accounts, and ensure that they don't break any existing workflows before you make the changes.

Note

Both IAM policies and an AWS Glue resource policy take a few seconds to propagate. After you attach a new policy, you might notice that the old policy is still in effect until the new policy has propagated through the system.

The following examples use the AWS Command Line Interface (AWS CLI) to interact with AWS Glue service APIs. You can perform the same operations on the AWS Glue console or using one of the AWS SDKs.

To set up the AWS CLI

1. Install the AWS CLI by following the instructions in [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.
2. Configure the AWS CLI by following the instructions in [Configuration and Credential Files](#). Create an admin profile using your AWS account administrator credentials. Configure the default AWS Region to us-west-2 (or a Region that you use), and set the default output format to **JSON**.
3. Test access to the AWS Glue API by running the following command (replacing **Alice** with a real IAM user or role in your account).

```
# Run as admin of account account-id
$ aws glue put-resource-policy --profile administrator-name --region us-west-2 --
policy-in-json '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:user/Alice"
                ]
            },
            "Effect": "Allow",
            "Action": [
                "glue:*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:account-id:catalog"
            ]
        }
    ]
}'
```

4. Configure a user profile for each IAM user in the accounts that you use for testing your resource policy and cross-account access.

Example 1. Use a Resource Policy to Control Access in the Same Account

In this example, an admin user in Account A creates a resource policy that grants IAM user Alice in Account A full access to the catalog. Alice has no IAM policy attached.

To do this, the administrator user runs the following AWS CLI command.

```
# Run as admin of Account A
```

```
$ aws glue put-resource-policy --profile administrator-name --region us-west-2 --policy-in-json '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-A-id:user/Alice"
                ]
            },
            "Effect": "Allow",
            "Action": [
                "glue:*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:account-A-id:*"
            ]
        }
    ]
}'
```

Instead of entering the JSON policy document as a part of your AWS CLI command, you can save a policy document in a file and reference the file path in the AWS CLI command, prefixed by `file://`. The following is an example of how you might do that.

```
$ echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-A-id:user/Alice"
                ]
            },
            "Effect": "Allow",
            "Action": [
                "glue:*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:account-A-id:*"
            ]
        }
    ]
}' > /temp/policy.json

$ aws glue put-resource-policy --profile admin1 \
    --region us-west-2 --policy-in-json file:///temp/policy.json
```

After this resource policy has propagated, Alice can access all AWS Glue resources in Account A, as follows.

```
# Run as user Alice
$ aws glue create-database --profile alice --region us-west-2 --database-input '{
    "Name": "new_database",
    "Description": "A new database created by Alice",
    "LocationUri": "s3://my-bucket"
}'

$ aws glue get-table --profile alice --region us-west-2 --database-name "default" --table-name "tbl1"
```

In response to Alice's `get-table` call, the AWS Glue service returns the following.

```
{
  "Table": {
    "Name": "tbl1",
    "PartitionKeys": [],
    "StorageDescriptor": {
      .....
    },
    .....
  }
}
```

Example 2. Use a Resource Policy to Grant Cross-Account Access

In this example, a resource policy in Account A is used to grant to Bob in Account B read-only access to all Data Catalog resources in Account A. To do this, four steps are needed:

- Verify that Account A has migrated its Amazon Athena data catalog to AWS Glue.**

Cross-account access to AWS Glue is not allowed if the resource-owner account has not migrated its Athena data catalog to AWS Glue. For more details on how to migrate the Athena catalog to AWS Glue, see [Upgrading to the AWS Glue Data Catalog Step-by-Step](#) in the *Amazon Athena User Guide*.

```
# Verify that the value "ImportCompleted" is true. This value is region specific.
$ aws glue get-catalog-import-status --profile admin1 --region us-west-2
{
  "ImportStatus": {
    "ImportCompleted": true,
    "ImportTime": 1502512345.0,
    "ImportedBy": "StatusSetByDefault"
  }
}
```

- An admin in Account A creates a resource policy granting Account B access.**

```
# Run as admin of Account A
$ aws glue put-resource-policy --profile admin1 --region us-west-2 --policy-in-json '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-B-id:root"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:Get*",
        "glue:BatchGet*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:/*"
      ]
    }
  ]
}'
```

- An admin in Account B grants Bob access to Account A using an IAM policy.**

```
# Run as admin of Account B
$ aws iam put-user-policy --profile admin2 --user-name Bob --policy-name
CrossAccountReadOnly --policy-document '{
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:Get*",
                "glue:BatchGet*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:account-A-id:*"
            ]
        }
    ]
}

```

4. Verify Bob's access to a resource in Account A.

```

# Run as user Bob. This call succeeds in listing Account A databases.
$ aws glue get-databases --profile bob --region us-west-2 --catalog-id account-A-id
{
    "DatabaseList": [
        {
            "Name": "db1"
        },
        {
            "Name": "db2"
        },
        .....
    ]
}

# This call succeeds in listing tables in the 'default' Account A database.
$ aws glue get-table --profile alice --region us-west-2 --catalog-id account-A-id \
--database-name "default" --table-name "tbl1"
{
    "Table": {
        "Name": "tbl1",
        "PartitionKeys": [],
        "StorageDescriptor": {
            .....
        },
        .....
    }
}

# This call fails with access denied, because Bob has only been granted read access.
$ aws glue create-database --profile bob --region us-west-2 --catalog-id account-A-id
--database-input '{
    "Name": "new_database2",
    "Description": "A new database created by Bob",
    "LocationUri": "s3://my-bucket2"
}'
An error occurred (AccessDeniedException) when calling the CreateDatabase operation:
User: arn:aws:iam:account-B-id:user/Bob is not authorized to perform:
glue>CreateDatabase on resource: arn:aws:glue:us-west-2:account-A-id:database/
new_database2

```

In step 2, the administrator in Account A grants permission to the root user of Account B. The root user can then delegate the permissions it owns to all IAM principals (users, roles, groups, and so forth) by attaching IAM policies to them. Because an admin user already has a full-access IAM policy attached, an administrator automatically owns the permissions granted to the root user, and also the permission to delegate permissions to other IAM users in the account.

Alternatively, in step 2, you could grant permission to the Amazon Resource Name (ARN) of user Bob directly. This restricts the cross-account access permission to Bob alone. However, step 3 is still required for Bob to actually gain the cross-account access. For cross-account access, *both* the resource policy in the resource account *and* an IAM policy in the user's account are required for access to work. This is different from the same-account access in Example 1, where either the resource policy or the IAM policy can grant access without needing the other.

AWS Glue API Permissions: Actions and Resources Reference

Use the following list as a reference when you're setting up [Identity and Access Management in AWS Glue \(p. 50\)](#) and writing a permissions policy to attach to an IAM identity (identity-based policy) or to a resource (resource policy). The list includes each AWS Glue API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's Action field, and you specify the resource value in the policy's Resource field.

Actions on some AWS Glue resources require that ancestor and child resource ARNs are also included in the policy's Resource field. For more information, see [Data Catalog ARNs \(p. 65\)](#).

Generally, you can replace ARN segments with wildcards. For more information, see [IAM JSON Policy Elements](#) in the *IAM User Guide*.

Condition keys for IAM policies are listed by API operation. You can use AWS-wide condition keys in your AWS Glue policies to express conditions. For a complete list of AWS-wide keys, see [AWS Global Condition Keys](#) in the *IAM User Guide*.

Note

To specify an action, use the `glue:` prefix followed by the API operation name (for example, `glue:GetTable`).

AWS Glue API Permissions: Actions and Resources Reference

[BatchCreatePartition \(batch_create_partition\) \(p. 492\)](#)

Action(s): `glue:BatchCreatePartition`

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[BatchDeleteConnection \(batch_delete_connection\) \(p. 505\)](#)

Action(s): `glue:BatchDeleteConnection`

Resource:

```
arn:aws:glue:region:account-id:connection/connection-name
arn:aws:glue:region:account-id:catalog
```

Note

All the connection deletions to be performed by the call must be authorized by IAM. If any of these deletions is not authorized, the call fails and no connections are deleted.

[BatchDeletePartition \(batch_delete_partition\) \(p. 494\)](#)

Action(s): glue:BatchDeletePartition

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

Note

All the partition deletions to be performed by the call must be authorized by IAM. If any of these deletions is not authorized, the call fails and no partitions are deleted.

[BatchDeleteTable \(batch_delete_table\) \(p. 481\)](#)

Action(s): glue:BatchDeleteTable

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

Note

All the table deletions to be performed by the call must be authorized by IAM. If any of these deletions is not authorized, the call fails and no tables are deleted.

[BatchDeleteTableVersion \(batch_delete_table_version\) \(p. 486\)](#)

Action(s): glue:BatchDeleteTableVersion

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[BatchGetCrawlers \(batch_get_crawlers\) \(p. 532\)](#)

Action(s): glue:BatchGetCrawlers

Resource:

```
arn:aws:glue:region:account-id:crawler/crawler-name
```

Condition keys: glue:resourceTag

[BatchGetDevEndpoints \(batch_get_dev_endpoints\) \(p. 590\)](#)

Action(s): glue:BatchGetDevEndpoints

Resource:

```
arn:aws:glue:region:account-id:devEndpoint/development-endpoint-name
```

Condition keys: glue:resourceTag

[BatchGetJobs \(batch_get_jobs\) \(p. 550\)](#)

Action(s): glue:BatchGetJobs

Resource:

```
arn:aws:glue:region:account-id:job/job-name
```

Condition keys: glue:resourceTag

[BatchGetPartition \(batch_get_partition\) \(p. 498\)](#)

Action(s): glue:BatchGetPartition

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name  
arn:aws:glue:region:account-id:database/database-name  
arn:aws:glue:region:account-id:catalog
```

[BatchGetTriggers \(batch_get_triggers\) \(p. 569\)](#)

Action(s): glue:BatchGetTriggers

Resource:

```
arn:aws:glue:region:account-id:trigger/trigger-name
```

Condition keys: glue:resourceTag

[BatchStopJobRun \(batch_stop_job_run\) \(p. 557\)](#)

Action(s): glue:BatchStopJobRun

Resource:

*

[CreateClassifier \(create_classifier\) \(p. 519\)](#)

Action(s): glue>CreateClassifier

Resource:

*

[CreateConnection \(create_connection\) \(p. 502\)](#)

Action(s): glue>CreateConnection

Resource:

```
arn:aws:glue:region:account-id:connection/connection-name
arn:aws:glue:region:account-id:catalog
```

[CreateCrawler \(create_crawler\) \(p. 527\)](#)

Action(s): glue:CreateCrawler

Resource:

```
arn:aws:glue:region:account-id:crawler/crawler-name
```

or

```
arn:aws:glue:region:account-id:crawler/*
```

Condition keys: awss:RequestTag

[CreateDatabase \(create_database\) \(p. 470\)](#)

Action(s): glue>CreateDatabase

Resource:

```
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[CreateDevEndpoint \(create_dev_endpoint\) \(p. 583\)](#)

Action(s): glue>CreateDevEndpoint

Resource:

```
arn:aws:glue:region:account-id:devEndpoint/development-endpoint-name
```

or

```
arn:aws:glue:region:account-id:devEndpoint/*
```

Condition keys: aws:RequestTag

[CreateJob \(create_job\) \(p. 545\)](#)

Action(s): glue>CreateJob

Resource:

```
arn:aws:glue:region:account-id:job/job-name
```

or

```
arn:aws:glue:region:account-id:job/*
```

Condition keys: aws:RequestTag

[CreatePartition \(create_partition\) \(p. 491\)](#)

Action(s): glue>CreatePartition

Resource:

```
arn:aws:glue:region:account-id:table/database-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[CreateScript \(create_script\) \(p. 537\)](#)

Action(s): glue:CreateScript

Resource:

*

[CreateSecurityConfiguration \(create_security_configuration\) \(p. 465\)](#)

Action(s): glue>CreateSecurityConfiguration

Resource:

*

[CreateTable \(create_table\) \(p. 479\)](#)

Action(s): glue:CreateTable

Resource:

```
arn:aws:glue:region:account-id:table/database-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[CreateTrigger \(create_trigger\) \(p. 564\)](#)

Action(s): glue>CreateTrigger

Resource:

arn:aws:glue:**region:account-id**:trigger/**trigger-name**

or

arn:aws:glue:**region:account-id**:trigger/*

Condition keys: aws:RequestTag

[CreateUserDefinedFunction \(create_user_defined_function\) \(p. 507\)](#)

Action(s): glue>CreateUserDefinedFunction

Resource:

```
arn:aws:glue:region:account-id:userDefinedFunction/database-name/user-defined-function-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[DeleteClassifier \(delete_classifier\) \(p. 520\)](#)

Action(s): glue:DeleteClassifier

Resource:

*

[DeleteConnection \(delete_connection\) \(p. 503\)](#)

Action(s): glue:DeleteConnection

Resource:

```
arn:aws:glue:region:account-id:connection/connection-name
arn:aws:glue:region:account-id:catalog
```

[DeleteCrawler \(delete_crawler\) \(p. 528\)](#)

Action(s): glue:DeleteCrawler

Resource:

arn:aws:glue:**region:account-id**:crawler/**crawler-name**

or

arn:aws:glue:**region:account-id**:crawler/*

Condition keys: glue:resourceTag

[DeleteDatabase \(delete_database\) \(p. 471\)](#)

Action(s): glue:DeleteDatabase

Resource:

```
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:userDefinedFunction/database-name/*
arn:aws:glue:region:account-id:table/database-name/*
arn:aws:glue:region:account-id:catalog
```

[DeleteDevEndpoint \(delete_dev_endpoint\) \(p. 588\)](#)

Action(s): glue:DeleteDevEndpoint

Resource:

arn:aws:glue:**region:account-id**:devEndpoint/**development-endpoint-name**

or

arn:aws:glue:**region:account-id**:devEndpoint/*

Condition keys: glue:resourceTag

[DeleteJob \(delete_job\) \(p. 549\)](#)

Action(s): glue:DeleteJob

Resource:

`arn:aws:glue:region:account-id:job/job-name`

or

`arn:aws:glue:region:account-id:job/*`

Condition keys: `glue:resourceTag`

[DeletePartition \(delete_partition\) \(p. 493\)](#)

Action(s): `glue:UpdatePartition`

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[DeleteResourcePolicy \(delete_resource_policy\) \(p. 465\)](#)

Action(s): `glue:DeleteResourcePolicy`

Resource:

*

[DeleteSecurityConfiguration \(delete_security_configuration\) \(p. 466\)](#)

Action(s): `glue:DeleteSecurityConfiguration`

Resource:

*

[DeleteTable \(delete_table\) \(p. 481\)](#)

Action(s): `glue:DeleteTable`

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[DeleteTableVersion \(delete_table_version\) \(p. 485\)](#)

Action(s): `glue:DeleteTableVersion`

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[DeleteTrigger \(delete_trigger\) \(p. 568\)](#)

Action(s): glue:DeleteTrigger

Resource:

arn:aws:glue:**region:account-id:trigger/trigger-name**

or

arn:aws:glue:**region:account-id:trigger/***

Condition keys: glue:resourceTag

[DeleteUserDefinedFunction \(delete_user_defined_function\) \(p. 509\)](#)

Action(s): glue:DeleteUserDefinedFunction

Resource:

```
arn:aws:glue:region:account-id:userDefinedFunction/database-name/user-defined-function-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetCatalogImportStatus \(get_catalog_import_status\) \(p. 512\)](#)

Action(s): glue:GetCatalogImportStatus

Resource:

```
arn:aws:glue:region:account-id:catalog
```

[GetClassifier \(get_classifier\) \(p. 520\)](#)

Action(s): glue:GetClassifier

Resource:

*

[GetClassifiers \(get_classifiers\) \(p. 521\)](#)

Action(s): glue:GetClassifiers

Resource:

*

[GetConnection \(get_connection\) \(p. 503\)](#)

Action(s): glue:GetConnection

Resource:

```
arn:aws:glue:region:account-id:connection/connection-name
arn:aws:glue:region:account-id:catalog
```

[GetConnections \(get_connections\) \(p. 504\)](#)

Action(s): glue:GetConnections

```
arn:aws:glue:region:account-id:connection/connection-names
arn:aws:glue:region:account-id:catalog
```

[GetCrawler \(get_crawler\) \(p. 528\)](#)

Action(s): glue:GetCrawler

Resource:

```
arn:aws:glue:region:account-id:crawler/crawler-name
```

or

```
arn:aws:glue:region:account-id:crawler/*
```

[GetCrawlerMetrics \(get_crawler_metrics\) \(p. 529\)](#)

Action(s): glue:GetCrawlerMetrics

Resource:

*

[GetCrawlers \(get_crawlers\) \(p. 529\)](#)

Action(s): glue:GetCrawlers

Resource:

*

Condition keys: glue:resourceTag

[GetDatabase \(get_database\) \(p. 471\)](#)

Action(s): glue:GetDatabase

Resource:

```
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetDatabases \(get_databases\) \(p. 472\)](#)

Action(s): glue:GetDatabases

Resource:

```
arn:aws:glue:region:account-id:database/database-names
arn:aws:glue:region:account-id:catalog
```

[GetDataCatalogEncryptionSettings \(get_data_catalog_encryption_settings\) \(p. 462\)](#)

Action(s): glue:GetDataCatalogEncryptionSettings

Resource:

*

[GetDataflowGraph \(get_dataflow_graph\) \(p. 538\)](#)

Action(s): glue:GetDataflowGraph

Resource:

*

[GetDevEndpoint \(get_dev_endpoint\) \(p. 588\)](#)

Action(s): glue:GetDevEndpoint

Resource:

arn:aws:glue:*region:account-id*:devEndpoint/*development-endpoint-name*

or

arn:aws:glue:*region:account-id*:devEndpoint/*

Condition keys: glue:resourceTag

[GetDevEndpoints \(get_dev_endpoints\) \(p. 589\)](#)

Action(s): glue:GetDevEndpoints

Resource:

*

[GetJob \(get_job\) \(p. 548\)](#)

Action(s): glue:GetJob

Resource:

arn:aws:glue:*region:account-id*:job/*job-name*

or

arn:aws:glue:*region:account-id*:job/*

Condition keys: glue:resourceTag

[GetJobRun \(get_job_run\) \(p. 558\)](#)

Action(s): glue:GetJobRun

Resource:

*

[GetJobRuns \(get_job_runs\) \(p. 558\)](#)

Action(s): glue:GetJobRuns

Resource:

*

[GetJobs \(get_jobs\) \(p. 549\)](#)

Action(s): glue:GetJobs

Resource:

*

[GetMapping \(get_mapping\) \(p. 539\)](#)

Action(s): glue:GetMapping

Resource:

*

[GetPartition \(get_partition\) \(p. 495\)](#)

Action(s): glue:GetPartition

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetPartitions \(get_partitions\) \(p. 495\)](#)

Action(s): glue:GetPartitions

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetPlan \(get_plan\) \(p. 539\)](#)

Action(s): glue:GetPlan

Resource:

*

[GetResourcePolicy \(get_resource_policy\) \(p. 464\)](#)

Action(s): glue:GetResourcePolicy

Resource:

*

[GetSecurityConfiguration \(get_security_configuration\) \(p. 466\)](#)

Action(s): glue:GetSecurityConfiguration

Resource:

*

[GetSecurityConfigurations \(get_security_configurations\) \(p. 467\)](#)

Action(s): glue:GetSecurityConfigurations

Resource:

*

[GetTable \(get_table\) \(p. 482\)](#)

Action(s): glue:GetTable

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetTables \(get_tables\) \(p. 483\)](#)

Action(s): glue:GetTables

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-names
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetTableVersion \(get_table_version\) \(p. 484\)](#)

Action(s): glue:GetTableVersion

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetTableVersions \(get_table_versions\) \(p. 484\)](#)

Action(s): glue:GetTableVersions

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetTags \(get_tags\) \(p. 616\)](#)

Action(s): glue:GetTags

Resource:

*

[GetTrigger \(get_trigger\) \(p. 566\)](#)

Action(s): glue:GetTrigger

Resource:

arn:aws:glue:**region**:**account-id**:trigger/**trigger-name**

or

arn:aws:glue:**region**:**account-id**:trigger/*

Condition keys: glue:resourceTag

[GetTriggers \(get_triggers\) \(p. 566\)](#)

Action(s): glue:GetTriggers

Resource:

*

[GetUserDefinedFunction \(get_user_defined_function\) \(p. 509\)](#)

Action(s): glue:GetUserDefinedFunction

Resource:

```
arn:aws:glue:region:account-id:userDefinedFunction/database-name/user-defined-function-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[GetUserDefinedFunctions \(get_user_defined_functions\) \(p. 510\)](#)

Action(s): glue:GetUserDefinedFunctions

Resource:

```
arn:aws:glue:region:account-id:userDefinedFunction/database-name/user-defined-function-names
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[ImportCatalogToGlue \(import_catalog_to_glue\) \(p. 511\)](#)

Action(s): glue:ImportCatalogToGlue

Resource:

```
arn:aws:glue:region:account-id:catalog
```

[ListCrawlers \(list_crawlers\) \(p. 532\)](#)

Action(s): glue>ListCrawlers

Resource:

*

[ListDevEndpoints \(list_dev_endpoints\) \(p. 590\)](#)

Action(s): glue:ListDevEndpoints

Resource:

*

[ListJobs \(list_jobs\) \(p. 550\)](#)

Action(s): glue>ListJobs

Resource:

*

[ListTriggers \(list_triggers\) \(p. 568\)](#)

Action(s): glue>ListTriggers

Resource:

*

[PutDataCatalogEncryptionSettings \(put_data_catalog_encryption_settings\) \(p. 463\)](#)

Action(s): glue:PutDataCatalogEncryptionSettings

Resource:

*

[PutResourcePolicy \(put_resource_policy\) \(p. 463\)](#)

Action(s): glue:PutResourcePolicy

Resource:

*

[ResetJobBookmark \(reset_job_bookmark\) \(p. 560\)](#)

Action(s): glue:ResetJobBookmark

Resource:

*

[StartCrawler \(start_crawler\) \(p. 531\)](#)

Action(s): glue:StartCrawler

Resource:

arn:aws:glue:**region**:**account-id**:crawler/**crawler-name**

or

arn:aws:glue:**region**:**account-id**:crawler/*

Condition keys: glue:resourceTag

[StartCrawlerSchedule \(start_crawler_schedule\) \(p. 534\)](#)

Action(s): glue:StartCrawlerSchedule

Resource:

*

[StartJobRun \(start_job_run\) \(p. 555\)](#)

Action(s): glue:StartJobRun

Resource:

*

[StartTrigger \(start_trigger\) \(p. 565\)](#)

Action(s): glue:StartTrigger

Resource:

arn:aws:glue:*region*:*account-id*:trigger/*trigger-name*

or

arn:aws:glue:*region*:*account-id*:trigger/*

Condition keys: glue:resourceTag

[StopCrawler \(stop_crawler\) \(p. 531\)](#)

Action(s): glue:StopCrawler

Resource:

arn:aws:glue:*region*:*account-id*:crawler/*crawler-name*

or

arn:aws:glue:*region*:*account-id*:crawler/*

Condition keys: glue:resourceTag

[StopCrawlerSchedule \(stop_crawler_schedule\) \(p. 534\)](#)

Action(s): glue:StopCrawlerSchedule

Resource:

*

[StopTrigger \(stop_trigger\) \(p. 567\)](#)

Action(s): glue:StopTrigger

Resource:

arn:aws:glue:*region*:*account-id*:trigger/*trigger-name*

or

arn:aws:glue:*region*:*account-id*:trigger/*

Condition keys: glue:resourceTag

[TagResource \(tag_resource\) \(p. 614\)](#)

Action(s): glue:TagResource

Resource:

*

Condition keys: aws:RequestTag
[UntagResource \(untag_resource\) \(p. 615\)](#)

Action(s): glue:UntagResource

Resource:

*

Condition keys: aws:RequestTag
[UpdateClassifier \(update_classifier\) \(p. 521\)](#)

Action(s): glue:UpdateClassifier

Resource:

*

[UpdateConnection \(update_connection\) \(p. 505\)](#)

Action(s): glue:UpdateConnection

Resource:

```
arn:aws:glue:region:account-id:connection/connection-name
arn:aws:glue:region:account-id:catalog
```

[UpdateCrawler \(update_crawler\) \(p. 530\)](#)

Action(s): glue:UpdateCrawler

Resource:

```
arn:aws:glue:region:account-id:crawler/crawler-name
```

or

```
arn:aws:glue:region:account-id:crawler/*
```

Condition keys: glue:resourceTag

[UpdateCrawlerSchedule \(update_crawler_schedule\) \(p. 533\)](#)

Action(s): glue:UpdateCrawlerSchedule

Resource:

*

[UpdateDatabase \(update_database\) \(p. 470\)](#)

Action(s): glue:UpdateDatabase

Resource:

```
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[UpdateDevEndpoint \(update_dev_endpoint\) \(p. 587\)](#)

Action(s): glue:UpdateDevEndpoint

Resource:

arn:aws:glue:*region*:*account-id*:devEndpoint/*development-endpoint-name*

or

arn:aws:glue:*region*:*account-id*:devEndpoint/*

Condition keys: glue:resourceTag

[UpdateJob \(update_job\) \(p. 548\)](#)

Action(s): glue:UpdateJob

Resource:

arn:aws:glue:*region*:*account-id*:job/*job-name*

or

arn:aws:glue:*region*:*account-id*:job/*

Condition keys: glue:resourceTag

[UpdatePartition \(update_partition\) \(p. 493\)](#)

Action(s): glue:UpdatePartition

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[UpdateTable \(update_table\) \(p. 480\)](#)

Action(s): glue:UpdateTable

Resource:

```
arn:aws:glue:region:account-id:table/database-name/table-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

[UpdateTrigger \(update_trigger\) \(p. 567\)](#)

Action(s): glue:UpdateTrigger

Resource:

arn:aws:glue:*region*:*account-id*:trigger/*trigger-name*

or

arn:aws:glue:*region*:*account-id*:trigger/*

Condition keys: `glue:resourceTag`

[UpdateUserDefinedFunction \(update_user_defined_function\) \(p. 508\)](#)

Action(s): `glue:UpdateUserDefinedFunction`

Resource:

```
arn:aws:glue:region:account-id:userDefinedFunction/database-name/user-defined-function-name
arn:aws:glue:region:account-id:database/database-name
arn:aws:glue:region:account-id:catalog
```

Related Topics

- [Identity and Access Management \(p. 50\)](#)

Logging and Monitoring in AWS Glue

You can automate the running of your ETL (extract, transform, and load) jobs. AWS Glue provides metrics for crawlers and jobs that you can monitor. After you set up the AWS Glue Data Catalog with the required metadata, AWS Glue provides statistics about the health of your environment. You can automate the invocation of crawlers and jobs with a time-based schedule based on cron. You can also trigger jobs when an event-based trigger fires.

AWS Glue is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or AWS service in AWS Glue. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, Amazon CloudWatch Logs, and Amazon CloudWatch Events. Every event or log entry contains information about who generated the request.

Use Amazon CloudWatch Events to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near-real time. You can write simple rules to indicate which events are of interest and what automated actions to take when an event matches a rule.

For more information, see [Automating AWS Glue with CloudWatch Events \(p. 231\)](#).

Compliance Validation for AWS Glue

Third-party auditors assess the security and compliance of AWS Glue as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, 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](#) in the *AWS Artifact User Guide*.

Your compliance responsibility when using AWS Glue is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of AWS Glue is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- [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 AWS Glue

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, AWS Glue offers several features to help support your data resiliency and backup needs.

Infrastructure Security in AWS Glue

As a managed service, AWS Glue 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 AWS Glue through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Topics

- [Using AWS Glue with VPC Endpoints \(p. 102\)](#)
- [Shared Amazon VPCs \(p. 103\)](#)

Using AWS Glue with VPC Endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and AWS Glue. You use this connection to enable AWS Glue to communicate with the resources in your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets,

route tables, and network gateways. To connect your VPC to AWS Glue, you define an interface VPC endpoint for AWS Glue. When you use a VPC interface endpoint, communication between your VPC and AWS Glue is conducted entirely and securely within the AWS network.

You can use AWS Glue with VPC endpoints in all AWS Regions that support both AWS Glue and Amazon VPC endpoints.

For more information, see these topics in the *Amazon VPC User Guide*:

- [What Is Amazon VPC?](#)
- [Creating an Interface Endpoint](#)

Shared Amazon VPCs

AWS Glue supports shared Amazon virtual private clouds (VPCs). Amazon VPC sharing allows multiple AWS accounts to create their application resources, such as Amazon EC2 instances and Amazon Relational Database Service (RDS) databases, into shared, centrally-managed Amazon VPCs. In this model, the account that owns the VPC (owner) shares one or more subnets with other accounts (participants) that belong to the same organization from AWS Organizations. After a subnet is shared, the participants can view, create, modify, and delete their application resources in the subnets that are shared with them.

In AWS Glue, to create a connection with a shared subnet, you must create a security group within your account and attach the security group to the shared subnet.

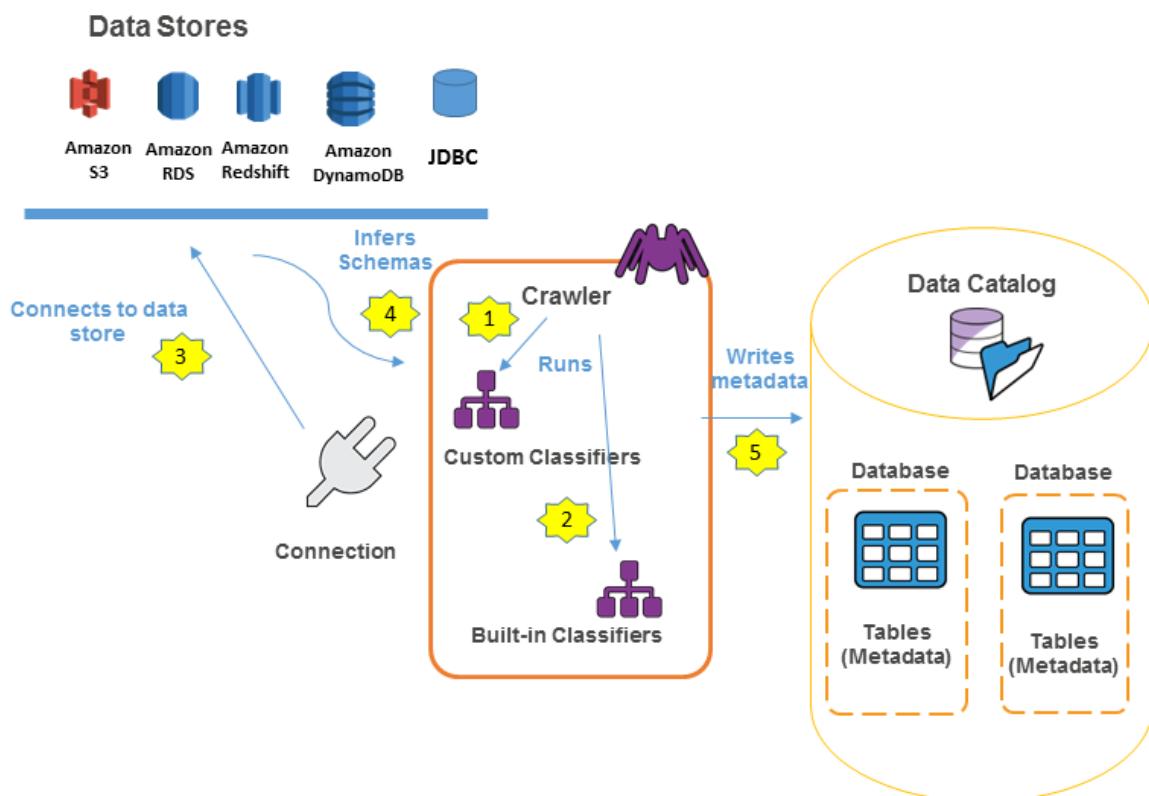
For more information, see these topics:

- [Working with Shared VPCs](#) in the *Amazon VPC User Guide*
- [What Is AWS Organizations?](#) in the *AWS Organizations User Guide*

Populating the AWS Glue Data Catalog

The AWS Glue Data Catalog contains references to data that is used as sources and targets of your extract, transform, and load (ETL) jobs in AWS Glue. To create your data warehouse, you must catalog this data. The AWS Glue Data Catalog is an index to the location, schema, and runtime metrics of your data. You use the information in the Data Catalog to create and monitor your ETL jobs. Information in the Data Catalog is stored as metadata tables, where each table specifies a single data store. Typically, you run a crawler to take inventory of the data in your data stores, but there are other ways to add metadata tables into your Data Catalog. For more information, see [Defining Tables in the AWS Glue Data Catalog \(p. 106\)](#).

The following workflow diagram shows how AWS Glue crawlers interact with data stores and other elements to populate the Data Catalog.



The following is the general workflow for how a crawler populates the AWS Glue Data Catalog:

1. A crawler runs any *custom classifiers* that you choose to infer the format and schema of your data. You provide the code for custom classifiers, and they run in the order that you specify.

The first custom classifier to successfully recognize the structure of your data is used to create a schema. Custom classifiers lower in the list are skipped. If no custom classifier matches your data's

schema, built-in classifiers try to recognize your data's schema. An example of a built-in classifier is one that recognizes JSON.

2. The crawler connects to the data store. Some data stores require connection properties for crawler access.
3. The inferred schema is created for your data.
4. The crawler writes metadata to the Data Catalog. A table definition contains metadata about the data in your data store. The table is written to a database, which is a container of tables in the Data Catalog. Attributes of a table include classification, which is a label created by the classifier that inferred the table schema.

Topics

- [Defining a Database in Your Data Catalog \(p. 105\)](#)
- [Defining Tables in the AWS Glue Data Catalog \(p. 106\)](#)
- [Adding a Connection to Your Data Store \(p. 110\)](#)
- [Defining Crawlers \(p. 116\)](#)
- [Adding Classifiers to a Crawler \(p. 129\)](#)
- [Working with Data Catalog Settings on the AWS Glue Console \(p. 144\)](#)
- [Updating the Data Catalog with New Partitions \(p. 145\)](#)
- [Populating the Data Catalog Using AWS CloudFormation Templates \(p. 146\)](#)

Defining a Database in Your Data Catalog

When you define a table in the AWS Glue Data Catalog, you add it to a database. A database is used to organize tables in AWS Glue. You can organize your tables using a crawler or using the AWS Glue console. A table can be in only one database at a time.

Your database can contain tables that define data from many different data stores. This data can include objects in Amazon Simple Storage Service (Amazon S3) and relational tables in Amazon Relational Database Service.

Note

When you delete a database, all the tables in the database are also deleted.

For more information about defining a database using the AWS Glue console, see [Working with Databases on the AWS Glue Console \(p. 105\)](#).

Working with Databases on the AWS Glue Console

A database in the AWS Glue Data Catalog is a container that holds tables. You use databases to organize your tables into separate categories. Databases are created when you run a crawler or add a table manually. The database list in the AWS Glue console displays descriptions for all your databases.

To view the list of databases, sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Choose **Databases**, and then choose a database name in the list to view the details.

From the **Databases** tab in the AWS Glue console, you can add, edit, and delete databases:

- To create a new database, choose **Add database** and provide a name and description. For compatibility with other metadata stores, such as Apache Hive, the name is folded to lowercase characters.

Note

If you plan to access the database from Amazon Athena, then provide a name with only alphanumeric and underscore characters. For more information, see [Athena names](#).

- To edit the description for a database, select the check box next to the database name and choose **Action, Edit database**.
- To delete a database, select the check box next to the database name and choose **Action, Delete database**.
- To display the list of tables contained in the database, select the check box next to the database name and choose **View tables**.

To change the database that a crawler writes to, you must change the crawler definition. For more information, see [Defining Crawlers \(p. 116\)](#).

Defining Tables in the AWS Glue Data Catalog

You can add table definitions to the Data Catalog in the following ways:

- Run a crawler that connects to one or more data stores, determines the data structures, and writes tables into the Data Catalog. The crawler uses built-in or custom classifiers to recognize the structure of the data. You can run your crawler on a schedule. For more information, see [Defining Crawlers \(p. 116\)](#).
- Use the AWS Glue console to manually create a table in the AWS Glue Data Catalog. For more information, see [Working with Tables on the AWS Glue Console \(p. 107\)](#).
- Use the `CreateTable` operation in the [AWS Glue API \(p. 452\)](#) to create a table in the AWS Glue Data Catalog. For more information, see [CreateTable Action \(Python: create_table\) \(p. 479\)](#).
- Use AWS CloudFormation templates. For more information, see [Populating the Data Catalog Using AWS CloudFormation Templates \(p. 146\)](#).
- Migrate an Apache Hive metastore. For more information, see [Migration between the Hive Metastore and the AWS Glue Data Catalog](#) on GitHub.

When you define a table manually using the console or an API, you specify the table schema and the value of a classification field that indicates the type and format of the data in the data source. If a crawler creates the table, the data format and schema are determined by either a built-in classifier or a custom classifier. For more information about creating a table using the AWS Glue console, see [Working with Tables on the AWS Glue Console \(p. 107\)](#).

Table Partitions

An AWS Glue table definition of an Amazon Simple Storage Service (Amazon S3) folder can describe a partitioned table. For example, to improve query performance, a partitioned table might separate monthly data into different files using the name of the month as a key. In AWS Glue, table definitions include the partitioning key of a table. When AWS Glue evaluates the data in Amazon S3 folders to catalog a table, it determines whether an individual table or a partitioned table is added.

All the following conditions must be true for AWS Glue to create a partitioned table for an Amazon S3 folder:

- The schemas of the files are similar, as determined by AWS Glue.
- The data format of the files is the same.
- The compression format of the files is the same.

For example, you might own an Amazon S3 bucket named `my-app-bucket`, where you store both iOS and Android app sales data. The data is partitioned by year, month, and day. The data files for iOS

and Android sales have the same schema, data format, and compression format. In the AWS Glue Data Catalog, the AWS Glue crawler creates one table definition with partitioning keys for year, month, and day.

The following Amazon S3 listing of `my-app-bucket` shows some of the partitions. The `=` symbol is used to assign partition key values.

```
my-app-bucket/Sales/year='2010'/month='feb'/day='1'/iOS.csv
my-app-bucket/Sales/year='2010'/month='feb'/day='1'/Android.csv
my-app-bucket/Sales/year='2010'/month='feb'/day='2'/iOS.csv
my-app-bucket/Sales/year='2010'/month='feb'/day='2'/Android.csv
...
my-app-bucket/Sales/year='2017'/month='feb'/day='4'/iOS.csv
my-app-bucket/Sales/year='2017'/month='feb'/day='4'/Android.csv
```

Updating Manually Created Data Catalog Tables Using Crawlers

You might want to create AWS Glue Data Catalog tables manually and then keep them updated with AWS Glue crawlers. Crawlers running on a schedule can add new partitions and update the tables with any schema changes. This also applies to tables migrated from an Apache Hive metastore.

To do this, when you define a crawler, instead of specifying one or more data stores as the source of a crawl, you specify one or more existing Data Catalog tables. The crawler then crawls the data stores specified by the catalog tables. In this case, no new tables are created; instead, your manually created tables are updated.

The following are other reasons why you might want to manually create catalog tables and specify catalog tables as the crawler source:

- You want to choose the catalog table name and not rely on the catalog table naming algorithm.
- You want to prevent new tables from being created in the case where files with a format that could disrupt partition detection are mistakenly saved in the data source path.

For more information, see [Crawler Source Type \(p. 119\)](#).

Working with Tables on the AWS Glue Console

A table in the AWS Glue Data Catalog is the metadata definition that represents the data in a data store. You create tables when you run a crawler, or you can create a table manually in the AWS Glue console. The **Tables** list in the AWS Glue console displays values of your table's metadata. You use table definitions to specify sources and targets when you create ETL (extract, transform, and load) jobs.

To get started, sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Choose the **Tables** tab, and use the **Add tables** button to create tables either with a crawler or by manually typing attributes.

Adding Tables on the Console

To use a crawler to add tables, choose **Add tables**, **Add tables using a crawler**. Then follow the instructions in the **Add crawler** wizard. When the crawler runs, tables are added to the AWS Glue Data Catalog. For more information, see [Defining Crawlers \(p. 116\)](#).

If you know the attributes that are required to create an Amazon Simple Storage Service (Amazon S3) table definition in your Data Catalog, you can create it with the table wizard. Choose **Add tables**, **Add table manually**, and follow the instructions in the **Add table** wizard.

When adding a table manually through the console, consider the following:

- If you plan to access the table from Amazon Athena, then provide a name with only alphanumeric and underscore characters. For more information, see [Athena names](#).
- The location of your source data must be an Amazon S3 path.
- The data format of the data must match one of the listed formats in the wizard. The corresponding classification, SerDe, and other table properties are automatically populated based on the format chosen. You can define tables with the following formats:

JSON

JavaScript Object Notation.

CSV

Character separated values. You also specify the delimiter of either comma, pipe, semicolon, tab, or Ctrl-A.

Parquet

Apache Parquet columnar storage.

Avro

Apache Avro JSON binary format.

XML

Extensible Markup Language format. Specify the XML tag that defines a row in the data. Columns are defined within row tags.

- You can define a partition key for the table.
- Currently, partitioned tables that you create with the console cannot be used in ETL jobs.

Table Attributes

The following are some important attributes of your table:

Table name

The name is determined when the table is created, and you can't change it. You refer to a table name in many AWS Glue operations.

Database

The container object where your table resides. This object contains an organization of your tables that exists within the AWS Glue Data Catalog and might differ from an organization in your data store. When you delete a database, all tables contained in the database are also deleted from the Data Catalog.

Location

The pointer to the location of the data in a data store that this table definition represents.

Classification

A categorization value provided when the table was created. Typically, this is written when a crawler runs and specifies the format of the source data.

Last updated

The time and date (UTC) that this table was updated in the Data Catalog.

Date added

The time and date (UTC) that this table was added to the Data Catalog.

Description

The description of the table. You can write a description to help you understand the contents of the table.

Deprecated

If AWS Glue discovers that a table in the Data Catalog no longer exists in its original data store, it marks the table as deprecated in the data catalog. If you run a job that references a deprecated table, the job might fail. Edit jobs that reference deprecated tables to remove them as sources and targets. We recommend that you delete deprecated tables when they are no longer needed.

Connection

If AWS Glue requires a connection to your data store, the name of the connection is associated with the table.

Viewing and Editing Table Details

To see the details of an existing table, choose the table name in the list, and then choose **Action, View details**.

The table details include properties of your table and its schema. This view displays the schema of the table, including column names in the order defined for the table, data types, and key columns for partitions. If a column is a complex type, you can choose **View properties** to display details of the structure of that field, as shown in the following example:

```
{
  "StorageDescriptor": {
    "cols": [
      {
        "FieldSchema": [
          {
            "name": "primary-1",
            "type": "CHAR",
            "comment": ""
          },
          {
            "name": "second",
            "type": "STRING",
            "comment": ""
          }
        ]
      },
      "location": "s3://aws-logs-111122223333-us-east-1",
      "inputFormat": "",
      "outputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
      "compressed": "false",
      "numBuckets": "0",
      "SerDeInfo": {
        "name": "",
        "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
        "parameters": {
          "separatorChar": "|"
        }
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "SkewedInfo": {},
      "storedAsSubDirectories": "false"
    ]
  }
}
```

```
},
"parameters": {
"classification": "csv"
}
}
```

For more information about the properties of a table, such as `StorageDescriptor`, see [StorageDescriptor Structure \(p. 476\)](#).

To change the schema of a table, choose **Edit schema** to add and remove columns, change column names, and change data types.

To compare different versions of a table, including its schema, choose **Compare versions** to see a side-by-side comparison of two versions of the schema for a table.

To display the files that make up an Amazon S3 partition, choose **View partition**. For Amazon S3 tables, the **Key** column displays the partition keys that are used to partition the table in the source data store. Partitioning is a way to divide a table into related parts based on the values of a key column, such as date, location, or department. For more information about partitions, search the internet for information about "hive partitioning."

Note

To get step-by-step guidance for viewing the details of a table, see the **Explore table** tutorial in the console.

Adding a Connection to Your Data Store

Connections are used by crawlers and jobs in AWS Glue to access certain types of data stores. For information about adding a connection using the AWS Glue console, see [Working with Connections on the AWS Glue Console \(p. 112\)](#).

When Is a Connection Used?

If your data store requires one, the connection is used when you crawl a data store to catalog its metadata in the AWS Glue Data Catalog. The connection is also used by any job that uses the data store as a source or target.

Defining a Connection in the AWS Glue Data Catalog

Some types of data stores require additional connection information to access your data. This information might include an additional user name and password (different from your AWS credentials), or other information that is required to connect to the data store.

After AWS Glue connects to a JDBC data store, it must have permission from the data store to perform operations. The username you provide with the connection must have the required permissions or privileges. For example, a crawler requires SELECT privileges to retrieve metadata from a JDBC data store. Likewise, a job that writes to a JDBC target requires the necessary privileges to INSERT, UPDATE, and DELETE data into an existing table.

AWS Glue can connect to the following data stores by using the JDBC protocol:

- Amazon Redshift
- Amazon Relational Database Service
 - Amazon Aurora
 - Microsoft SQL Server
 - MySQL

- Oracle
- PostgreSQL
- Publicly accessible databases
 - Amazon Aurora
 - MariaDB
 - Microsoft SQL Server
 - MySQL
 - Oracle
 - PostgreSQL

A connection is not typically required for Amazon S3. However, to access Amazon S3 from within your virtual private cloud (VPC), an Amazon S3 VPC endpoint is required. For more information, see [Amazon VPC Endpoints for Amazon S3 \(p. 30\)](#).

In your connection information, you also must consider whether data is accessed through a VPC and then set up network parameters accordingly. AWS Glue requires a private IP for JDBC endpoints. Connections to databases can be over a VPN and DirectConnect as they provide private IP access to on-premises databases.

For information about how to connect to on-premises databases, see the blog post [How to access and analyze on-premises data stores using AWS Glue](#).

Connecting to a JDBC Data Store in a VPC

Typically, you create resources inside Amazon Virtual Private Cloud (Amazon VPC) so that they cannot be accessed over the public internet. By default, resources in a VPC can't be accessed from AWS Glue. To enable AWS Glue to access resources inside your VPC, you must provide additional VPC-specific configuration information that includes VPC subnet IDs and security group IDs. AWS Glue uses this information to set up [elastic network interfaces](#) that enable your function to connect securely to other resources in your private VPC.

Accessing VPC Data Using Elastic Network Interfaces

When AWS Glue connects to a JDBC data store in a VPC, AWS Glue creates an elastic network interface (with the prefix `Glue_`) in your account to access your VPC data. You can't delete this network interface as long as it's attached to AWS Glue. As part of creating the elastic network interface, AWS Glue associates one or more security groups to it. To enable AWS Glue to create the network interface, security groups that are associated with the resource must allow inbound access with a source rule. This rule contains a security group that is associated with the resource. This gives the elastic network interface access to your data store with the same security group.

To allow AWS Glue to communicate with its components, specify a security group with a self-referencing inbound rule for all TCP ports. By creating a self-referencing rule, you can restrict the source to the same security group in the VPC and not open it to all networks. The default security group for your VPC might already have a self-referencing inbound rule for `All Traffic`.

You can create rules in the Amazon VPC console. To update rule settings via the AWS Management Console, navigate to the VPC console (<https://console.aws.amazon.com/vpc/>), and select the appropriate security group. Specify the inbound rule for `All TCP` to have as its source the same security group name. For more information about security group rules, see [Security Groups for Your VPC](#).

Each elastic network interface is assigned a private IP address from the IP address range in the subnets that you specify. The network interface is not assigned any public IP addresses. AWS Glue requires internet access (for example, to access AWS services that don't have VPC endpoints). You can configure a network address translation (NAT) instance inside your VPC, or you can use the Amazon VPC NAT

gateway. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*. You can't directly use an internet gateway attached to your VPC as a route in your subnet route table because that requires the network interface to have public IP addresses.

The VPC network attributes `enableDnsHostnames` and `enableDnsSupport` must be set to true. For more information, see [Using DNS with your VPC](#).

Important

Don't put your data store in a public subnet or in a private subnet that doesn't have internet access. Instead, attach it only to private subnets that have internet access through a NAT instance or an Amazon VPC NAT gateway.

Elastic Network Interface Properties

To create the elastic network interface, you must supply the following properties:

VPC

The name of the VPC that contains your data store.

Subnet

The subnet in the VPC that contains your data store.

Security groups

The security groups that are associated with your data store. AWS Glue associates these security groups with the elastic network interface that is attached to your VPC subnet. To allow AWS Glue components to communicate and also prevent access from other networks, at least one chosen security group must specify a self-referencing inbound rule for all TCP ports.

For information about managing a VPC with Amazon Redshift, see [Managing Clusters in an Amazon Virtual Private Cloud \(VPC\)](#).

For information about managing a VPC with Amazon RDS, see [Working with an Amazon RDS DB Instance in a VPC](#).

Working with Connections on the AWS Glue Console

A connection contains the properties that are needed to access your data store. To see a list of all the connections that you have created, open the AWS Glue console at <https://console.aws.amazon.com/glue/>, and choose the **Connections** tab.

The Connections list displays the following properties about each connection:

Name

When you create a connection, you give it a unique name.

Type

The data store type and the properties that are required for a successful connection. AWS Glue uses the JDBC protocol to access several types of data stores.

Date created

The date and time (UTC) that the connection was created.

Last updated

The date and time (UTC) that the connection was last updated.

Updated by

The user who created or last updated the connection.

From the **Connections** tab in the AWS Glue console, you can add, edit, and delete connections. To see more details for a connection, choose the connection name in the list. Details include the information you defined when you created the connection.

As a best practice, before you use a data store connection in an ETL job, choose **Test connection**. AWS Glue uses the parameters in your connection to confirm that it can access your data store and reports back any errors. Connections are required for Amazon Redshift, Amazon Relational Database Service (Amazon RDS), and JDBC data stores. For more information, see [Connecting to a JDBC Data Store in a VPC \(p. 111\)](#).

Important

Currently, an ETL job can use only one JDBC connection. If you have multiple data stores in a job, they must be on the same subnet.

Adding a JDBC Connection to a Data Store

To add a connection in the AWS Glue console, choose **Add connection**. The wizard guides you through adding the properties that are required to create a JDBC connection to a data store. If you choose Amazon Redshift or Amazon RDS, AWS Glue tries to determine the underlying JDBC properties to create the connection.

When you define a connection, values for the following properties are required:

Connection name

Type a unique name for your connection.

Connection type

Choose either Amazon Redshift, Amazon RDS, or JDBC.

- If you choose Amazon Redshift, choose a **Cluster**, **Database name**, **Username**, and **Password** in your account to create a JDBC connection.
- If you choose Amazon RDS, choose an **Instance**, **Database name**, **Username**, and **Password** in your account to create a JDBC connection. The console also lists the supported database engine types.

Require SSL connection

Select this option to require AWS Glue to verify that the JDBC database connection is connected over a trusted Secure Socket Layer (SSL). This option is optional. If not selected, AWS Glue can ignore failures when it uses SSL to encrypt a connection to a JDBC database. See the documentation for your database for configuration instructions. When you select this option, if AWS Glue cannot connect using SSL, the job run, crawler, or ETL statements in a development endpoint fail.

This option is validated on the AWS Glue client side. AWS Glue only connects to JDBC over SSL with certificate and host name validation. Support is available for:

- Oracle
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL (Amazon RDS instances only)
- Aurora MySQL (Amazon RDS instances only)
- Aurora Postgres (Amazon RDS instances only)

Note

To enable an **Amazon RDS Oracle** data store to use **Require SSL connection**, you need to create and attach an option group to the Oracle instance.

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

2. Add an **Option group** to the Amazon RDS Oracle instance. For more information about how to add an option group on the Amazon RDS console, see [Creating an Option Group](#)
3. Add an **Option** to the option group for **SSL**. The **Port** you specify for SSL is later used when you create an AWS Glue JDBC connection URL for the Amazon RDS Oracle instance. For more information about how to add an option on the Amazon RDS console, see [Adding an Option to an Option Group](#). For more information about the Oracle SSL option, see [Oracle SSL](#).
4. On the AWS Glue console, create a connection to the Amazon RDS Oracle instance. In the connection definition, select **Require SSL connection**, and when requested, enter the **Port** you used in the Amazon RDS Oracle SSL option.

Custom certificate fields (optional)

If you have a certificate that you are currently using for SSL communication with your on-premise or cloud databases, you can use that certificate for SSL connections to AWS Glue data sources or targets. The following optional fields are available when you select **Require SSL connection**:

Custom JDBC certificate

Enter an Amazon S3 location containing a custom root certificate. AWS Glue uses this certificate to establish an SSL connection to the database. AWS Glue handles only X.509 certificates. The certificate must be DER-encoded and supplied in Base64 encoding PEM format.

If this field is left blank, the default certificate is used.

Skip certificate validation

Select this check box to skip validation of the custom certificate by AWS Glue. If you choose to validate, AWS Glue validates the signature algorithm and subject public key algorithm for the certificate. If the certificate fails validation, any ETL job or crawler that uses the connection fails.

The only permitted signature algorithms are SHA256withRSA, SHA384withRSA, or SHA512withRSA. For the subject public key algorithm, the key length must be at least 2048.

Custom JDBC certificate string

Enter database-specific certificate information. This is a string that is used for domain matching or distinguished name (DN) matching. For Oracle Database, this maps to the **SSL_SERVER_CERT_DN** parameter in the security section of the **tnsnames.ora** file. For Microsoft SQL Server, this is used as **hostNameInCertificate**.

The following is an example for the Oracle Database **SSL_SERVER_CERT_DN** parameter.

```
cn=sales,cn=OracleContext,dc=us,dc=example,dc=com
```

JDBC URL

Type the URL for your JDBC data store. For most database engines, this field is in the following format.

`jdbc:protocol://host:port/db_name`

Depending on the database engine, a different JDBC URL format might be required. This format can have slightly different use of the colon (:) and slash (/) or different keywords to specify databases.

For JDBC to connect to the data store, a **db_name** in the data store is required. The **db_name** is used to establish a network connection with the supplied **username** and **password**. When connected, AWS Glue can access other databases in the data store to run a crawler or run an ETL job.

The following JDBC URL examples show the syntax for several database engines.

- To connect to an Amazon Redshift cluster data store with a **dev** database:

```
jdbc:redshift://xxx.us-east-1.redshift.amazonaws.com:8192/dev
```

- To connect to an Amazon RDS for MySQL data store with an employee database:

```
jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/
employee
```

- To connect to an Amazon RDS for PostgreSQL data store with an employee database:

```
jdbc:postgresql://xxx-cluster.cluster-xxx.us-
east-1.rds.amazonaws.com:5432/employee
```

- To connect to an Amazon RDS for Oracle data store with an employee service name:

```
jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-
east-1.rds.amazonaws.com:1521/employee
```

The syntax for Amazon RDS for Oracle can follow the following patterns:

- `jdbc:oracle:thin://@host:port/service_name`

- `jdbc:oracle:thin://@host:port:SID`

- To connect to an Amazon RDS for Microsoft SQL Server data store with an employee database:

```
jdbc:sqlserver://xxx-cluster.cluster-xxx.us-
east-1.rds.amazonaws.com:1433;databaseName=employee
```

The syntax for Amazon RDS for SQL Server can follow the following patterns:

- `jdbc:sqlserver://server_name:port;database=db_name`

- `jdbc:sqlserver://server_name:port;databaseName=db_name`

The following table lists the JDBC driver versions that AWS Glue supports.

Product	JDBC Driver Version
Microsoft SQL Server	6.x
MySQL	5.1
Oracle Database	11.2
PostgreSQL	42.x
Amazon Redshift	4.1

Username

Provide a user name that has permission to access the JDBC data store.

Password

Type the password for the user name that has access permission to the JDBC data store.

Port

Type the port used in the JDBC URL to connect to an Amazon RDS Oracle instance. This field is only shown when **Require SSL connection** is selected for an Amazon RDS Oracle instance.

VPC

Choose the name of the virtual private cloud (VPC) that contains your data store. The AWS Glue console lists all VPCs for the current region.

Subnet

Choose the subnet within the VPC that contains your data store. The AWS Glue console lists all subnets for the data store in your VPC.

Security groups

Choose the security groups that are associated with your data store. AWS Glue requires one or more security groups with an inbound source rule that allows AWS Glue to connect. The AWS Glue console lists all security groups that are granted inbound access to your VPC. AWS Glue associates these security groups with the elastic network interface that is attached to your VPC subnet.

Defining Crawlers

You can use a crawler to populate the AWS Glue Data Catalog with tables. This is the primary method used by most AWS Glue users. A crawler can crawl multiple data stores in a single run. Upon completion, the crawler creates or updates one or more tables in your Data Catalog. Extract, transform, and load (ETL) jobs that you define in AWS Glue use these Data Catalog tables as sources and targets. The ETL job reads from and writes to the data stores that are specified in the source and target Data Catalog tables.

For more information about using the AWS Glue console to add a crawler, see [Working with Crawlers on the AWS Glue Console \(p. 127\)](#).

Which Data Stores Can I Crawl?

Crawlers can crawl both file-based and table-based data stores.

Crawlers can crawl the following data stores through their respective native interfaces:

- Amazon Simple Storage Service (Amazon S3)
- Amazon DynamoDB

Crawlers can crawl the following data stores through a JDBC connection:

- Amazon Redshift
- Amazon Relational Database Service (Amazon RDS)
 - Amazon Aurora
 - MariaDB
 - Microsoft SQL Server
 - MySQL
 - Oracle
 - PostgreSQL
- Publicly accessible databases
 - Aurora
 - MariaDB
 - SQL Server
 - MySQL
 - Oracle
 - PostgreSQL

For Amazon S3 and Amazon DynamoDB, crawlers use an AWS Identity and Access Management (IAM) role for permission to access your data stores. *The role you pass to the crawler must have permission to access Amazon S3 paths and Amazon DynamoDB tables that are crawled.* For JDBC connections, crawlers use user name and password credentials. For more information, see [Adding a Connection to Your Data Store \(p. 110\)](#).

When you define an Amazon S3 data store to crawl, you can choose whether to crawl a path in your account or another account. The output of the crawler is one or more metadata tables defined in the AWS Glue Data Catalog. A table is created for one or more files found in your data store. If all the Amazon S3 files in a folder have the same schema, the crawler creates one table. Also, if the Amazon S3 object is partitioned, only one metadata table is created.

If the data store that is being crawled is a relational database, the output is also a set of metadata tables defined in the AWS Glue Data Catalog. When you crawl a relational database, you must provide authorization credentials for a connection to read objects in the database engine. Depending on the type of database engine, you can choose which objects are crawled, such as databases, schemas, and tables.

If the data store that is being crawled is one or more Amazon DynamoDB tables, the output is one or more metadata tables in the AWS Glue Data Catalog. When defining a crawler using the AWS Glue console, you specify a DynamoDB table. If you're using the AWS Glue API, you specify a list of tables.

What Happens When a Crawler Runs?

When a crawler runs, it takes the following actions to interrogate a data store:

- **Classifies data to determine the format, schema, and associated properties of the raw data** – You can configure the results of classification by creating a custom classifier.
- **Groups data into tables or partitions** – Data is grouped based on crawler heuristics.
- **Writes metadata to the Data Catalog** – You can configure how the crawler adds, updates, and deletes tables and partitions.

The metadata tables that a crawler creates are contained in a database when you define a crawler. If your crawler does not define a database, your tables are placed in the default database. In addition, each table has a classification column that is filled in by the classifier that first successfully recognized the data store.

If the file that is crawled is compressed, the crawler must download it to process it. When a crawler runs, it interrogates files to determine their format and compression type and writes these properties into the Data Catalog. Some file formats (for example, Apache Parquet) enable you to compress parts of the file as it is written. For these files, the compressed data is an internal component of the file, and AWS Glue does not populate the `compressionType` property when it writes tables into the Data Catalog. In contrast, if an *entire file* is compressed by a compression algorithm (for example, gzip), then the `compressionType` property is populated when tables are written into the Data Catalog.

The crawler generates the names for the tables that it creates. The names of the tables that are stored in the AWS Glue Data Catalog follow these rules:

- Only alphanumeric characters and underscore (`_`) are allowed.
- Any custom prefix cannot be longer than 64 characters.
- The maximum length of the name cannot be longer than 128 characters. The crawler truncates generated names to fit within the limit.
- If duplicate table names are encountered, the crawler adds a hash string suffix to the name.

If your crawler runs more than once, perhaps on a schedule, it looks for new or changed files or tables in your data store. The output of the crawler includes new tables and partitions found since a previous run.

How Does a Crawler Determine When to Create Partitions?

When an AWS Glue crawler scans Amazon S3 and detects multiple folders in a bucket, it determines the root of a table in the folder structure and which folders are partitions of a table. The name of the table

is based on the Amazon S3 prefix or folder name. You provide an **Include path** that points to the folder level to crawl. When the majority of schemas at a folder level are similar, the crawler creates partitions of a table instead of two separate tables. To influence the crawler to create separate tables, add each table's root folder as a separate data store when you define the crawler.

For example, with the following Amazon S3 structure:

```
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 schemas for `table1` and `table2` are similar, and a single data store is defined in the crawler with **Include path** `s3://bucket01/folder1/`, the crawler creates a single table with two partition columns. One partition column contains `table1` and `table2`, and a second partition column contains `partition1` through `partition5`. To create two separate tables, define the crawler with two data stores. In this example, define the first **Include path** as `s3://bucket01/folder1/table1/` and the second as `s3://bucket01/folder1/table2`.

Note

In Amazon Athena, each table corresponds to an Amazon S3 prefix with all the objects in it. If objects have different schemas, Athena does not recognize different objects within the same prefix as separate tables. This can happen if a crawler creates multiple tables from the same Amazon S3 prefix. This might lead to queries in Athena that return zero results. For Athena to properly recognize and query tables, create the crawler with a separate **Include path** for each different table schema in the Amazon S3 folder structure. For more information, see [Best Practices When Using Athena with AWS Glue](#) and this [AWS Knowledge Center article](#).

Crawler Properties

When defining a crawler using the AWS Glue console or the AWS Glue API, you specify the following information:

Crawler name and optional descriptors and settings

Settings include tags, security configuration, and custom classifiers. You define custom classifiers before defining crawlers. For more information, see the following:

- [AWS Tags in AWS Glue \(p. 229\)](#)
- [Adding Classifiers to a Crawler \(p. 129\)](#)

Crawler source type

The crawler can access data stores directly as the source of the crawl, or it can use existing catalog tables as the source. If the crawler uses existing catalog tables, it crawls the data stores that are specified by those catalog tables. For more information, see [Crawler Source Type \(p. 119\)](#).

Crawler sources (Choose one of the following two types.)

- One or more data stores

A crawler can crawl multiple data stores of different types (Amazon S3, Amazon DynamoDB, and JDBC) in a single run.

- List of Data Catalog tables

The catalog tables specify the data stores to crawl. The crawler can crawl only catalog tables in a single run; it can't mix in other source types.

Connection (for JDBC data stores only)

Select or add a connection. For information about connections, see [the section called "Adding a Connection to Your Data Store" \(p. 110\)](#).

Include path

For an Amazon S3 data store

Choose whether to specify a path in your account or another account, and then browse to choose an Amazon S3 path.

For a JDBC data store

Enter `<database>/<schema>/<table>` or `<database>/<table>`, depending on the database product. Oracle Database and MySQL don't support schema in the path. You can substitute the percent (%) character for `<schema>` or `<table>`. For example, for an Oracle database with a system identifier (SID) of `orcl`, enter `orcl/%` to import all tables to which the user named in the connection has access.

Important

This field is case-sensitive.

Exclude patterns

These enable you to exclude certain files or tables from the crawl. For more information, see [Include and Exclude Patterns \(p. 120\)](#).

IAM role or JDBC credentials for accessing the data stores

For more information, see [Managing Access Permissions for AWS Glue Resources \(p. 52\)](#)

Crawler schedule

You can run a crawler on demand or define a schedule for automatic running of the crawler. For more information, see [Scheduling an AWS Glue Crawler \(p. 126\)](#).

Destination database within the Data Catalog for the created catalog tables

For more information, see [Defining a Database in Your Data Catalog \(p. 105\)](#).

Crawler configuration options

Options include how the crawler should handle detected schema changes, deleted objects in the data store, and more. For more information, see [Configuring a Crawler \(p. 123\)](#).

When you define a crawler, you choose one or more classifiers that evaluate the format of your data to infer a schema. When the crawler runs, the first classifier in your list to successfully recognize your data store is used to create a schema for your table. You can use built-in classifiers or define your own. You define your custom classifiers in a separate operation, before you define the crawlers. AWS Glue provides built-in classifiers to infer schemas from common files with formats that include JSON, CSV, and Apache Avro. For the current list of built-in classifiers in AWS Glue, see [Built-In Classifiers in AWS Glue \(p. 130\)](#).

Crawler Source Type

A crawler can access data stores directly as the source of the crawl or use existing catalog tables as the source. If the crawler uses existing catalog tables, it crawls the data stores specified by those catalog tables.

A common reason to specify a catalog table as the source is that you created the table manually (because you already knew the structure of the data store) and you want a crawler to keep the table updated, including adding new partitions. For a discussion of other reasons, see [Updating Manually Created Data Catalog Tables Using Crawlers \(p. 107\)](#).

When you specify existing tables as the crawler source type, the following conditions apply:

- Database name is optional.
- Only catalog tables that specify Amazon S3 or Amazon DynamoDB data stores are permitted.
- No new catalog tables are created when the crawler runs. Existing tables are updated as needed, including adding new partitions.
- Deleted objects found in the data stores are ignored; no catalog tables are deleted. Instead, the crawler writes a log message. (`SchemaChangePolicy.DeleteBehavior=LOG`)
- The crawler configuration option to create a single schema for each Amazon S3 path is enabled by default and cannot be disabled. (`TableGroupingPolicy=CombineCompatibleSchemas`) For more information, see [How to Create a Single Schema for Each Amazon S3 Include Path \(p. 126\)](#).
- You can't mix catalog tables as a source with any other source types (for example Amazon S3 or Amazon DynamoDB).

Include and Exclude Patterns

When evaluating what to include or exclude in a crawl, a crawler starts by evaluating the required include path for Amazon S3 and relational data stores. For every data store that you want to crawl, you must specify a single include path.

For Amazon S3 data stores, the syntax is `bucket-name/folder-name/file-name.ext`. To crawl all objects in a bucket, you specify just the bucket name in the include path.

For JDBC data stores, the syntax is either `database-name/schema-name/table-name` or `database-name/table-name`. The syntax depends on whether the database engine supports schemas within a database. For example, for database engines such as MySQL or Oracle, don't specify a `schema-name` in your include path. You can substitute the percent sign (%) for a schema or table in the include path to represent all schemas or all tables in a database. You cannot substitute the percent sign (%) for database in the include path.

A crawler connects to a JDBC data store using an AWS Glue connection that contains a JDBC URI connection string. The crawler only has access to objects in the database engine using the JDBC user name and password in the AWS Glue connection. *The crawler can only create tables that it can access through the JDBC connection.* After the crawler accesses the database engine with the JDBC URI, the include path is used to determine which tables in the database engine are created in the Data Catalog. For example, with MySQL, if you specify an include path of `MyDatabase/%`, then all tables within `MyDatabase` are created in the Data Catalog. When accessing Amazon Redshift, if you specify an include path of `MyDatabase/%`, then all tables within all schemas for database `MyDatabase` are created in the Data Catalog. If you specify an include path of `MyDatabase/MySchema/%`, then all tables in database `MyDatabase` and schema `MySchema` are created.

After you specify an include path, you can then exclude objects from the crawl that your include path would otherwise include by specifying one or more Unix-style glob exclude patterns. These patterns are applied to your include path to determine which objects are excluded. These patterns are also stored as a property of tables created by the crawler. AWS Glue PySpark extensions, such as `create_dynamic_frame.from_catalog`, read the table properties and exclude objects defined by the exclude pattern.

AWS Glue supports the following kinds of glob patterns in the exclude pattern.

Exclude pattern	Description
<code>*.csv</code>	Matches an Amazon S3 path that represents an object name in the current folder ending in <code>.csv</code>
<code>*.*</code>	Matches all object names that contain a dot
<code>*.{csv,avro}</code>	Matches object names ending with <code>.csv</code> or <code>.avro</code>

Exclude pattern	Description
foo.?	Matches object names starting with <code>foo.</code> that are followed by a single character extension
<code>myfolder/*</code>	Matches objects in one level of subfolder from <code>myfolder</code> , such as <code>/myfolder/mysource</code>
<code>myfolder/**</code>	Matches objects in two levels of subfolders from <code>myfolder</code> , such as <code>/myfolder/mysource/data</code>
<code>myfolder/***</code>	Matches objects in all subfolders of <code>myfolder</code> , such as <code>/myfolder/mysource/mydata</code> and <code>/myfolder/mysource/data</code>
<code>myfolder**</code>	Matches subfolder <code>myfolder</code> as well as files below <code>myfolder</code> , such as <code>/myfolder</code> and <code>/myfolder/mydata.txt</code>
<code>Market*</code>	Matches tables in a JDBC database with names that begin with <code>Market</code> , such as <code>Market_us</code> and <code>Market_fr</code>

AWS Glue interprets glob exclude patterns as follows:

- The slash (/) character is the delimiter to separate Amazon S3 keys into a folder hierarchy.
- The asterisk (*) character matches zero or more characters of a name component without crossing folder boundaries.
- A double asterisk (**) matches zero or more characters crossing folder or schema boundaries.
- The question mark (?) character matches exactly one character of a name component.
- The backslash (\) character is used to escape characters that otherwise can be interpreted as special characters. The expression \\ matches a single backslash, and \{ matches a left brace.
- Brackets [] create a bracket expression that matches a single character of a name component out of a set of characters. For example, [abc] matches a, b, or c. The hyphen (-) can be used to specify a range, so [a-z] specifies a range that matches from a through z (inclusive). These forms can be mixed, so [abce-g] matches a, b, c, e, f, or g. If the character after the bracket ([]) is an exclamation point (!), the bracket expression is negated. For example, [!a-c] matches any character except a, b, or c.

Within a bracket expression, the *, ?, and \ characters match themselves. The hyphen (-) character matches itself if it is the first character within the brackets, or if it's the first character after the ! when you are negating.

- Braces ({ }) enclose a group of subpatterns, where the group matches if any subpattern in the group matches. A comma (,) character is used to separate the subpatterns. Groups cannot be nested.
- Leading period or dot characters in file names are treated as normal characters in match operations. For example, the * exclude pattern matches the file name .hidden.

Example of Amazon S3 Exclude Patterns

Each exclude pattern is evaluated against the include path. For example, suppose that you have the following Amazon S3 directory structure:

```
/mybucket/myfolder/
  departments/
    finance.json
```

```

market-us.json
market-emea.json
market-ap.json
employees/
    hr.json
    john.csv
    jane.csv
    juan.txt

```

Given the include path `s3://mybucket/myfolder/`, the following are some sample results for exclude patterns:

Exclude pattern	Results
<code>departments/**</code>	Excludes all files and folders below <code>departments</code> and includes the <code>employees</code> folder and its files
<code>departments/market*</code>	Excludes <code>market-us.json</code> , <code>market-emea.json</code> , and <code>market-ap.json</code>
<code>**.csv</code>	Excludes all objects below <code>myfolder</code> that have a name ending with <code>.csv</code>
<code>employees/*.csv</code>	Excludes all <code>.csv</code> files in the <code>employees</code> folder

Example of Excluding a Subset of Amazon S3 Partitions

Suppose that your data is partitioned by day, so that each day in a year is in a separate Amazon S3 partition. For January 2015, there are 31 partitions. Now, to crawl data for only the first week of January, you must exclude all partitions except days 1 through 7:

```
2015/01/{[!0],0[8-9]}**, 2015/0[2-9]/**, 2015/1[0-2]**
```

Take a look at the parts of this glob pattern. The first part, `2015/01/{[!0],0[8-9]}**`, excludes all days that don't begin with a "0" in addition to day 08 and day 09 from month 01 in year 2015. Notice that "***" is used as the suffix to the day number pattern and crosses folder boundaries to lower-level folders. If "*" is used, lower folder levels are not excluded.

The second part, `2015/0[2-9]/**`, excludes days in months 02 to 09, in year 2015.

The third part, `2015/1[0-2]**`, excludes days in months 10, 11, and 12, in year 2015.

Example of JDBC Exclude Patterns

Suppose that you are crawling a JDBC database with the following schema structure:

```

MyDatabase/MySchema/
    HR_us
    HR_fr
    Employees_Table
    Finance
    Market_US_Table
    Market_EMEA_Table
    Market_AP_Table

```

Given the include path `MyDatabase/MySchema/%`, the following are some sample results for exclude patterns:

Exclude pattern	Results
HR*	Excludes the tables with names that begin with <code>HR</code>
Market_*	Excludes the tables with names that begin with <code>Market_</code>
**_Table	Excludes all tables with names that end with <code>_Table</code>

Configuring a Crawler

When a crawler runs, it might encounter changes to your data store that result in a schema or partition that is different from a previous crawl. You can use the AWS Management Console or the AWS Glue API to configure how your crawler processes certain types of changes.

Topics

- [Configuring a Crawler on the AWS Glue Console \(p. 123\)](#)
- [Configuring a Crawler Using the API \(p. 124\)](#)
- [How to Prevent the Crawler from Changing an Existing Schema \(p. 125\)](#)
- [How to Create a Single Schema for Each Amazon S3 Include Path \(p. 126\)](#)

Configuring a Crawler on the AWS Glue Console

When you define a crawler using the AWS Glue console, you have several options for configuring the behavior of your crawler. For more information about using the AWS Glue console to add a crawler, see [Working with Crawlers on the AWS Glue Console \(p. 127\)](#).

When a crawler runs against a previously crawled data store, it might discover that a schema has changed or that some objects in the data store have been deleted. The crawler logs changes to a schema. Depending on the source type for the crawler, new tables and partitions might be created regardless of the schema change policy.

To specify what the crawler does when it finds changes in the schema, you can choose one of the following actions on the console:

- **Update the table definition in the Data Catalog** – Add new columns, remove missing columns, and modify the definitions of existing columns in the AWS Glue Data Catalog. Remove any metadata that is not set by the crawler. This is the default setting.
- **Add new columns only** – For tables that map to an Amazon S3 data store, add new columns as they are discovered, but don't remove or change the type of existing columns in the Data Catalog. Choose this option when the current columns in the Data Catalog are correct and you don't want the crawler to remove or change the type of the existing columns. If a fundamental Amazon S3 table attribute changes, such as classification, compression type, or CSV delimiter, mark the table as deprecated. Maintain input format and output format as they exist in the Data Catalog. Update SerDe parameters only if the parameter is one that is set by the crawler. *For all other data stores, modify existing column definitions.*
- **Ignore the change and don't update the table in the Data Catalog** – Only new tables and partitions are created.

A crawler might also discover new or changed partitions. By default, new partitions are added and existing partitions are updated if they have changed. In addition, you can set a crawler configuration

option to **Update all new and existing partitions with metadata from the table** on the AWS Glue console. When this option is set, partitions inherit metadata properties—such as their classification, input format, output format, SerDe information, and schema—from their parent table. Any changes to these properties in a table are propagated to its partitions. When this configuration option is set on an existing crawler, existing partitions are updated to match the properties of their parent table the next time the crawler runs.

To specify what the crawler does when it finds a deleted object in the data store, choose one of the following actions:

- **Delete tables and partitions from the Data Catalog**
- **Ignore the change and don't update the table in the Data Catalog**
- **Mark the table as deprecated in the Data Catalog** – This is the default setting.

Configuring a Crawler Using the API

When you define a crawler using the AWS Glue API, you can choose from several fields to configure your crawler. The `SchemaChangePolicy` in the crawler API determines what the crawler does when it discovers a changed schema or a deleted object. The crawler logs schema changes as it runs.

When a crawler runs, new tables and partitions are always created regardless of the schema change policy. You can choose one of the following actions in the `UpdateBehavior` field in the `SchemaChangePolicy` structure to determine what the crawler does when it finds a changed table schema:

- `UPDATE_IN_DATABASE` – Update the table in the AWS Glue Data Catalog. Add new columns, remove missing columns, and modify the definitions of existing columns. Remove any metadata that is not set by the crawler.
- `LOG` – Ignore the changes, and don't update the table in the Data Catalog.

You can also override the `SchemaChangePolicy` structure using a JSON object supplied in the crawler API Configuration field. This JSON object can contain a key-value pair to set the policy to not update existing columns and only add new columns. For example, provide the following JSON object as a string:

```
{  
    "Version": 1.0,  
    "CrawlerOutput": {  
        "Tables": { "AddOrUpdateBehavior": "MergeNewColumns" }  
    }  
}
```

This option corresponds to the **Add new columns only** option on the AWS Glue console. It overrides the `SchemaChangePolicy` structure for tables that result from crawling Amazon S3 data stores only. Choose this option if you want to maintain the metadata as it exists in the Data Catalog (the source of truth). New columns are added as they are encountered, including nested data types. But existing columns are not removed, and their type is not changed. If an Amazon S3 table attribute changes significantly, mark the table as deprecated, and log a warning that an incompatible attribute needs to be resolved.

When a crawler runs against a previously crawled data store, it might discover new or changed partitions. By default, new partitions are added and existing partitions are updated if they have changed. In addition, you can set a crawler configuration option to `InheritFromTable` (corresponding to the **Update all new and existing partitions with metadata from the table** option on the AWS Glue console). When this option is set, partitions inherit metadata properties from their parent table, such as their classification, input format, output format, SerDe information, and schema. Any property changes to the parent table are propagated to its partitions.

When this configuration option is set on an existing crawler, existing partitions are updated to match the properties of their parent table the next time the crawler runs. This behavior is set crawler API Configuration field. For example, provide the following JSON object as a string:

```
{  
    "Version": 1.0,  
    "CrawlerOutput": {  
        "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }  
    }  
}
```

The crawler API Configuration field can set multiple configuration options. For example, to configure the crawler output for both partitions and tables, you can provide a string representation of the following JSON object:

```
{  
    "Version": 1.0,  
    "CrawlerOutput": {  
        "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },  
        "Tables": { "AddOrUpdateBehavior": "MergeNewColumns" }  
    }  
}
```

You can choose one of the following actions to determine what the crawler does when it finds a deleted object in the data store. The DeleteBehavior field in the SchemaChangePolicy structure in the crawler API sets the behavior of the crawler when it discovers a deleted object.

- **DELETE_FROM_DATABASE** – Delete tables and partitions from the Data Catalog.
- **LOG** – Ignore the change. Don't update the Data Catalog. Write a log message instead.
- **DEPRECATE_IN_DATABASE** – Mark the table as deprecated in the Data Catalog. This is the default setting.

How to Prevent the Crawler from Changing an Existing Schema

If you don't want a crawler to overwrite updates you made to existing fields in an Amazon S3 table definition, choose the option on the console to **Add new columns only** or set the configuration option `MergeNewColumns`. This applies to tables and partitions, unless `Partitions.AddOrUpdateBehavior` is overridden to `InheritFromTable`.

If you don't want a table schema to change at all when a crawler runs, set the schema change policy to **LOG**. You can also set a configuration option that sets partition schemas to inherit from the table.

If you are configuring the crawler on the console, you can choose the following actions:

- **Ignore the change and don't update the table in the Data Catalog**
- **Update all new and existing partitions with metadata from the table**

When you configure the crawler using the API, set the following parameters:

- Set the `UpdateBehavior` field in `SchemaChangePolicy` structure to `LOG`.
- Set the `Configuration` field with a string representation of the following JSON object in the crawler API; for example:

```
{  
    "Version": 1.0,
```

```
    "CrawlerOutput": {  
        "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }  
    }  
}
```

How to Create a Single Schema for Each Amazon S3 Include Path

By default, when a crawler defines tables for data stored in Amazon S3, it considers both data compatibility and schema similarity. Data compatibility factors that it considers include whether the data is of the same format (for example, JSON), the same compression type (for example, GZIP), the structure of the Amazon S3 path, and other data attributes. Schema similarity is a measure of how closely the schemas of separate Amazon S3 objects are similar.

You can configure a crawler to `CombineCompatibleSchemas` into a common table definition when possible. With this option, the crawler still considers data compatibility, but ignores the similarity of the specific schemas when evaluating Amazon S3 objects in the specified include path.

If you are configuring the crawler on the console, to combine schemas, select the crawler option **Create a single schema for each S3 path**.

When you configure the crawler using the API, set the following configuration option:

- Set the `Configuration` field with a string representation of the following JSON object in the crawler API; for example:

```
{  
    "Version": 1.0,  
    "Grouping": {  
        "TableGroupingPolicy": "CombineCompatibleSchemas" }  
}
```

To help illustrate this option, suppose that you define a crawler with an include path `s3://bucket/table1/`. When the crawler runs, it finds two JSON files with the following characteristics:

- File 1 – S3://bucket/table1/year=2017/data1.json**
- File content – { “A”: 1, “B”: 2 }*
- Schema – A:int, B:int*

- File 2 – S3://bucket/table1/year=2018/data2.json**
- File content – { “C”: 3, “D”: 4 }*
- Schema – C: int, D: int*

By default, the crawler creates two tables, named `year_2017` and `year_2018` because the schemas are not sufficiently similar. However, if the option **Create a single schema for each S3 path** is selected, and if the data is compatible, the crawler creates one table. The table has the schema `A:int,B:int,C:int,D:int` and `partitionKey year:string`.

Scheduling an AWS Glue Crawler

You can run an AWS Glue crawler on demand or on a regular schedule. Crawler schedules can be expressed in `cron` format. For more information, see [cron](#) in Wikipedia.

When you create a crawler based on a schedule, you can specify certain constraints, such as the frequency the crawler runs, which days of the week it runs, and at what time. These constraints are based on cron. When setting up a crawler schedule, you should consider the features and limitations of cron. For example, if you choose to run your crawler on day 31 each month, keep in mind that some months don't have 31 days.

For more information about using cron to schedule jobs and crawlers, see [Time-Based Schedules for Jobs and Crawlers \(p. 221\)](#).

Working with Crawlers on the AWS Glue Console

A crawler accesses your data store, extracts metadata, and creates table definitions in the AWS Glue Data Catalog. The **Crawlers** pane in the AWS Glue console lists all the crawlers that you create. The list displays status and metrics from the last run of your crawler.

To add a crawler using the console

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Choose **Crawlers** in the navigation pane.
2. Choose **Add crawler**, and follow the instructions in the **Add crawler** wizard.

Note

To get step-by-step guidance for adding a crawler, choose **Add crawler** under **Tutorials** in the navigation pane. You can also use the **Add crawler** wizard to create and modify an IAM role that attaches a policy that includes permissions for your Amazon Simple Storage Service (Amazon S3) data stores.

Optionally, you can tag your crawler with a **Tag key** and optional **Tag value**. Once created, tag keys are read-only. Use tags on some resources to help you organize and identify them. For more information, see [AWS Tags in AWS Glue \(p. 229\)](#).

Optionally, you can add a security configuration to a crawler to specify at-rest encryption options.

When a crawler runs, the provided IAM role must have permission to access the data store that is crawled. For an Amazon S3 data store, you can use the AWS Glue console to create a policy or add a policy similar to the following:

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

If the crawler reads KMS encrypted Amazon S3 data, then the **IAM role** must have decrypt permission on the KMS key. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#).

For an Amazon DynamoDB data store, you can use the AWS Glue console to create a policy or add a policy similar to the following:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb:DescribeTable",  
                "dynamodb:Scan"  
            ],  
            "Resource": [  
                "arn:aws:dynamodb:region:account-id:table/table-name*"  
            ]  
        }  
    ]  
}
```

For Amazon S3 data stores, an exclude pattern is relative to the include path. For more information about glob patterns, see [Which Data Stores Can I Crawl? \(p. 116\)](#).

When you crawl a JDBC data store, a connection is required. For more information, see [Working with Connections on the AWS Glue Console \(p. 112\)](#). An exclude path is relative to the include path. For example, to exclude a table in your JDBC data store, type the table name in the exclude path.

When you crawl DynamoDB tables, you can choose one table name from the list of DynamoDB tables in your account.

Viewing Crawler Results

To view the results of a crawler, find the crawler name in the list and choose the **Logs** link. This link takes you to the CloudWatch Logs, where you can see details about which tables were created in the AWS Glue Data Catalog and any errors that were encountered. You can manage your log retention period in the CloudWatch console. The default log retention is `Never Expire`. For more information about how to change the retention period, see [Change Log Data Retention in CloudWatch Logs](#).

To see details of a crawler, choose the crawler name in the list. Crawler details include the information you defined when you created the crawler with the **Add crawler** wizard. When a crawler run completes, choose **Tables** in the navigation pane to see the tables that were created by your crawler in the database that you specified.

Note

The crawler assumes the permissions of the **IAM role** that you specify when you define it. This IAM role must have permissions to extract data from your data store and write to the Data Catalog. The AWS Glue console lists only IAM roles that have attached a trust policy for the AWS Glue principal service. From the console, you can also create an IAM role with an IAM policy to access Amazon S3 data stores accessed by the crawler. For more information about providing roles for AWS Glue, see [Identity-Based Policies \(p. 55\)](#).

The following are some important properties and metrics about the last run of a crawler:

Name

When you create a crawler, you must give it a unique name.

Schedule

You can choose to run your crawler on demand or choose a frequency with a schedule. For more information about scheduling a crawler, see [Scheduling a Crawler \(p. 126\)](#).

Status

A crawler can be ready, starting, stopping, scheduled, or schedule paused. A running crawler progresses from starting to stopping. You can resume or pause a schedule attached to a crawler.

Logs

Links to any available logs from the last run of the crawler.

Last runtime

The amount of time it took the crawler to run when it last ran.

Median runtime

The median amount of time it took the crawler to run since it was created.

Tables updated

The number of tables in the AWS Glue Data Catalog that were updated by the latest run of the crawler.

Tables added

The number of tables that were added into the AWS Glue Data Catalog by the latest run of the crawler.

Adding Classifiers to a Crawler

A classifier reads the data in a data store. If it recognizes the format of the data, it generates a schema. The classifier also returns a certainty number to indicate how certain the format recognition was.

AWS Glue provides a set of built-in classifiers, but you can also create custom classifiers. AWS Glue invokes custom classifiers first, in the order that you specify in your crawler definition. Depending on the results that are returned from custom classifiers, AWS Glue might also invoke built-in classifiers. If a classifier returns `certainty=1.0` during processing, it indicates that it's 100 percent certain that it can create the correct schema. AWS Glue then uses the output of that classifier.

If no classifier returns `certainty=1.0`, AWS Glue uses the output of the classifier that has the highest certainty. If no classifier returns a certainty greater than `0.0`, AWS Glue returns the default classification string of `UNKNOWN`.

When Do I Use a Classifier?

You use classifiers when you crawl a data store to define metadata tables in the AWS Glue Data Catalog. You can set up your crawler with an ordered set of classifiers. When the crawler invokes a classifier, the classifier determines whether the data is recognized. If the classifier can't recognize the data or is not 100 percent certain, the crawler invokes the next classifier in the list to determine whether it can recognize the data.

For more information about creating a classifier using the AWS Glue console, see [Working with Classifiers on the AWS Glue Console \(p. 142\)](#).

Custom Classifiers

The output of a classifier includes a string that indicates the file's classification or format (for example, `json`) and the schema of the file. For custom classifiers, you define the logic for creating the schema based on the type of classifier. Classifier types include defining schemas based on grok patterns, XML tags, and JSON paths.

If you change a classifier definition, any data that was previously crawled using the classifier is not reclassified. A crawler keeps track of previously crawled data. New data is classified with the updated classifier, which might result in an updated schema. If the schema of your data has evolved, update the

classifier to account for any schema changes when your crawler runs. To reclassify data to correct an incorrect classifier, create a new crawler with the updated classifier.

For more information about creating custom classifiers in AWS Glue, see [Writing Custom Classifiers \(p. 132\)](#).

Note

If your data format is recognized by one of the built-in classifiers, you don't need to create a custom classifier.

Built-In Classifiers in AWS Glue

AWS Glue provides built-in classifiers for various formats, including JSON, CSV, web logs, and many database systems.

If AWS Glue doesn't find a custom classifier that fits the input data format with 100 percent certainty, it invokes the built-in classifiers in the order shown in the following table. The built-in classifiers return a result to indicate whether the format matches (`certainty=1.0`) or does not match (`certainty=0.0`). The first classifier that has `certainty=1.0` provides the classification string and schema for a metadata table in your Data Catalog.

Classifier type	Classification string	Notes
Apache Avro	<code>avro</code>	Reads the schema at the beginning of the file to determine format.
Apache ORC	<code>orc</code>	Reads the file metadata to determine format.
Apache Parquet	<code>parquet</code>	Reads the schema at the end of the file to determine format.
JSON	<code>json</code>	Reads the beginning of the file to determine format.
Binary JSON	<code>bson</code>	Reads the beginning of the file to determine format.
XML	<code>xml</code>	Reads the beginning of the file to determine format. AWS Glue determines the table schema based on XML tags in the document. For information about creating a custom XML classifier to specify rows in the document, see Writing XML Custom Classifiers (p. 136) .
Amazon Ion	<code>ion</code>	Reads the beginning of the file to determine format.
Combined Apache log	<code>combined_apache</code>	Determines log formats through a grok pattern.
Apache log	<code>apache</code>	Determines log formats through a grok pattern.
Linux kernel log	<code>linux_kernel</code>	Determines log formats through a grok pattern.
Microsoft log	<code>microsoft_log</code>	Determines log formats through a grok pattern.
Ruby log	<code>ruby_logger</code>	Reads the beginning of the file to determine format.
Squid 3.x log	<code>squid</code>	Reads the beginning of the file to determine format.
Redis monitor log	<code>redismonlog</code>	Reads the beginning of the file to determine format.
Redis log	<code>redislog</code>	Reads the beginning of the file to determine format.

Classifier type	Classification string	Notes
CSV	csv	Checks for the following delimiters: comma (,), pipe (), tab (\t), semicolon (;), and Ctrl-A (\u0001). Ctrl-A is the Unicode control character for Start Of Heading.
Amazon Redshift	redshift	Uses JDBC connection to import metadata.
MySQL	mysql	Uses JDBC connection to import metadata.
PostgreSQL	postgresql	Uses JDBC connection to import metadata.
Oracle database	oracle	Uses JDBC connection to import metadata.
Microsoft SQL Server	sqlserver	Uses JDBC connection to import metadata.
Amazon DynamoDB	dynamodb	Reads data from the DynamoDB table.

Files in the following compressed formats can be classified:

- ZIP (supported for archives containing only a single file). Note that Zip is not well-supported in other services (because of the archive).
- BZIP
- GZIP
- LZ4
- Snappy (supported for both standard and Hadoop native Snappy formats)

Built-In CSV Classifier

The built-in CSV classifier parses CSV file contents to determine the schema for an AWS Glue table. This classifier checks for the following delimiters:

- Comma (,)
- Pipe (|)
- Tab (\t)
- Semicolon (;)
- Ctrl-A (\u0001)

Ctrl-A is the Unicode control character for Start Of Heading.

To be classified as CSV, the table schema must have at least two columns and two rows of data. The CSV classifier uses a number of heuristics to determine whether a header is present in a given file. If the classifier can't determine a header from the first row of data, column headers are displayed as col1, col2, col3, and so on. The built-in CSV classifier determines whether to infer a header by evaluating the following characteristics of the file:

- Every column in a potential header parses as a STRING data type.
- Except for the last column, every column in a potential header has content that is fewer than 150 characters. To allow for a trailing delimiter, the last column can be empty throughout the file.
- Every column in a potential header must meet the AWS Glue regex requirements for a column name.
- The header row must be sufficiently different from the data rows. To determine this, one or more of the rows must parse as other than STRING type. If all columns are of type STRING, then the first row of data is not sufficiently different from subsequent rows to be used as the header.

Note

If the built-in CSV classifier does not create your AWS Glue table as you want, you might be able to use one of the following alternatives:

- Change the column names in the Data Catalog, set the `SchemaChangePolicy` to `LOG`, and set the partition output configuration to `InheritFromTable` for future crawler runs.
- Create a custom grok classifier to parse the data and assign the columns that you want.
- The built-in CSV classifier creates tables referencing the `LazySimpleSerDe` as the serialization library, which is a good choice for type inference. However, if the CSV data contains quoted strings, edit the table definition and change the SerDe library to `OpenCSVSerDe`. Adjust any inferred types to `STRING`, set the `SchemaChangePolicy` to `LOG`, and set the partitions output configuration to `InheritFromTable` for future crawler runs. For more information about SerDe libraries, see [SerDe Reference](#) in the Amazon Athena User Guide.

Writing Custom Classifiers

You can provide a custom classifier to classify your data in AWS Glue. You can create a custom classifier using a grok pattern, an XML tag, JavaScript Object Notation (JSON), or comma-separated values (CSV). An AWS Glue crawler calls a custom classifier. If the classifier recognizes the data, it returns the classification and schema of the data to the crawler. You might need to define a custom classifier if your data doesn't match any built-in classifiers, or if you want to customize the tables that are created by the crawler.

For more information about creating a classifier using the AWS Glue console, see [Working with Classifiers on the AWS Glue Console \(p. 142\)](#).

AWS Glue runs custom classifiers before built-in classifiers, in the order you specify. When a crawler finds a classifier that matches the data, the classification string and schema are used in the definition of tables that are written to your AWS Glue Data Catalog.

Topics

- [Writing Grok Custom Classifiers \(p. 132\)](#)
- [Writing XML Custom Classifiers \(p. 136\)](#)
- [Writing JSON Custom Classifiers \(p. 137\)](#)
- [Writing CSV Custom Classifiers \(p. 142\)](#)

Writing Grok Custom Classifiers

Grok is a tool that is used to parse textual data given a matching pattern. A grok pattern is a named set of regular expressions (regex) that are used to match data one line at a time. AWS Glue uses grok patterns to infer the schema of your data. When a grok pattern matches your data, AWS Glue uses the pattern to determine the structure of your data and map it into fields.

AWS Glue provides many built-in patterns, or you can define your own. You can create a grok pattern using built-in patterns and custom patterns in your custom classifier definition. You can tailor a grok pattern to classify custom text file formats.

Note

AWS Glue grok custom classifiers use the `GrokSerDe` serialization library for tables created in the AWS Glue Data Catalog. If you are using the AWS Glue Data Catalog with Amazon Athena, Amazon EMR, or Redshift Spectrum, check the documentation about those services for information about support of the `GrokSerDe`. Currently, you might encounter problems querying tables created with the `GrokSerDe` from Amazon EMR and Redshift Spectrum.

The following is the basic syntax for the components of a grok pattern:

```
%{PATTERN:field-name}
```

Data that matches the named **PATTERN** is mapped to the **field-name** column in the schema, with a default data type of **string**. Optionally, the data type for the field can be cast to **byte**, **boolean**, **double**, **short**, **int**, **long**, or **float** in the resulting schema.

```
%{PATTERN:field-name:data-type}
```

For example, to cast a **num** field to an **int** data type, you can use this pattern:

```
%{NUMBER:num:int}
```

Patterns can be composed of other patterns. For example, you can have a pattern for a **SYSLOG** timestamp that is defined by patterns for month, day of the month, and time (for example, **Feb 1 06:25:43**). For this data, you might define the following pattern:

```
SYSLOGTIMESTAMP %{MONTH} + %{MONTHDAY} %{TIME}
```

Note

Grok patterns can process only one line at a time. Multiple-line patterns are not supported. Also, line breaks within a pattern are not supported.

Custom Classifier Values in AWS Glue

When you define a grok classifier, you supply the following values to AWS Glue to create the custom classifier.

Name

Name of the classifier.

Classification

The text string that is written to describe the format of the data that is classified; for example, **special-logs**.

Grok pattern

The set of patterns that are applied to the data store to determine whether there is a match. These patterns are from AWS Glue [built-in patterns \(p. 134\)](#) and any custom patterns that you define.

The following is an example of a grok pattern:

```
%{TIMESTAMP_ISO8601:timestamp} \[%{MESSAGEPREFIX:message_prefix}\]  
%{CRAWLERLOGLEVEL:loglevel} : %{GREEDYDATA:message}
```

When the data matches **TIMESTAMP_ISO8601**, a schema column **timestamp** is created. The behavior is similar for the other named patterns in the example.

Custom patterns

Optional custom patterns that you define. These patterns are referenced by the grok pattern that classifies your data. You can reference these custom patterns in the grok pattern that is applied to

your data. Each custom component pattern must be on a separate line. [Regular expression \(regex\)](#) syntax is used to define the pattern.

The following is an example of using custom patterns:

```
CRAWLERLOGLEVEL (BENCHMARK|ERROR|WARN|INFO|TRACE)  
MESSAGEPREFIX .*-.*-.*-.*-.*
```

The first custom named pattern, CRAWLERLOGLEVEL, is a match when the data matches one of the enumerated strings. The second custom pattern, MESSAGEPREFIX, tries to match a message prefix string.

AWS Glue keeps track of the creation time, last update time, and version of your classifier.

AWS Glue Built-In Patterns

AWS Glue provides many common patterns that you can use to build a custom classifier. You add a named pattern to the `grok_pattern` in a classifier definition.

The following list consists of a line for each pattern. In each line, the pattern name is followed its definition. [Regular expression \(regex\) syntax](#) is used in defining the pattern.

```

Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)) {3}))|:)))|(.+)?
IPV4 (?<![0-9])(?:(:25[0-5]|2[0-4][0-9])[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9])[0-1]?[0-9]{1,2})[.] (?>[0-9])(?:25[0-5]|2[0-4][0-9])[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9])[0-1]?[0-9]{1,2}) (?>[0-9])
IP (:%{IPV6:UNWANTED}|%{IPV4:UNWANTED})
HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z_]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z_-]{0,62}))*(\.\.|\b)
HOST %{HOSTNAME:UNWANTED}
IPORHOST (?%{HOSTNAME:UNWANTED}|%{IP:UNWANTED})
HOSTPORT (?%{IPORHOST}:%{POSINT:PORT})

# paths
PATH (?%{UNIXPATH}|%{WINPATH})
UNIXPATH (?>/(?>[\w_!$@:,~-]+|\.\.)*+
#UNIXPATH (?<![\w\//])(?:[^/\s?*]*+
TTY (?:/dev/(pts|tty([pq])))(\w+)?/?(:[0-9]+))
WINPATH (?>[A-Za-z]+:|\.\.)(?:\.\.\?*)+
URIPROTO [A-Za-z]+(\+[A-Za-z+])?
URIHOST %{IPORHOST}(:%{POSINT:port})?
# uripath comes loosely from RFC1738, but mostly from what Firefox
# doesn't turn into %XX
URIPATH (:/[A-Za-z0-9$.+!*'(){},-:=@#_\-\-]+
#URIPARAM \??:[A-Za-z0-9]+(:=(:[^&]*))?(?:&(:[A-Za-z0-9]+(:=(?:[^&]*))?)?)*)?
URIPARAM \?:[A-Za-z0-9$.+!*'(){},-@#%&=/:;_?\-\[\]]*
URIPATHPARAM %{URIPATH}(:%{URIPARAM})?
URI %{URIPROTO}://(:%{USER}(:^@*))?(@)?(:%{URIHOST})?(:%{URIPATHPARAM})?

# Months: January, Feb, 3, 03, 12, December
MONTH \b(?:Jan(?:uary)?|Feb(?:bruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\b
MONTHNUM (:0?[1-9]|1[0-2])
MONTHNUM2 (:0[1-9]|1[0-2])
MONTHDAY (:0[1-9])|(:12[0-9])|(:3[01])|[1-9])

# Days: Monday, Tue, Thu, etc...
DAY (:Mon(?:day)?|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day)?|Sat(?:urday)?|Sun(?:day)?)?

# Years?
YEAR (?>\d\d){1,2}
# Time: HH:MM:SS
#TIME \d{2}:\d{2}(:\d{2}(:\.\d+)?)?
# TIME %{POSINT<24}:%{POSINT<60}(:%{POSINT<60}(:\.\%{POSINT})?)?
HOUR (:2[0123]|[01]?[0-9])
MINUTE (:0-5[0-9])
# '60' is a leap second in most time standards and thus is valid.
SECOND (:0-5[0-9]|60)(:0-9)+)?
TIME (?![0-9])%{HOUR}:%{MINUTE}(:%{SECOND})(?!0-9])
# datestamp is YYYY/MM/DD-HH:MM:SS.UUUU (or something like it)
DATE_US %{MONTHNUM}[-] %{MONTHDAY}[-] %{YEAR}
DATE_EU %{MONTHDAY}[-] %{MONTHNUM}[-] %{YEAR}
DATESTAMP_US %{DATE_US}[-] %{TIME}
DATESTAMP_EU %{DATE_EU}[-] %{TIME}
ISO8601_TIMEZONE (:Z|[:-] %{HOUR}(:%{MINUTE}))
ISO8601_SECOND (:%{SECOND}|60)
TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:%{MINUTE}(:%{SECOND})?
%{ISO8601_TIMEZONE}?
TZ (:[PMCE][SD]T|UTC)
DATESTAMP_RFC822 %{DAY} %{MONTH} %{MONTHDAY} %{YEAR} %{TIME} %{TZ}
DATESTAMP_RFC2822 %{DAY}, %{MONTHDAY} %{MONTH} %{YEAR} %{TIME} %{ISO8601_TIMEZONE}
DATESTAMP_OTHER %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{TZ} %{YEAR}
DATESTAMP_EVENTLOG %{YEAR} %{MONTHNUM} %{MONTHDAY} %{HOUR} %{MINUTE} %{SECOND}
CISCOTIMESTAMP %{MONTH} %{MONTHDAY} %{TIME}

# Syslog Dates: Month Day HH:MM:SS

```

```

SYSLOGTIMESTAMP %{MONTH} + %{MONTHDAY} %{TIME}
PROG (?:[\w._/-]+)
SYSLOGPROG %{PROG:program}(?:\[%\{POSINT:pid\}\])?
SYSLOGHOST %{IPORHOST}
SYSLOGFACILITY <%{NONNEGINT:facility}.%{NONNEGINT:priority}>
HTTPDATE %{MONTHDAY}/%{MONTH}/%{YEAR}: %{TIME} %{INT}

# Shortcuts
QS %{QUOTEDSTRING:UNWANTED}

# Log formats
SYSLOGBASE %{SYSLOGTIMESTAMP:timestamp} (?:%{SYSLOGFACILITY} )?%{SYSLOGHOST:logsource}
%{SYSLOGPROG}:

MESSAGESLOG %{SYSLOGBASE} %{DATA}

COMMONAPACHELOG %{IPORHOST:clientip} %{USER:ident} %{USER:auth} \[%{HTTPDATE:timestamp}\]
"(?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest})"
%{NUMBER:response} (?:%{Bytes:bytes=%{NUMBER}}|-)
COMBINEDAPACHELOG %{COMMONAPACHELOG} %{QS:referrer} %{QS:agent}
COMMONAPACHELOG_DATATYPED %{IPORHOST:clientip} %{USER:ident:boolean} %{USER:auth}
\[%{HTTPDATE:timestamp};date;dd/MMM/yyyy:HH:mm:ss Z\] "(?:%{WORD:verb:string}
%{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion:float})?|%{DATA:rawrequest})"
%{NUMBER:response:int} (?:%{NUMBER:bytes:long})|-)

# Log Levels
LOGLEVEL ([A|a]lert|ALERT|[T|t]race|TRACE|[D|d]ebug|DEBUG|[N|n]otice|NOTICE|[I|i]nfo|
INFO|[W|w]arn(?:ing)?|WARN(?:ING)?|[E|e]rr(?:or)?|ERR(?:OR)?|[C|c]rit(?:ical)?|CRIT?|
?:ICAL)?|[F|f]atal|FATAL|[S|s]evere|SEVERE|EMERG(?:ENCY)?|[Ee]merg(?:ency)?)"

```

Writing XML Custom Classifiers

XML defines the structure of a document with the use of tags in the file. With an XML custom classifier, you can specify the tag name used to define a row.

Custom Classifier Values in AWS Glue

When you define an XML classifier, you supply the following values to AWS Glue to create the classifier. The classification field of this classifier is set to `xml`.

Name

Name of the classifier.

Row tag

The XML tag name that defines a table row in the XML document, without angle brackets `< >`. The name must comply with XML rules for a tag.

Note

The element containing the row data **cannot** be a self-closing empty element. For example, this empty element is **not** parsed by AWS Glue:

```
<row att1="xx" att2="yy" />
```

Empty elements can be written as follows:

```
<row att1="xx" att2="yy"> </row>
```

AWS Glue keeps track of the creation time, last update time, and version of your classifier.

For example, suppose that you have the following XML file. To create an AWS Glue table that only contains columns for author and title, create a classifier in the AWS Glue console with **Row tag** as AnyCompany. Then add and run a crawler that uses this custom classifier.

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <AnyCompany>
      <author>Rivera, Martha</author>
      <title>AnyCompany Developer Guide</title>
    </AnyCompany>
  </book>
  <book id="bk102">
    <AnyCompany>
      <author>Stiles, John</author>
      <title>Style Guide for AnyCompany</title>
    </AnyCompany>
  </book>
</catalog>
```

Writing JSON Custom Classifiers

JSON is a data-interchange format. It defines data structures with name-value pairs or an ordered list of values. With a JSON custom classifier, you can specify the JSON path to a data structure that is used to define the schema for your table.

Custom Classifier Values in AWS Glue

When you define a JSON classifier, you supply the following values to AWS Glue to create the classifier. The classification field of this classifier is set to json.

Name

Name of the classifier.

JSON path

A JSON path that points to an object that is used to define a table schema. The JSON path can be written in dot notation or bracket notation. The following operators are supported:

Description
\$Root element of a JSON object. This starts all path expressions
*Wildcard character. Available anywhere a name or numeric are required in the JSON path.
.Dot-notation child. Specifies a child field in a JSON object.
[Bracket-notation child. Specifies child field in a JSON object. Only a single child field can be specified.]
[Array-index.]Specifies the value of an array by index.

AWS Glue keeps track of the creation time, last update time, and version of your classifier.

Example of Using a JSON Classifier to Pull Records from an Array

Suppose that your JSON data is an array of records. For example, the first few lines of your file might look like the following:

```
[  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:ak",  
    "name": "Alaska"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:1",  
    "name": "Alabama's 1st congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:2",  
    "name": "Alabama's 2nd congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:3",  
    "name": "Alabama's 3rd congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:4",  
    "name": "Alabama's 4th congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:5",  
    "name": "Alabama's 5th congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:6",  
    "name": "Alabama's 6th congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:al\\cd:7",  
    "name": "Alabama's 7th congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:ar\\cd:1",  
    "name": "Arkansas's 1st congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:ar\\cd:2",  
    "name": "Arkansas's 2nd congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:ar\\cd:3",  
    "name": "Arkansas's 3rd congressional district"  
  },  
  {  
    "type": "constituency",  
    "id": "ocd-division\\country:us\\state:ar\\cd:4",  
    "name": "Arkansas's 4th congressional district"  
  }]
```

```
        "name": "Arkansas's 4th congressional district"
    }
]
```

When you run a crawler using the built-in JSON classifier, the entire file is used to define the schema. Because you don't specify a JSON path, the crawler treats the data as one object, that is, just an array. For example, the schema might look like the following:

```
root
|-- record: array
```

However, to create a schema that is based on each record in the JSON array, create a custom JSON classifier and specify the JSON path as `$[*]`. When you specify this JSON path, the classifier interrogates all 12 records in the array to determine the schema. The resulting schema contains separate fields for each object, similar to the following example:

```
root
|-- type: string
|-- id: string
|-- name: string
```

Example of Using a JSON Classifier to Examine Only Parts of a File

Suppose that your JSON data follows the pattern of the example JSON file `s3://awsglue-datasets/examples/us-legislators/all/areas.json` drawn from <http://everypolitician.org/>. Example objects in the JSON file look like the following:

```
{
  "type": "constituency",
  "id": "ocd-division\\country:us\\state:ak",
  "name": "Alaska"
}
{
  "type": "constituency",
  "identifiers": [
    {
      "scheme": "dmoz",
      "identifier": "Regional\\North_America\\United_States\\Alaska\\"
    },
    {
      "scheme": "freebase",
      "identifier": "\\m\\0hjy"
    },
    {
      "scheme": "fips",
      "identifier": "US02"
    },
    {
      "scheme": "quora",
      "identifier": "Alaska-state"
    },
    {
      "scheme": "britannica",
      "identifier": "place\\Alaska"
    },
    {
      "scheme": "wikidata",
      "identifier": "Q797"
    }
}
```

```

        },
        ],
        "other_names": [
        {
        "lang": "en",
        "note": "multilingual",
        "name": "Alaska"
        },
        {
        "lang": "fr",
        "note": "multilingual",
        "name": "Alaska"
        },
        {
        "lang": "nov",
        "note": "multilingual",
        "name": "Alaska"
        }
        ],
        "id": "ocd-division\\country:us\\state:ak",
        "name": "Alaska"
    }
}

```

When you run a crawler using the built-in JSON classifier, the entire file is used to create the schema. You might end up with a schema like this:

```

root
|-- type: string
|-- id: string
|-- name: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string

```

However, to create a schema using just the "id" object, create a custom JSON classifier and specify the JSON path as \$.id. Then the schema is based on only the "id" field:

```

root
|-- record: string

```

The first few lines of data extracted with this schema look like this:

```

{"record": "ocd-division/country:us/state:ak"}
{"record": "ocd-division/country:us/state:al/cd:1"}
{"record": "ocd-division/country:us/state:al/cd:2"}
{"record": "ocd-division/country:us/state:al/cd:3"}
{"record": "ocd-division/country:us/state:al/cd:4"}
 {"record": "ocd-division/country:us/state:al/cd:5"}
 {"record": "ocd-division/country:us/state:al/cd:6"}
 {"record": "ocd-division/country:us/state:al/cd:7"}
 {"record": "ocd-division/country:us/state:ar/cd:1"}
 {"record": "ocd-division/country:us/state:ar/cd:2"}

```

```
{"record": "ocd-division/country:us/state:ar/cd:3"}  
{"record": "ocd-division/country:us/state:ar/cd:4"}  
{"record": "ocd-division/country:us/state:as"}  
{"record": "ocd-division/country:us/state:az/cd:1"}  
{"record": "ocd-division/country:us/state:az/cd:2"}  
{"record": "ocd-division/country:us/state:az/cd:3"}  
{"record": "ocd-division/country:us/state:az/cd:4"}  
{"record": "ocd-division/country:us/state:az/cd:5"}  
{"record": "ocd-division/country:us/state:az/cd:6"}  
{"record": "ocd-division/country:us/state:az/cd:7"}
```

To create a schema based on a deeply nested object, such as "identifier," in the JSON file, you can create a custom JSON classifier and specify the JSON path as `$.identifiers[*].identifier`. Although the schema is similar to the previous example, it is based on a different object in the JSON file.

The schema looks like the following:

```
root  
|-- record: string
```

Listing the first few lines of data from the table shows that the schema is based on the data in the "identifier" object:

```
{"record": "Regional/North_America/United_States/Alaska/"}  
{"record": "/m/0hjy"}  
{"record": "US02"}  
{"record": "5879092"}  
{"record": "4001016-8"}  
{"record": "destination/alaska"}  
{"record": "1116270"}  
{"record": "139487266"}  
{"record": "n79018447"}  
{"record": "01490999-8dec-4129-8254-eef6e80fadcc3"}  
{"record": "Alaska-state"}  
{"record": "place/Alaska"}  
{"record": "Q797"}  
{"record": "Regional/North_America/United_States/Alabama/"}  
{"record": "/m/0gyh"}  
{"record": "US01"}  
{"record": "4829764"}  
{"record": "4084839-5"}  
{"record": "161950"}  
{"record": "131885589"}
```

To create a table based on another deeply nested object, such as the "name" field in the "other_names" array in the JSON file, you can create a custom JSON classifier and specify the JSON path as `$.other_names[*].name`. Although the schema is similar to the previous example, it is based on a different object in the JSON file. The schema looks like the following:

```
root  
|-- record: string
```

Listing the first few lines of data in the table shows that it is based on the data in the "name" object in the "other_names" array:

```
{"record": "Alaska"}
```

```
{"record": "Alaska"}  
{"record": "#####"}  
{"record": "Alaska"}  
{"record": "Alyaska"}  
{"record": "Alaska"}  
{"record": "Alaska"}  
{"record": "#### #####"}  
{"record": "#####"}  
{"record": "Alaska"}  
{"record": "#####"}  
{"record": "#####"}
```

Writing CSV Custom Classifiers

You can use a custom CSV classifier to infer the schema of various types of CSV data. The custom attributes that you can provide for your classifier include delimiters, options about the header, and whether to perform certain validations on the data.

Custom Classifier Values in AWS Glue

When you define a CSV classifier, you provide the following values to AWS Glue to create the classifier. The classification field of this classifier is set to `csv`.

Name

Name of the classifier.

Column delimiter

A custom symbol to denote what separates each column entry in the row.

Quote symbol

A custom symbol to denote what combines content into a single column value. Must be different from the column delimiter.

Column headings

Indicates the behavior for how column headings should be detected in the CSV file. If your custom CSV file has column headings, enter a comma-delimited list of the column headings.

Processing options: Allow files with single column

Enables the processing of files that contain only one column.

Processing options: Trim white space before identifying column values

Specifies whether to trim values before identifying the type of column values.

Working with Classifiers on the AWS Glue Console

A classifier determines the schema of your data. You can write a custom classifier and point to it from AWS Glue. To see a list of all the classifiers that you have created, open the AWS Glue console at <https://console.aws.amazon.com/glue/>, and choose the **Classifiers** tab.

The list displays the following properties about each classifier:

Classifier

The classifier name. When you create a classifier, you must provide a name for it.

Classification

The classification type of tables inferred by this classifier.

Last updated

The last time this classifier was updated.

From the **Classifiers** list in the AWS Glue console, you can add, edit, and delete classifiers. To see more details for a classifier, choose the classifier name in the list. Details include the information you defined when you created the classifier.

To add a classifier in the AWS Glue console, choose **Add classifier**. When you define a classifier, you supply values for the following:

Classifier name

Provide a unique name for your classifier.

Classification

For grok classifiers, describe the format or type of data that is classified or provide a custom label.

Grok pattern

For grok classifiers, this is used to parse your data into a structured schema. The grok pattern is composed of named patterns that describe the format of your data store. You write this grok pattern using the named built-in patterns provided by AWS Glue and custom patterns you write and include in the **Custom patterns** field. Although grok debugger results might not match the results from AWS Glue exactly, we suggest that you try your pattern using some sample data with a grok debugger. You can find grok debuggers on the web. The named built-in patterns provided by AWS Glue are generally compatible with grok patterns that are available on the web.

Build your grok pattern by iteratively adding named patterns and check your results in a debugger. This activity gives you confidence that when the AWS Glue crawler runs your grok pattern, your data can be parsed.

Custom patterns

For grok classifiers, these are optional building blocks for the **Grok pattern** that you write. When built-in patterns cannot parse your data, you might need to write a custom pattern. These custom patterns are defined in this field and referenced in the **Grok pattern** field. Each custom pattern is defined on a separate line. Just like the built-in patterns, it consists of a named pattern definition that uses [regular expression \(regex\)](#) syntax.

For example, the following has the name MESSAGEPREFIX followed by a regular expression definition to apply to your data to determine whether it follows the pattern.

```
MESSAGEPREFIX .*-.*-.*-.*-.*
```

Row tag

For XML classifiers, this is the name of the XML tag that defines a table row in the XML document. Type the name without angle brackets < >. The name must comply with XML rules for a tag.

JSON path

For JSON classifiers, this is the JSON path to the object, array, or value that defines a row of the table being created. Type the name in either dot or bracket JSON syntax using AWS Glue

supported operators. For more information, see the list of operators in [Writing JSON Custom Classifiers \(p. 137\)](#).

For more information, see [Writing Custom Classifiers \(p. 132\)](#).

Working with Data Catalog Settings on the AWS Glue Console

The Data Catalog settings page contains options to set properties for the Data Catalog in your account.

To change the fine-grained access control of the Data Catalog

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Settings**, and then in the **Permissions** editor, add the policy statement to change fine-grained access control of the Data Catalog for your account. Only one policy at a time can be attached to a Data Catalog.
3. Choose **Save** to update your Data Catalog with any changes you made.

You can also use AWS Glue API operations to put, get, and delete resource policies. For more information, see [Security APIs in AWS Glue \(p. 459\)](#).

The **Settings** page displays the following options:

Metadata encryption

Select this check box to encrypt the metadata in your Data Catalog. Metadata is encrypted at rest using the AWS Key Management Service (AWS KMS) key that you specify.

Important

AWS Glue supports only symmetric customer master keys (CMKs). The **AWS KMS key list** displays only symmetric keys. However, if you select **Choose a KMS key ARN**, the console lets you enter an ARN for any key type. Ensure that you enter only ARNs for symmetric keys.

Encrypt connection passwords

Select this check box to encrypt passwords in the AWS Glue connection object when the connection is created or updated. Passwords are encrypted using the AWS KMS key that you specify. When passwords are returned, they are encrypted. This option is a global setting for all AWS Glue connections in the Data Catalog. If you clear this check box, previously encrypted passwords remain encrypted using the key that was used when they were created or updated. For more information about AWS Glue connections, see [Adding a Connection to Your Data Store \(p. 110\)](#).

When you enable this option, choose an AWS KMS key, or choose **Enter a key ARN** and provide the Amazon Resource Name (ARN) for the key. Enter the ARN in the form `arn:aws:kms:region:account-id:key/key-id`. You can also provide the ARN as a key alias, such as `arn:aws:kms:region:account-id:alias/alias-name`.

Important

If this option is selected, any user or role that creates or updates a connection must have `kms:Encrypt` permission on the specified KMS key.

Permissions

Add a resource policy to define fine-grained access control of the Data Catalog. You can paste a JSON resource policy into this control. For more information, see [Resource Policies \(p. 59\)](#).

Updating the Data Catalog with New Partitions

If your extract, transfer, and load (ETL) job creates new table partitions in the target data store, and you want to view the new partitions in the AWS Glue Data Catalog, you can do one of the following:

- When the job finishes, rerun the crawler, and view the new partitions on the console when the crawler finishes.
- When the job finishes, view the new partitions on the console right away, without having to rerun the crawler. You can enable this feature by adding a few lines of code to your ETL script, as shown in the following examples. The code uses the `enableUpdateCatalog` argument to indicate that the Data Catalog is to be updated during the job run as the new partitions are created. This capability is supported for Amazon Simple Storage Service (Amazon S3) targets only, and only for the `json`, `csv`, `avro`, `parquet`, and `orc` formats.

Method 1

Pass `enableUpdateCatalog` and `partitionKeys` in an options argument.

Python

```
additionalOptions = {"enableUpdateCatalog": True}
additionalOptions["partitionKeys"] = ["region", "year", "month", "day"]

sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
    database=<target_db_name>,                                              table_name=<target_table_name>,
    transformation_ctx="write_sink",
    additional_options=additionalOptions)
```

Scala

```
val options = JsonOptions(Map("path" -> <S3_output_path>, "partitionKeys" ->
  Seq("region", "year", "month", "day"), "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(database = <target_db_name>, tableName
  = <target_table_name>, additionalOptions = options)
sink.writeDynamicFrame(df)
```

Method 2

Pass `enableUpdateCatalog` and `partitionKeys` in `getSink()`, and call `setCatalogInfo()` on the `DataSink` object.

Python

```
sink = glueContext.getSink(connection_type="s3", path=<S3_output_path>,
                           enableUpdateCatalog=True,
                           partitionKeys=["region", "year", "month", "day"])
sink.setFormat("json")
sink.setCatalogInfo(catalogDatabase=<target_db_name>,
                    catalogTableName=<target_table_name>)
sink.writeFrame(last_transform)
```

Scala

```
val options = JsonOptions(Map("path" -> <S3_output_path>, "partitionKeys" ->
  Seq("region", "year", "month", "day"), "enableUpdateCatalog" -> true))
```

```
val sink = glueContext.getSink("s3", options).withFormat("json")
sink.setCatalogInfo(<target_db_name>, <target_table_name>)
sink.writeDynamicFrame(df)
```

For more information, see [ETL Programming \(p. 292\)](#).

Populating the Data Catalog Using AWS CloudFormation Templates

AWS CloudFormation is a service that can create many AWS resources. AWS Glue provides API operations to create objects in the AWS Glue Data Catalog. However, it might be more convenient to define and create AWS Glue objects and other related AWS resource objects in an AWS CloudFormation template file. Then you can automate the process of creating the objects.

AWS CloudFormation provides a simplified syntax—either JSON (JavaScript Object Notation) or YAML (YAML Ain't Markup Language)—to express the creation of AWS resources. You can use AWS CloudFormation templates to define Data Catalog objects such as databases, tables, partitions, crawlers, classifiers, and connections. You can also define ETL objects such as jobs, triggers, and development endpoints. You create a template that describes all the AWS resources you want, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

For more information, see [What Is AWS CloudFormation?](#) and [Working with AWS CloudFormation Templates](#) in the *AWS CloudFormation User Guide*.

If you plan to use AWS CloudFormation templates that are compatible with AWS Glue, as an administrator, you must grant access to AWS CloudFormation and to the AWS services and actions on which it depends. To grant permissions to create AWS CloudFormation resources, attach the following policy to the IAM users that work with AWS CloudFormation:

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

The following table contains the actions that an AWS CloudFormation template can perform on your behalf. It includes links to information about the AWS resource types and their property types that you can add to an AWS CloudFormation template.

AWS Glue Resource	AWS CloudFormation Template	AWS Glue Samples
Classifier	AWS::Glue::Classifier	Grok classifier (p. 151) , JSON classifier (p. 151) , XML classifier (p. 152)
Connection	AWS::Glue::Connection	MySQL connection (p. 154)

AWS Glue Resource	AWS CloudFormation Template	AWS Glue Samples
Crawler	AWS::Glue::Crawler	Amazon S3 crawler (p. 153) , MySQL crawler (p. 155)
Database	AWS::Glue::Database	Empty database (p. 147) , Database with tables (p. 148)
Development endpoint	AWS::Glue::DevEndpoint	Development endpoint (p. 161)
Job	AWS::Glue::Job	Amazon S3 job (p. 157) , JDBC job (p. 158)
Partition	AWS::Glue::Partition	Partitions of a table (p. 148)
Table	AWS::Glue::Table	Table in a database (p. 148)
Trigger	AWS::Glue::Trigger	On-demand trigger (p. 159) , Scheduled trigger (p. 160) , Conditional trigger (p. 161)

To get started, use the following sample templates and customize them with your own metadata. Then use the AWS CloudFormation console to create an AWS CloudFormation stack to add objects to AWS Glue and any associated services. Many fields in an AWS Glue object are optional. These templates illustrate the fields that are required or are necessary for a working and functional AWS Glue object.

An AWS CloudFormation template can be in either JSON or YAML format. In these examples, YAML is used for easier readability. The examples contain comments (#) to describe the values that are defined in the templates.

AWS CloudFormation templates can include a `Parameters` section. This section can be changed in the sample text or when the YAML file is submitted to the AWS CloudFormation console to create a stack. The `Resources` section of the template contains the definition of AWS Glue and related objects. AWS CloudFormation template syntax definitions might contain properties that include more detailed property syntax. Not all properties might be required to create an AWS Glue object. These samples show example values for common properties to create an AWS Glue object.

Sample AWS CloudFormation Template for an AWS Glue Database

An AWS Glue database in the Data Catalog contains metadata tables. The database consists of very few properties and can be created in the Data Catalog with an AWS CloudFormation template. The following sample template is provided to get you started and to illustrate the use of AWS CloudFormation stacks with AWS Glue. The only resource created by the sample template is a database named `cfn-mysampledatabase`. You can change it by editing the text of the sample or changing the value on the AWS CloudFormation console when you submit the YAML.

The following shows example values for common properties to create an AWS Glue database. For more information about the AWS CloudFormation database template for AWS Glue, see [AWS::Glue::Database](#).

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database named
mysampledatabase
# The metadata created in the Data Catalog points to the flights public S3 bucket
```

```

#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-mysampledatabase

# Resources section defines metadata for the Data Catalog
Resources:
# Create an AWS Glue database
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      # The database is created in the Data Catalog for your account
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        # The name of the database is defined in the Parameters section above
        Name: !Ref CFNDatabaseName
        Description: Database to hold tables for flights data
        LocationUri: s3://crawler-public-us-east-1/flight/2016/csv/
      #Parameters: Leave AWS database parameters blank

```

Sample AWS CloudFormation Template for an AWS Glue Database, Table, and Partition

An AWS Glue table contains the metadata that defines the structure and location of data that you want to process with your ETL scripts. Within a table, you can define partitions to parallelize the processing of your data. A partition is a chunk of data that you defined with a key. For example, using month as a key, all the data for January is contained in the same partition. In AWS Glue, databases can contain tables, and tables can contain partitions.

The following sample shows how to populate a database, a table, and partitions using an AWS CloudFormation template. The base data format is csv and delimited by a comma (,). Because a database must exist before it can contain a table, and a table must exist before partitions can be created, the template uses the `DependsOn` statement to define the dependency of these objects when they are created.

The values in this sample define a table that contains flight data from a publicly available Amazon S3 bucket. For illustration, only a few columns of the data and one partitioning key are defined. Four partitions are also defined in the Data Catalog. Some fields to describe the storage of the base data are also shown in the `StorageDescriptor` fields.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database, a table, and
# partitions
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters substituted in the Resources section
# These parameters are names of the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1
  CFNTableName1:
    Type: String
    Default: cfn-manual-table-flights-1
# Resources to create metadata in the Data Catalog

```

```

Resources:
#####
# Create an AWS Glue database
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: Database to hold tables for flights data
#####
# Create an AWS Glue table
CFNTableFlights:
  # Creating the table waits for the database to be created
  DependsOn: CFNDatabaseFlights
  Type: AWS::Glue::Table
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableInput:
      Name: !Ref CFNTableName1
      Description: Define the first few columns of the flights table
      TableType: EXTERNAL_TABLE
      Parameters: {
        "classification": "csv"
      }
    #
    ViewExpandedText: String
    PartitionKeys:
      # Data is partitioned by month
      - Name: mon
        Type: bigint
    StorageDescriptor:
      OutputFormat: org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat
      Columns:
        - Name: year
          Type: bigint
        - Name: quarter
          Type: bigint
        - Name: month
          Type: bigint
        - Name: day_of_month
          Type: bigint
      InputFormat: org.apache.hadoop.mapred.TextInputFormat
      Location: s3://crawler-public-us-east-1/flight/2016/csv/
      SerdeInfo:
        Parameters:
          field.delim: ","
        SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 1
# Create an AWS Glue partition
CFNPartitionMon1:
  DependsOn: CFNTableFlights
  Type: AWS::Glue::Partition
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
      Values:
        - 1
    StorageDescriptor:
      OutputFormat: org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat
      Columns:
        - Name: mon
          Type: bigint
      InputFormat: org.apache.hadoop.mapred.TextInputFormat

```

```

        Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=1/
        SerdeInfo:
            Parameters:
                field.delim: ","
                SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 2
# Create an AWS Glue partition
CFNPartitionMon2:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
        Values:
        - 2
        StorageDescriptor:
            OutputFormat: org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat
            Columns:
            - Name: mon
              Type: bigint
            InputFormat: org.apache.hadoop.mapred.TextInputFormat
            Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=2/
            SerdeInfo:
                Parameters:
                    field.delim: ","
                    SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 3
# Create an AWS Glue partition
CFNPartitionMon3:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
        Values:
        - 3
        StorageDescriptor:
            OutputFormat: org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat
            Columns:
            - Name: mon
              Type: bigint
            InputFormat: org.apache.hadoop.mapred.TextInputFormat
            Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=3/
            SerdeInfo:
                Parameters:
                    field.delim: ","
                    SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 4
# Create an AWS Glue partition
CFNPartitionMon4:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
        Values:
        - 4
        StorageDescriptor:
            OutputFormat: org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat
            Columns:

```

```
- Name: mon
  Type: bigint
InputFormat: org.apache.hadoop.mapred.TextInputFormat
Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=4/
SerdeInfo:
  Parameters:
    field.delim: ","
SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```

Sample AWS CloudFormation Template for an AWS Glue Grok Classifier

An AWS Glue classifier determines the schema of your data. One type of custom classifier uses a grok pattern to match your data. If the pattern matches, then the custom classifier is used to create your table's schema and set the classification to the value set in the classifier definition.

This sample creates a classifier that creates a schema with one column named `message` and sets the classification to `greedy`.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-grok-one-column-1

  #
  #
# Resources section defines metadata for the Data Catalog
Resources:
  # Create classifier that uses grok pattern to put all data in one column and classifies it
  # as "greedy".
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      GrokClassifier:
        #Grok classifier that puts all data in one column
        Name: !Ref CFNClassifierName
        Classification: greedy
        GrokPattern: "%{GREEDYDATA:message}"
        #CustomPatterns: none
```

Sample AWS CloudFormation Template for an AWS Glue JSON Classifier

An AWS Glue classifier determines the schema of your data. One type of custom classifier uses a `JsonPath` string defining the JSON data for the classifier to classify. AWS Glue supports a subset of the operators for `JsonPath`, as described in [Writing JsonPath Custom Classifiers](#).

If the pattern matches, then the custom classifier is used to create your table's schema.

This sample creates a classifier that creates a schema with each record in the `Records3` array in an object.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a JSON classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
CFNClassifierName:
  Type: String
  Default: cfn-classifier-json-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses a JSON pattern.
CFNClassifierFlights:
  Type: AWS::Glue::Classifier
  Properties:
    JSONClassifier:
      #JSON classifier
      Name: !Ref CFNClassifierName
      JsonPath: $.Records3[*]
```

Sample AWS CloudFormation Template for an AWS Glue XML Classifier

An AWS Glue classifier determines the schema of your data. One type of custom classifier specifies an XML tag to designate the element that contains each record in an XML document that is being parsed. If the pattern matches, then the custom classifier is used to create your table's schema and set the classification to the value set in the classifier definition.

This sample creates a classifier that creates a schema with each record in the `Record` tag and sets the classification to `XML`.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an XML classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
CFNClassifierName:
  Type: String
  Default: cfn-classifier-xml-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses the XML pattern and classifies it as "XML".
CFNClassifierFlights:
```

```
Type: AWS::Glue::Classifier
Properties:
  XMLClassifier:
    #XML classifier
    Name: !Ref CFNClassifierName
    Classification: XML
    RowTag: <Records>
```

Sample AWS CloudFormation Template for an AWS Glue Crawler for Amazon S3

An AWS Glue crawler creates metadata tables in your Data Catalog that correspond to your data. You can then use these table definitions as sources and targets in your ETL jobs.

This sample creates a crawler, the required IAM role, and an AWS Glue database in the Data Catalog. When this crawler is run, it assumes the IAM role and creates a table in the database for the public flights data. The table is created with the prefix "cfn_sample_1_". The IAM role created by this template allows global permissions; you might want to create a custom role. No custom classifiers are defined by this classifier. AWS Glue built-in classifiers are used by default.

When you submit this sample to the AWS CloudFormation console, you must confirm that you want to create the IAM role.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
CFNCrawlerName:
  Type: String
  Default: cfn-crawler-flights-1
CFNDatabaseName:
  Type: String
  Default: cfn-database-flights-1
CFNTablePrefixName:
  Type: String
  Default: cfn_sample_1_
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
permissions.
CFNRoleFlights:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "glue.amazonaws.com"
          Action:
            - "sts:AssumeRole"
```

```

Path: "/"
Policies:
-
  PolicyName: "root"
  PolicyDocument:
    Version: "2012-10-17"
    Statement:
    -
      Effect: "Allow"
      Action: "*"
      Resource: "*"
# Create a database to contain tables created by the crawler
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: "AWS Glue container to hold metadata tables for the flights crawler"
#Create a crawler to crawl the flights data on a public S3 bucket
CFNCrawlerFlights:
  Type: AWS::Glue::Crawler
  Properties:
    Name: !Ref CFNCrawlerName
    Role: !GetAtt CFNRoleFlights.Arn
    #Classifiers: none, use the default classifier
    Description: AWS Glue crawler to crawl flights data
    #Schedule: none, use default run-on-demand
    DatabaseName: !Ref CFNDatabaseName
    Targets:
      S3Targets:
        # Public S3 bucket with the flights data
        - Path: "s3://crawler-public-us-east-1/flight/2016/csv"
    TablePrefix: !Ref CFNTablePrefixName
    SchemaChangePolicy:
      UpdateBehavior: "UPDATE_IN_DATABASE"
      DeleteBehavior: "LOG"
    Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":{},\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":\"MergeNewColumns\"}}}"
  
```

Sample AWS CloudFormation Template for an AWS Glue Connection

An AWS Glue connection in the Data Catalog contains the JDBC and network information that is required to connect to a JDBC database. This information is used when you connect to a JDBC database to crawl or run ETL jobs.

This sample creates a connection to an Amazon RDS MySQL database named devdb. When this connection is used, an IAM role, database credentials, and network connection values must also be supplied. See the details of necessary fields in the template.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a connection
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The name of the connection to be created
  
```

```

CFNConnectionName:
  Type: String
  Default: cfn-connection-mysql-flights-1
CFNJDBCString:
  Type: String
  Default: "jdbc:mysql://xxx-mysql.yyyyyyyyyyyyyy.us-east-1.rds.amazonaws.com:3306/devdb"
CFNJDBCUser:
  Type: String
  Default: "master"
CFNJDBCPassword:
  Type: String
  Default: "12345678"
  NoEcho: true
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNConnectionMySQL:
    Type: AWS::Glue::Connection
    Properties:
      CatalogId: !Ref AWS::AccountId
      ConnectionInput:
        Description: "Connect to MySQL database."
        ConnectionType: "JDBC"
        #MatchCriteria: none
      PhysicalConnectionRequirements:
        AvailabilityZone: "us-east-1d"
        SecurityGroupIdList:
          - "sg-7d52b812"
        SubnetId: "subnet-84f326ee"
      ConnectionProperties: {
        "JDBC_CONNECTION_URL": !Ref CFNJDBCString,
        "USERNAME": !Ref CFNJDBCUser,
        "PASSWORD": !Ref CFNJDBCPassword
      }
      Name: !Ref CFNConnectionName

```

Sample AWS CloudFormation Template for an AWS Glue Crawler for JDBC

An AWS Glue crawler creates metadata tables in your Data Catalog that correspond to your data. You can then use these table definitions as sources and targets in your ETL jobs.

This sample creates a crawler, required IAM role, and an AWS Glue database in the Data Catalog. When this crawler is run, it assumes the IAM role and creates a table in the database for the public flights data that has been stored in a MySQL database. The table is created with the prefix "cfn_jdbc_1_". The IAM role created by this template allows global permissions; you might want to create a custom role. No custom classifiers can be defined for JDBC data. AWS Glue built-in classifiers are used by default.

When you submit this sample to the AWS CloudFormation console, you must confirm that you want to create the IAM role.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

```

```

# The name of the crawler to be created
CFNCrawlerName:
  Type: String
  Default: cfn-crawler-jdbc-flights-1
# The name of the database to be created to contain tables
CFNDatabaseName:
  Type: String
  Default: cfn-database-jdbc-flights-1
# The prefix for all tables crawled and created
CFNTablePrefixName:
  Type: String
  Default: cfn_jdbc_1_
# The name of the existing connection to the MySQL database
CFNConnectionName:
  Type: String
  Default: cfn-connection-mysql-flights-1
# The name of the JDBC path (database/schema/table) with wildcard (%) to crawl
CFNJDBCPPath:
  Type: String
  Default: saldev/%
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
permissions.
CFNRoleFlights:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "glue.amazonaws.com"
          Action:
            - "sts:AssumeRole"
      Path: "/"
      Policies:
        -
          PolicyName: "root"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action: "*"
                Resource: "*"
# Create a database to contain tables created by the crawler
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: "AWS Glue container to hold metadata tables for the flights crawler"
#Create a crawler to crawl the flights data in MySQL database
CFNCrawlerFlights:
  Type: AWS::Glue::Crawler
  Properties:
    Name: !Ref CFNCrawlerName
    Role: !GetAtt CFNRoleFlights.Arn
    #Classifiers: none, use the default classifier
    Description: AWS Glue crawler to crawl flights data

```

```

#Schedule: none, use default run-on-demand
DatabaseName: !Ref CFNDatabaseName
Targets:
  JdbcTargets:
    # JDBC MySQL database with the flights data
    - ConnectionName: !Ref CFNConnectionName
      Path: !Ref CFNJDBCPath
    #Exclusions: none
TablePrefix: !Ref CFNTablePrefixName
SchemaChangePolicy:
  UpdateBehavior: "UPDATE_IN_DATABASE"
  DeleteBehavior: "LOG"
Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":{},\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":\"MergeNewColumns\"}}}"

```

Sample AWS CloudFormation Template for an AWS Glue Job for Amazon S3 to Amazon S3

An AWS Glue job in the Data Catalog contains the parameter values that are required to run a script in AWS Glue.

This sample creates a job that reads flight data from an Amazon S3 bucket in csv format and writes it to an Amazon S3 Parquet file. The script that is run by this job must already exist. You can generate an ETL script for your environment with the AWS Glue console. When this job is run, an IAM role with the correct permissions must also be supplied.

Common parameter values are shown in the template. For example, `AllocatedCapacity (DPUs)` defaults to 5.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using the public flights S3 table in a
public bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-2
  # The name of the IAM role that the job assumes. It must have access to data, script,
  temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
  # The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-test2
  #
  #
  # Resources section defines metadata for the Data Catalog
Resources:
  # Create job to run script which accesses flightscsv table and write to S3 file as parquet.
  # The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job

```

```

Properties:
  Role: !Ref CFNIAMRoleName
  #DefaultArguments: JSON object
  # If script written in Scala, then set DefaultArguments={'--job-language': 'scala',
'--class': 'your scala class'}
  #Connections: No connection needed for S3 to S3 job
  # ConnectionsList
  #MaxRetries: Double
  Description: Job created with CloudFormation
  #LogUri: String
  Command:
    Name: glueetl
    ScriptLocation: !Ref CFNScriptLocation
      # for access to directories use proper IAM role with permission to buckets and
      # folders that begin with "aws-glue-"
      # script uses temp directory from job definition if required (temp directory
      # not used S3 to S3)
      # script defines target for output as s3://aws-glue-target/sal
    AllocatedCapacity: 5
    ExecutionProperty:
      MaxConcurrentRuns: 1
  Name: !Ref CFNJobName

```

Sample AWS CloudFormation Template for an AWS Glue Job for JDBC to Amazon S3

An AWS Glue job in the Data Catalog contains the parameter values that are required to run a script in AWS Glue.

This sample creates a job that reads flight data from a MySQL JDBC database as defined by the connection named cfn-connection-mysql-flights-1 and writes it to an Amazon S3 Parquet file. The script that is run by this job must already exist. You can generate an ETL script for your environment with the AWS Glue console. When this job is run, an IAM role with the correct permissions must also be supplied.

Common parameter values are shown in the template. For example, `AllocatedCapacity` (DPUs) defaults to 5.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using a MySQL JDBC DB with the flights data
# to an S3 file
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
CFNJobName:
  Type: String
  Default: cfn-job-JDBC-to-S3-1
# The name of the IAM role that the job assumes. It must have access to data, script,
# temporary directory
CFNIAMRoleName:
  Type: String
  Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
CFNScriptLocation:
  Type: String
  Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-dec4a

```

```

# The name of the connection used for JDBC data source
CFNConnectionName:
  Type: String
  Default: cfn-connection-mysql-flights-1
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses JDBC flights table via a connection and write to
S3 file as parquet.
# The script already exists and is called by this job
CFNJobFlights:
  Type: AWS::Glue::Job
  Properties:
    Role: !Ref CFNIAMRoleName
    #DefaultArguments: JSON object
    # For example, if required by script, set temporary directory as
DefaultArguments='{"TempDir": "s3://aws-glue-temporary-xyc/sal"}'
    Connections:
      Connections:
        - !Ref CFNConnectionName
    #MaxRetries: Double
    Description: Job created with CloudFormation using existing script
    #LogUri: String
    Command:
      Name: glueetl
      ScriptLocation: !Ref CFNScriptLocation
        # for access to directories use proper IAM role with permission to buckets and
folders that begin with "aws-glue-"
        # if required, script defines temp directory as argument TempDir and used in
script like redshift_tmp_dir = args["TempDir"]
        # script defines target for output as s3://aws-glue-target/sal
    AllocatedCapacity: 5
    ExecutionProperty:
      MaxConcurrentRuns: 1
    Name: !Ref CFNJobName
  
```

Sample AWS CloudFormation Template for an AWS Glue On-Demand Trigger

An AWS Glue trigger in the Data Catalog contains the parameter values that are required to start a job run when the trigger fires. An on-demand trigger fires when you enable it.

This sample creates an on-demand trigger that starts one job named `cfn-job-S3-to-S3-1`.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an on-demand trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-ondemand-flights-1
# 
```

```
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating an on-demand trigger for a job
Resources:
# Create trigger to run an existing job (CFNJobName) on an on-demand schedule.
CFNTriggerSample:
  Type: AWS::Glue::Trigger
  Properties:
    Name:
      Ref: CFNTriggerName
    Description: Trigger created with CloudFormation
    Type: ON_DEMAND
    Actions:
      - JobName: !Ref CFNJobName
        # Arguments: JSON object
    #Schedule:
    #Predicate:
```

Sample AWS CloudFormation Template for an AWS Glue Scheduled Trigger

An AWS Glue trigger in the Data Catalog contains the parameter values that are required to start a job run when the trigger fires. A scheduled trigger fires when it is enabled and the cron timer pops.

This sample creates a scheduled trigger that starts one job named `cfn-job-S3-to-S3-1`. The timer is a cron expression to run the job every 10 minutes on weekdays.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a scheduled trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-scheduled-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating a scheduled trigger for a job
#
Resources:
# Create trigger to run an existing job (CFNJobName) on a cron schedule.
TriggerSample1CFN:
  Type: AWS::Glue::Trigger
  Properties:
    Name:
      Ref: CFNTriggerName
    Description: Trigger created with CloudFormation
    Type: SCHEDULED
    Actions:
      - JobName: !Ref CFNJobName
        # Arguments: JSON object
    # # Run the trigger every 10 minutes on Monday to Friday
    Schedule: cron(0/10 * ? * MON-FRI *)
    #Predicate:
```

Sample AWS CloudFormation Template for an AWS Glue Conditional Trigger

An AWS Glue trigger in the Data Catalog contains the parameter values that are required to start a job run when the trigger fires. A conditional trigger fires when it is enabled and its conditions are met, such as a job completing successfully.

This sample creates a conditional trigger that starts one job named `cfn-job-S3-to-S3-1`. This job starts when the job named `cfn-job-S3-to-S3-2` completes successfully.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a conditional trigger for a job, which starts
when another job completes
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The existing job that when it finishes causes trigger to fire
  CFNJobName2:
    Type: String
    Default: cfn-job-S3-to-S3-2
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-conditional-1
#
Resources:
  # Create trigger to run an existing job (CFNJobName) when another job completes
  # (CFNJobName2).
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: CONDITIONAL
      Actions:
        - JobName: !Ref CFNJobName
          # Arguments: JSON object
      #Schedule: none
      Predicate:
        #Value for Logical is required if more than 1 job listed in Conditions
        Logical: AND
        Conditions:
          - LogicalOperator: EQUALS
            JobName: !Ref CFNJobName2
            State: SUCCEEDED
```

Sample AWS CloudFormation Template for an AWS Glue Development Endpoint

An AWS Glue development endpoint is an environment that you can use to develop and test your AWS Glue scripts.

This sample creates a development endpoint with the minimal network parameter values required to successfully create it. For more information about the parameters that you need to set up a development endpoint, see [Setting Up Your Environment for Development Endpoints \(p. 34\)](#).

You provide an existing IAM role ARN (Amazon Resource Name) to create the development endpoint. Supply a valid RSA public key and keep the corresponding private key available if you plan to create a notebook server on the development endpoint.

Note

For any notebook server that you create that is associated with a development endpoint, you manage it. Therefore, if you delete the development endpoint, to delete the notebook server, you must delete the AWS CloudFormation stack on the AWS CloudFormation console.

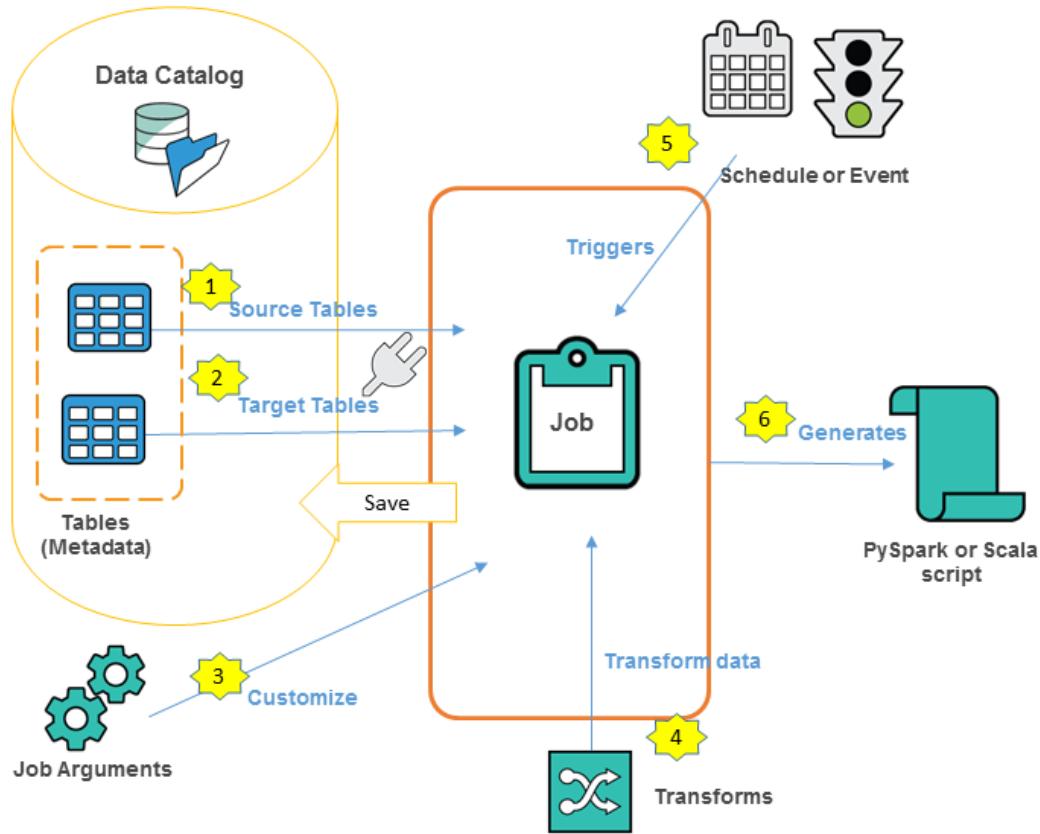
```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a development endpoint
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
CFNEndpointName:
  Type: String
  Default: cfn-devendpoint-1
CFNIAMRoleArn:
  Type: String
  Default: arn:aws:iam::123456789012:role/AWSGlueServiceRoleGA
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNDevEndpoint:
    Type: AWS::Glue::DevEndpoint
    Properties:
      EndpointName: !Ref CFNEndpointName
      #ExtraJarsS3Path: String
      #ExtraPythonLibsS3Path: String
      NumberOfNodes: 5
      PublicKey: ssh-rsa public.....key myuserid-key
      RoleArn: !Ref CFNIAMRoleArn
      SecurityGroupIds:
        - sg-64986c0b
      SubnetId: subnet-c67ccac
```

Authoring Jobs in AWS Glue

A *job* is the business logic that performs the extract, transform, and load (ETL) work in AWS Glue. When you start a job, AWS Glue runs a script that extracts data from sources, transforms the data, and loads it into targets. You can create jobs in the ETL section of the AWS Glue console. For more information, see [Working with Jobs on the AWS Glue Console \(p. 174\)](#).

The following diagram summarizes the basic workflow and steps involved in authoring a job in AWS Glue:



Topics

- [Workflow Overview \(p. 164\)](#)
- [Adding Jobs in AWS Glue \(p. 164\)](#)
- [Editing Scripts in AWS Glue \(p. 179\)](#)
- [Triggering Jobs in AWS Glue \(p. 182\)](#)
- [Developing Scripts Using Development Endpoints \(p. 184\)](#)
- [Managing Notebooks \(p. 209\)](#)

Workflow Overview

When you author a job, you supply details about data sources, targets, and other information. The result is a generated Apache Spark API (PySpark) script. You can then store your job definition in the AWS Glue Data Catalog.

The following describes an overall process of authoring jobs in the AWS Glue console:

1. You choose a data source for your job. The tables that represent your data source must already be defined in your Data Catalog. If the source requires a connection, the connection is also referenced in your job. If your job requires multiple data sources, you can add them later by editing the script.
2. You choose a data target of your job. The tables that represent the data target can be defined in your Data Catalog, or your job can create the target tables when it runs. You choose a target location when you author the job. If the target requires a connection, the connection is also referenced in your job. If your job requires multiple data targets, you can add them later by editing the script.
3. You customize the job-processing environment by providing arguments for your job and generated script. For more information, see [Adding Jobs in AWS Glue \(p. 164\)](#).
4. Initially, AWS Glue generates a script, but you can also edit this script to add sources, targets, and transforms. For more information about transforms, see [Built-In Transforms \(p. 172\)](#).
5. You specify how your job is invoked, either on demand, by a time-based schedule, or by an event. For more information, see [Triggering Jobs in AWS Glue \(p. 182\)](#).
6. Based on your input, AWS Glue generates a PySpark or Scala script. You can tailor the script based on your business needs. For more information, see [Editing Scripts in AWS Glue \(p. 179\)](#).

Adding Jobs in AWS Glue

A job consists of the business logic that performs work in AWS Glue. Typically, a job runs extract, transform, and load (ETL) scripts. You can monitor job runs to understand runtime metrics such as success, duration, and start time. The output of a job is your transformed data, written to a location that you specify.

Job runs can be initiated by triggers that start a job when they fire. A job contains a script that connects to your source data, processes your data using the script's logic, and then writes it out to your data target. Your job can have multiple data sources and multiple data targets. You can use scripts that are generated by AWS Glue to transform data, or you can provide your own. The AWS Glue code generator can automatically create an Apache Spark API (PySpark) script given a source schema and target location or schema. You can use this script as a starting point and edit it to meet your goals.

AWS Glue can write output files in several data formats, including JSON, CSV, ORC (Optimized Row Columnar), Apache Parquet, and Apache Avro. For some data formats, common compression formats can be written.

There are two types of jobs in AWS Glue: *Spark* and *Python shell*.

- An Apache Spark ETL job consists of the business logic that performs ETL work in AWS Glue. You can monitor job runs to understand runtime metrics such as success, duration, and start time. The output of a job is your transformed data, written to a location that you specify.
- A Python shell job runs Python scripts as a shell. With a Python shell job, you can run scripts that are compatible with Python 2.7 or Python 3.6. You can use these jobs to schedule and run tasks that don't require Spark ETL jobs.

Defining Job Properties

When you define your job on the [AWS Glue console \(p. 174\)](#), you provide values for properties to control the AWS Glue runtime environment. The following list describes some of the properties of a Spark job. For the properties of a Python shell job, see [Defining Job Properties for Python Shell Jobs \(p. 168\)](#).

Name

Provide a UTF-8 string with a maximum length of 255 characters.

IAM role

Specify the IAM role that is used for authorization to resources used to run the job and access data stores. For more information about permissions for running jobs in AWS Glue, see [Managing Access Permissions for AWS Glue Resources \(p. 52\)](#).

Type

Specify the type of job environment to run:

- Choose **Spark** to run an Apache Spark ETL script with the job command named `glueetl`.
- Choose **Python shell** to run a Python script with the job command named `pythonshell`. For more information, see [Adding Python Shell Jobs in AWS Glue \(p. 168\)](#).

Glue version

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark. The following table lists the available AWS Glue versions and corresponding Spark and Python versions.

Glue version	Supported Spark and Python versions
0.9	<ul style="list-style-type: none">• Spark 2.2.1• Python 2.7
1.0	<ul style="list-style-type: none">• Spark 2.4.3• Python 2.7• Python 3.6

Jobs that were created without specifying a Glue version default to Glue 0.9.

Generated or custom script

The code in the ETL script defines your job's procedural logic. The script can be coded in Python or Scala. You can choose whether the script that the job runs is generated by AWS Glue or provided by you. You provide the script name and location in Amazon Simple Storage Service (Amazon S3). Confirm that there isn't a file with the same name as the script directory in the path. To learn more about using scripts, see [Editing Scripts in AWS Glue \(p. 179\)](#).

Scala class name

If the script is coded in Scala, you must provide a class name. The default class name for AWS Glue generated scripts is **GlueApp**.

Temporary directory

Provide the location of a working directory in Amazon S3 where temporary intermediate results are written when AWS Glue runs the script. Confirm that there isn't a file with the same name as the temporary directory in the path. This directory is used when AWS Glue reads and writes to Amazon Redshift and by certain AWS Glue transforms.

Job bookmark

Specify how AWS Glue processes state information when the job runs. You can have it remember previously processed data, update state information, or ignore state information.

Job metrics

Enable or disable the creation of Amazon CloudWatch metrics when this job runs. To see profiling data, you must enable this option. For more information about how to enable and visualize metrics, see [Job Monitoring and Debugging \(p. 259\)](#).

Tags

Tag your job with a **Tag key** and an optional **Tag value**. After tag keys are created, they are read-only. Use tags on some resources to help you organize and identify them. For more information, see [AWS Tags in AWS Glue \(p. 229\)](#).

Server-side encryption

If you select this option, when the ETL job writes to Amazon S3, the data is encrypted at rest using SSE-S3 encryption. Both your Amazon S3 data target and any data that is written to an Amazon S3 temporary directory is encrypted. For more information, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#).

Important

Currently, a security configuration overrides any server-side encryption (SSE-S3) setting passed as an ETL job parameter. Thus, if both a security configuration and an SSE-S3 parameter are associated with a job, the SSE-S3 parameter is ignored.

Script libraries

If your script requires it, you can specify locations for the following:

- Python library path
- Dependent jars path
- Referenced files path

You can define the comma-separated Amazon S3 paths for these libraries when you define a job. You can override these paths when you run the job. For more information, see [Providing Your Own Custom Scripts \(p. 181\)](#).

Worker type

The following worker types are available:

- **Standard** – When you choose this type, you also provide a value for **Maximum capacity**. Maximum capacity is the number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. The **Standard** worker type has a 50 GB disk and 2 executors.
- **G.1X** – When you choose this type, you also provide a value for **Number of workers**. Each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- **G.2X** – When you choose this type, you also provide a value for **Number of workers**. Each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs and jobs that run ML transforms.

You are charged an hourly rate based on the number of DPUs used to run your ETL jobs. For more information, see the [AWS Glue pricing page](#).

When you configure a job using the console and specify a **Worker type** of **Standard**, the **Maximum capacity** is set and the **Number of workers** becomes the value of **Maximum capacity - 1**. If you

use the AWS Command Line Interface (AWS CLI) or AWS SDK, you can specify the **Max capacity** parameter, or you can specify both **Worker type** and the **Number of workers**. For more information, see [Jobs](#).

Number of workers

The number of workers of a defined `workerType` that are allocated when a job runs.

With **G.1X** and **G.2X Worker types**, you must specify the number of workers of that type.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

Maximum capacity

The maximum number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. You are charged an hourly rate based on the number of DPUs used to run your ETL jobs. For more information, see the [AWS Glue pricing page](#).

With the **Standard Worker type**, you must specify the maximum capacity of the job.

Choose an integer from 2 to 100. The default is 10. This job type cannot have a fractional DPU allocation.

Max concurrency

Sets the maximum number of concurrent runs that are allowed for this job. The default is 1. An error is returned when this threshold is reached. The maximum value you can specify is controlled by a service limit. For example, if a previous run of a job is still running when a new instance is started, you might want to return an error to prevent two instances of the same job from running concurrently.

Job timeout

Sets the maximum execution time in minutes. The default is 2880 minutes. If this limit is greater than the execution time, the job run state changes to "TIMEOUT".

Delay notification threshold

Sets the threshold (in minutes) before a delay notification is sent. You can set this threshold to send notifications when a RUNNING, STARTING, or STOPPING job run takes more than an expected number of minutes.

Number of retries

Specify the number of times, from 0 to 10, that AWS Glue should automatically restart the job if it fails.

Job parameters

A set of key-value pairs that are passed as named parameters to the script invoked by the job. These are default values that are used when the script is run, but you can override them at run time. The key name is prefixed with `--`, for example `--myKey` and the value is `value-for-myKey`.

```
'--myKey' : 'value-for-myKey'
```

For more examples, see Python parameters in [Passing and Accessing Python Parameters in AWS Glue \(p. 315\)](#).

Source

Specify a catalog table.

Target

Do one of the following:

- To specify an Amazon S3 path or JDBC data store, choose **Create tables in your data target**.
- To specify a catalog table, choose **Use tables in the data catalog and update your data target**.

For Amazon S3 target locations, provide the location of a directory where your output is written. Confirm that there isn't a file with the same name as the target path directory in the path. For JDBC targets, AWS Glue creates schema objects as needed if the specified objects do not exist.

Note

Source and target are not listed under the console **Details** tab for a job. Review the script to see source and target details.

Use Glue Data Catalog as the Hive metastore

Enables you to use the AWS Glue Data Catalog as a Spark Hive metastore.

Spark UI

Enable the use of Spark UI for monitoring this job. For more information, see [Enabling the Apache Spark Web UI for AWS Glue Jobs \(p. 237\)](#).

For more information about adding a job using the AWS Glue console, see [Working with Jobs on the AWS Glue Console \(p. 174\)](#).

Adding Python Shell Jobs in AWS Glue

You can use a Python shell job to run Python scripts as a shell in AWS Glue. With a Python shell job, you can run scripts that are compatible with Python 2.7 or Python 3.6.

You can't use job bookmarks with Python shell jobs. Most of the other features that are available for Apache Spark jobs are also available for Python shell jobs.

The Amazon CloudWatch Logs group for Python shell jobs output is `/aws-glue/python-jobs/` output. For errors, see the log group `/aws-glue/python-jobs/errors`.

Topics

- [Defining Job Properties for Python Shell Jobs \(p. 168\)](#)
- [Supported Libraries for Python Shell Jobs \(p. 169\)](#)
- [Limitations \(p. 170\)](#)
- [Providing Your Own Python Library \(p. 170\)](#)

Defining Job Properties for Python Shell Jobs

When you define your Python shell job on the console (see [the section called "Jobs on the Console" \(p. 174\)](#)), you provide some of the following properties:

IAM role

Specify the AWS Identity and Access Management (IAM) role that is used for authorization to resources that are used to run the job and access data stores. For more information about permissions for running jobs in AWS Glue, see [Managing Access Permissions for AWS Glue Resources \(p. 52\)](#).

Type

Choose **Python shell** to run a Python script with the job command named `pythonshell`.

Python version

Choose the Python version. The default is Python 3.

Custom script

The code in the script defines your job's procedural logic. You provide the script name and location in Amazon Simple Storage Service (Amazon S3). Confirm that there isn't a file with the same name as the script directory in the path. To learn more about using scripts, see [Editing Scripts in AWS Glue \(p. 179\)](#).

An existing or new script

The code in the script defines your job's procedural logic. You can code the script in Python 2.7 or Python 3.6. You can edit a script on the AWS Glue console, but it is not generated by AWS Glue.

Maximum capacity

The maximum number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see [AWS Glue pricing](#).

You can set the value to 0.0625 or 1. The default is 0.0625.

For descriptions of additional properties, see [Defining Job Properties \(p. 165\)](#). For more information about adding a job using the AWS Glue console, see [Working with Jobs on the AWS Glue Console \(p. 174\)](#).

You can also create a **Python shell** job using the AWS CLI, as in the following example.

```
aws glue create-job --name python-job-cli --role Glue_DefaultRole
    --command '{"Name" : "pythonshell", "ScriptLocation" : "s3://aws-glue-
scripts-123456789012-us-east-1/Admin/python-job-cli.py"}'
```

Jobs that you create with the AWS CLI default to Python 2. To specify Python 3, add this tuple to the `--command` parameter:

```
"PythonVersion": "3"
```

To set the maximum capacity used by a Python shell job, provide the `--max-capacity` parameter. For Python shell jobs, the `--allocated-capacity` parameter can't be used.

Supported Libraries for Python Shell Jobs

The environment for running a Python shell job supports the following libraries:

- Boto3
- collections
- CSV
- gzip
- multiprocessing
- NumPy

- pandas (required to be installed via the python setuptools configuration, `setup.py`)
- pickle
- PyGreSQL
- re
- SciPy
- sklearn
- sklearn.feature_extraction
- sklearn.preprocessing
- xml.etree.ElementTree
- zipfile

You can use the `NumPy` library in a Python shell job for scientific computing. For more information, see [NumPy](#). The following example shows a NumPy script that can be used in a Python shell job. The script prints "Hello world" and the results of several mathematical calculations.

```
import numpy as np
print("Hello world")

a = np.array([20,30,40,50])
print(a)

b = np.arange( 4 )

print(b)

c = a-b

print(c)

d = b**2

print(d)
```

Limitations

Note the following limitations on packaging your Python libraries:

- Creating an `.egg` file on Windows 10 Pro using Python 3.7 is not supported.
- Creating an `.egg` file on WSL (Windows Linux Subsystem, hosted by Windows 10 Pro) using Python 3.6 is supported.

Providing Your Own Python Library

You might already have one or more Python libraries packaged as an `.egg` or a `.whl` file. If so, you can specify them to your job using the AWS Command Line Interface (AWS CLI) under the `--extra-py-files` flag, as in the following example.

```
aws glue create-job --name python-redshift-test-cli --role role --command '{"Name" : "pythonshell", "ScriptLocation" : "s3://MyBucket/python/library/redshift_test.py"}' --connections Connections=connection-name --default-arguments '{"--extra-py-files" : ["s3://MyBucket/python/library/redshift_module-0.1-py2.7.egg", "s3://MyBucket/python/library/redshift_module-0.1-py2.7-none-any.whl"]}'
```

If you aren't sure how to create an .egg or a .whl file from a Python library, use the following steps. This example is applicable on macOS, Linux, and Windows Subsystem for Linux (WSL).

To create a Python .egg or .whl file

1. Create an Amazon Redshift cluster in a virtual private cloud (VPC), and add some data to a table.
2. Create an AWS Glue connection for the VPC-SecurityGroup-Subnet combination that you used to create the cluster. Test that the connection is successful.
3. Create a directory named `redshift_example`, and create a file named `setup.py`. Paste the following code into `setup.py`.

```
from setuptools import setup

setup(
    name="redshift_module",
    version="0.1",
    packages=['redshift_module']
)
```

4. In the `redshift_example` directory, create a `redshift_module` directory. In the `redshift_module` directory, create the files `__init__.py` and `pygresql_redshift_common.py`.
5. Leave the `__init__.py` file empty. In `pygresql_redshift_common.py`, paste the following code. Replace `port`, `db_name`, `user`, and `password_for_user` with details specific to your Amazon Redshift cluster. Replace `table_name` with the name of the table in Amazon Redshift.

```
import pg

def get_connection(host):
    rs_conn_string = "host=%s port=%s dbname=%s user=%s password=%s" % (
        host, port, db_name, user, password_for_user)

    rs_conn = pg.connect(dbname=rs_conn_string)
    rs_conn.query("set statement_timeout = 1200000")
    return rs_conn

def query(con):
    statement = "Select * from table_name;"
    res = con.query(statement)
    return res
```

6. If you're not already there, change to the `redshift_module` directory.
7. Do one of the following:

- To create an .egg file, run the following command.

```
python setup.py bdist_egg
```

- To create a .whl file, run the following command.

```
python setup.py bdist_wheel
```

8. Install the dependencies that are required for the preceding command.
9. The command creates a file in the `dist` directory:
 - If you created an egg file, it's named `redshift_module-0.1-py2.7.egg`.

- If you created a wheel file, it's named `redshift_module-0.1-py2.7-none-any.whl`.

Upload this file to Amazon S3.

In this example, the uploaded file path is either `s3://MyBucket/python/library/redshift_module-0.1-py2.7.egg` or `s3://MyBucket/python/library/redshift_module-0.1-py2.7-none-any.whl`.

10. Create a Python file to be used as a script for the AWS Glue job, and add the following code to the file.

```
from redshift_module import pygresql_redshift_common as rs_common

con1 = rs_common.get_connection(redshift_endpoint)
res = rs_common.query(con1)

print "Rows in the table cities are: "

print res
```

11. Upload the preceding file to Amazon S3. In this example, the uploaded file path is `s3://MyBucket/python/library/redshift_test.py`.
12. Create a Python shell job using this script. On the AWS Glue console, on the **Job properties** page, specify the path to the `.egg/.whl` file in the **Python library path** box. If you have multiple `.egg/.whl` files and Python files, provide a comma-separated list in this box.

Using the AWS CLI, create a job with a command, as in the following example.

```
aws glue create-job --name python-redshift-test-cli --role Role --command '{"Name": "pythonshell", "ScriptLocation": "s3://MyBucket/python/library/redshift_test.py"}' --connections Connections="connection-name" --default-arguments '{"--extra-py-files": ["s3://MyBucket/python/library/redshift_module-0.1-py2.7.egg", "s3://MyBucket/python/library/redshift_module-0.1-py2.7-none-any.whl"]}'
```

When the job runs, the script prints the rows created in the `table_name` table in the Amazon Redshift cluster.

Built-In Transforms

AWS Glue provides a set of built-in transforms that you can use to process your data. You can call these transforms from your ETL script. Your data passes from transform to transform in a data structure called a *DynamicFrame*, which is an extension to an Apache Spark SQL *DataFrame*. The *DynamicFrame* contains your data, and you reference its schema to process your data. For more information about these transforms, see [AWS Glue PySpark Transforms Reference \(p. 360\)](#).

AWS Glue provides the following built-in transforms:

ApplyMapping

Maps source columns and data types from a *DynamicFrame* to target columns and data types in a returned *DynamicFrame*. You specify the `mapping` argument, which is a list of tuples that contain source column, source type, target column, and target type.

DropFields

Removes a field from a *DynamicFrame*. The output *DynamicFrame* contains fewer fields than the input. You specify which fields to remove using the `paths` argument. The `paths` argument points

to a field in the schema tree structure using dot notation. For example, to remove field B, which is a child of field A in the tree, type `A.B` for the path.

DropNullFields

Removes null fields from a `DynamicFrame`. The output `DynamicFrame` does not contain fields of the null type in the schema.

Filter

Selects records from a `DynamicFrame` and returns a filtered `DynamicFrame`. You specify a function, such as a Lambda function, which determines whether a record is output (function returns true) or not (function returns false).

Join

Equijoin of two `DynamicFrames`. You specify the key fields in the schema of each frame to compare for equality. The output `DynamicFrame` contains rows where keys match.

Map

Applies a function to the records of a `DynamicFrame` and returns a transformed `DynamicFrame`. The supplied function is applied to each input record and transforms it to an output record. The map transform can add fields, delete fields, and perform lookups using an external API operation. If there is an exception, processing continues, and the record is marked as an error.

MapToCollection

Applies a transform to each `DynamicFrame` in a `DynamicFrameCollection`.

Relationalize

Converts a `DynamicFrame` to a relational (rows and columns) form. Based on the data's schema, this transform flattens nested structures and creates `DynamicFrames` from arrays structures. The output is a collection of `DynamicFrames` that can result in data written to multiple tables.

RenameField

Renames a field in a `DynamicFrame`. The output is a `DynamicFrame` with the specified field renamed. You provide the new name and the path in the schema to the field to be renamed.

ResolveChoice

Use `ResolveChoice` to specify how a column should be handled when it contains values of multiple types. You can choose to either cast the column to a single data type, discard one or more of the types, or retain all types in either separate columns or a structure. You can select a different resolution policy for each column or specify a global policy that is applied to all columns.

SelectFields

Selects fields from a `DynamicFrame` to keep. The output is a `DynamicFrame` with only the selected fields. You provide the paths in the schema to the fields to keep.

SelectFromCollection

Selects one `DynamicFrame` from a collection of `DynamicFrames`. The output is the selected `DynamicFrame`. You provide an index to the `DynamicFrame` to select.

Spigot

Writes sample data from a `DynamicFrame`. Output is a JSON file in Amazon S3. You specify the Amazon S3 location and how to sample the `DynamicFrame`. Sampling can be a specified number of records from the beginning of the file or a probability factor used to pick records to write.

SplitFields

Splits fields into two `DynamicFrames`. Output is a collection of `DynamicFrames`: one with selected fields, and one with the remaining fields. You provide the paths in the schema to the selected fields.

SplitRows

Splits rows in a `DynamicFrame` based on a predicate. The output is a collection of two `DynamicFrames`: one with selected rows, and one with the remaining rows. You provide the comparison based on fields in the schema. For example, `A > 4`.

Unbox

Unboxes a string field from a `DynamicFrame`. The output is a `DynamicFrame` with the selected string field reformatted. The string field can be parsed and replaced with several fields. You provide a path in the schema for the string field to reformat and its current format type. For example, you might have a CSV file that has one field that is in JSON format `{"a": 3, "b": "foo", "c": 1.2}`. This transform can reformat the JSON into three fields: an `int`, a `string`, and a `double`.

Working with Jobs on the AWS Glue Console

A job in AWS Glue consists of the business logic that performs extract, transform, and load (ETL) work. You can create jobs in the **ETL** section of the AWS Glue console.

To view existing jobs, sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Then choose the **Jobs** tab in AWS Glue. The **Jobs** list displays the location of the script that is associated with each job, when the job was last modified, and the current job bookmark option.

From the **Jobs** list, you can do the following:

- To start an existing job, choose **Action**, and then choose **Run job**.
- To stop a **Running** or **Starting** job, choose **Action**, and then choose **Stop job run**.
- To add triggers that start a job, choose **Action**, **Choose job triggers**.
- To modify an existing job, choose **Action**, and then choose **Edit job** or **Delete**.
- To change a script that is associated with a job, choose **Action**, **Edit script**.
- To reset the state information that AWS Glue stores about your job, choose **Action**, **Reset job bookmark**.
- To create a development endpoint with the properties of this job, choose **Action**, **Create development endpoint**.

To add a new job using the console

1. Open the AWS Glue console, and choose the **Jobs** tab.
2. Choose **Add job**, and follow the instructions in the **Add job** wizard.

If you decide to have AWS Glue generate a script for your job, you must specify the job properties, data sources, and data targets, and verify the schema mapping of source columns to target columns. The generated script is a starting point for you to add code to perform your ETL work. Verify the code in the script and modify it to meet your business needs.

Note

To get step-by-step guidance for adding a job with a generated script, see the **Add job** tutorial in the console.

Optionally, you can add a security configuration to a job to specify at-rest encryption options.

If you provide or author the script, your job defines the sources, targets, and transforms. But you **must specify any connections that are required by the script in the job**. For information about creating your own script, see [Providing Your Own Custom Scripts \(p. 181\)](#).

Note

The job assumes the permissions of the **IAM role** that you specify when you create it. This IAM role must have permission to extract data from your data store and write to your target. The AWS Glue console only lists IAM roles that have attached a trust policy for the AWS Glue principal service. For more information about providing roles for AWS Glue, see [Identity-Based Policies \(p. 55\)](#).

If the job reads AWS KMS encrypted Amazon Simple Storage Service (Amazon S3) data, then the **IAM role** must have decrypt permission on the KMS key. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#).

Important

Check [Troubleshooting Errors in AWS Glue \(p. 626\)](#) for known problems when a job runs.

To learn about the properties that are required for each job, see [Defining Job Properties \(p. 165\)](#).

To get step-by-step guidance for adding a job with a generated script, see the **Add job** tutorial in the AWS Glue console.

Viewing Job Details

To see details of a job, select the job in the **Jobs** list and review the information on the following tabs:

- History
- Details
- Script
- Metrics

History

The **History** tab shows your job run history and how successful a job has been in the past. For each job, the run metrics include the following:

- **Run ID** is an identifier created by AWS Glue for each run of this job.
- **Retry attempt** shows the number of attempts for jobs that required AWS Glue to automatically retry.
- **Run status** shows the success of each run listed with the most recent run at the top. If a job is **Running** or **Starting**, you can choose the action icon in this column to stop it.
- **Error** shows the details of an error message if the run was not successful.
- **Logs** links to the logs written to `stdout` for this job run.

The **Logs** link takes you to Amazon CloudWatch Logs, where you can see all the details about the tables that were created in the AWS Glue Data Catalog and any errors that were encountered. You can manage your log retention period in the CloudWatch console. The default log retention is **Never Expire**. For more information about how to change the retention period, see [Change Log Data Retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

- **Error logs** links to the logs written to `stderr` for this job run.

This link takes you to CloudWatch Logs, where you can see details about any errors that were encountered. You can manage your log retention period on the CloudWatch console. The default log retention is **Never Expire**. For more information about how to change the retention period, see [Change Log Data Retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

- **Execution time** shows the length of time during which the job run consumed resources. The amount is calculated from when the job run starts consuming resources until it finishes.
- **Timeout** shows the maximum execution time during which this job run can consume resources before it stops and goes into timeout status.

- **Delay** shows the threshold before sending a job delay notification. When a job run execution time reaches this threshold, AWS Glue sends a notification ("Glue Job Run Status") to CloudWatch Events.
- **Triggered by** shows the trigger that fired to start this job run.
- **Start time** shows the date and time (local time) that the job started.
- **End time** shows the date and time (local time) that the job ended.

For a specific job run, you can [View run metrics](#), which displays graphs of metrics for the selected job run. For more information about how to enable metrics and interpret the graphs, see [Job Monitoring and Debugging \(p. 259\)](#).

Details

The **Details** tab includes attributes of your job. It shows you the details about the job definition and also lists the triggers that can start this job. Each time one of the triggers in the list fires, the job is started. For the list of triggers, the details include the following:

- **Trigger name** shows the names of triggers that start this job when fired.
- **Trigger type** lists the type of trigger that starts this job.
- **Trigger status** displays whether the trigger is created, activated, or deactivated.
- **Trigger parameters** shows parameters that define when the trigger fires.
- **Jobs to trigger** shows the list of jobs that start when this trigger fires.

Note

The **Details** tab does not include source and target information. Review the script to see the source and target details.

Script

The **Script** tab shows the script that runs when your job is started. You can invoke an [Edit script](#) view from this tab. For more information about the script editor in the AWS Glue console, see [Working with Scripts on the AWS Glue Console \(p. 181\)](#). For information about the functions that are called in your script, see [Program AWS Glue ETL Scripts in Python \(p. 314\)](#).

Metrics

The **Metrics** tab shows metrics collected when a job runs and profiling is enabled. The following graphs are shown:

- ETL Data Movement
- Memory Profile: Driver and Executors

Choose [View additional metrics](#) to show the following graphs:

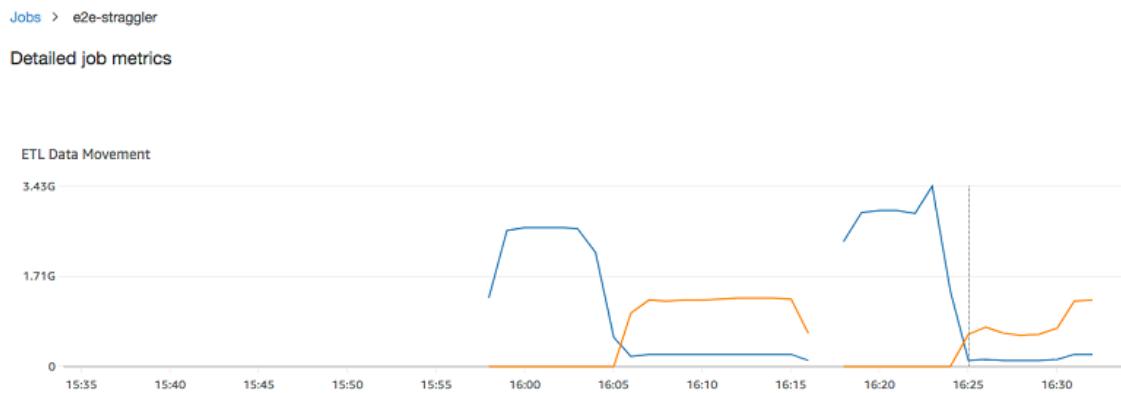
- ETL Data Movement
- Memory Profile: Driver and Executors
- Data Shuffle Across Executors
- CPU Load: Driver and Executors
- Job Execution: Active Executors, Completed Stages & Maximum Needed Executors

Data for these graphs is pushed to CloudWatch metrics if the job is enabled to collect metrics. For more information about how to enable metrics and interpret the graphs, see [Job Monitoring and Debugging \(p. 259\)](#).

Example of ETL Data Movement Graph

The ETL Data Movement graph shows the following metrics:

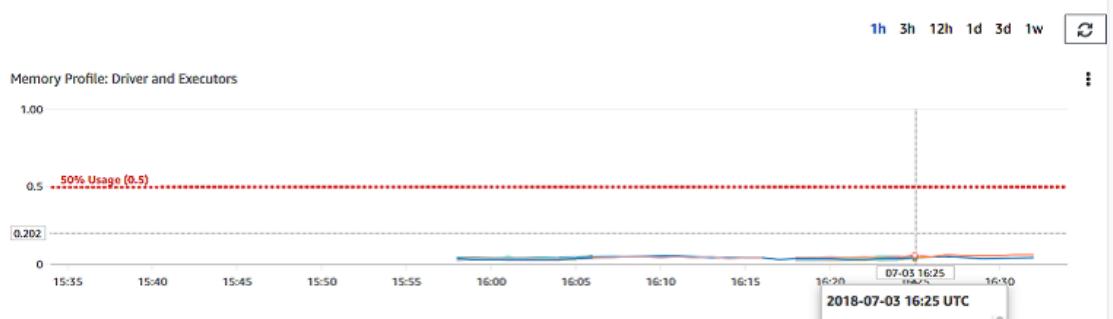
- The number of bytes read from Amazon S3 by all executors
—[glue.ALL.s3.filesystem.read_bytes \(p. 253\)](#)
- The number of bytes written to Amazon S3 by all executors
—[glue.ALL.s3.filesystem.write_bytes \(p. 254\)](#)



Example of Memory Profile Graph

The Memory Profile graph shows the following metrics:

- The fraction of memory used by the JVM heap for this driver (scale: 0–1) by the driver, an executor identified by *executorId*, or all executors—
 - [glue.driver.jvm.heap.usage \(p. 251\)](#)
 - [glue.executorId.jvm.heap.usage \(p. 251\)](#)
 - [glue.ALL.jvm.heap.usage \(p. 251\)](#)



Example of Data Shuffle Across Executors Graph

The Data Shuffle Across Executors graph shows the following metrics:

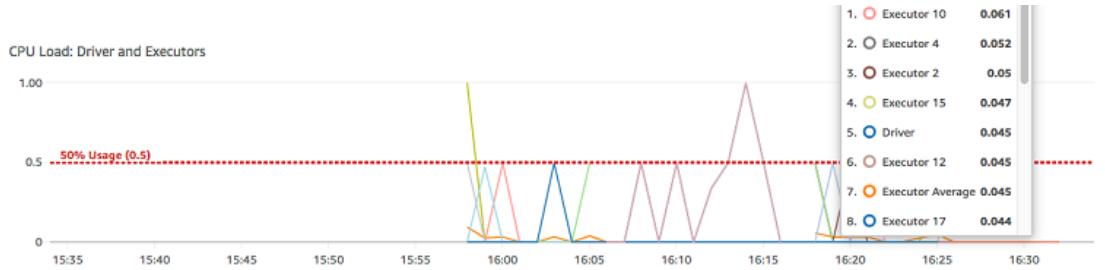
- The number of bytes read by all executors to shuffle data between them
—[glue.driver.aggregate.shuffleLocalBytesRead \(p. 248\)](#)
- The number of bytes written by all executors to shuffle data between them
—[glue.driver.aggregate.shuffleBytesWritten \(p. 247\)](#)



Example of CPU Load Graph

The CPU Load graph shows the following metrics:

- The fraction of CPU system load used (scale: 0–1) by the driver, an executor identified by *executorId*, or all executors—
 - [glue.driver.system.cpuSystemLoad \(p. 255\)](#)
 - [glue.executorId.system.cpuSystemLoad \(p. 255\)](#)
 - [glue.ALL.system.cpuSystemLoad \(p. 255\)](#)



Example of Job Execution Graph

The Job Execution graph shows the following metrics:

- The number of actively running executors
—[glue.driver.ExecutorAllocationManager.executors.numberAllExecutors \(p. 249\)](#)
- The number of completed stages—[glue.aggregate.numCompletedStages \(p. 245\)](#)
- The number of maximum needed executors
—[glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors \(p. 250\)](#)



Editing Scripts in AWS Glue

A script contains the code that extracts data from sources, transforms it, and loads it into targets. AWS Glue runs a script when it starts a job.

AWS Glue ETL scripts can be coded in Python or Scala. Python scripts use a language that is an extension of the PySpark Python dialect for extract, transform, and load (ETL) jobs. The script contains *extended constructs* to deal with ETL transformations. When you automatically generate the source code logic for your job, a script is created. You can edit this script, or you can provide your own script to process your ETL work.

For information about defining and editing scripts using the AWS Glue console, see [Working with Scripts on the AWS Glue Console \(p. 181\)](#).

Defining a Script

Given a source and target, AWS Glue can generate a script to transform the data. This proposed script is an initial version that fills in your sources and targets, and suggests transformations in PySpark. You can verify and modify the script to fit your business needs. Use the script editor in AWS Glue to add arguments that specify the source and target, and any other arguments that are required to run. Scripts are run by jobs, and jobs are started by triggers, which can be based on a schedule or an event. For more information about triggers, see [Triggering Jobs in AWS Glue \(p. 182\)](#).

In the AWS Glue console, the script is represented as code. You can also view the script as a diagram that uses annotations (##) embedded in the script. These annotations describe the parameters, transform types, arguments, inputs, and other characteristics of the script that are used to generate a diagram in the AWS Glue console.

The diagram of the script shows the following:

- Source inputs to the script
- Transforms
- Target outputs written by the script

Scripts can contain the following annotations:

Annotation	Usage
@params	Parameters from the ETL job that the script requires.
@type	Type of node in the diagram, such as the transform type, data source, or data sink.
@args	Arguments passed to the node, except reference to input data.
@return	Variable returned from script.
@inputs	Data input to node.

To learn about the code constructs within a script, see [Program AWS Glue ETL Scripts in Python \(p. 314\)](#).

Example

The following is an example of a script generated by AWS Glue. The script is for a job that copies a simple dataset from one Amazon Simple Storage Service (Amazon S3) location to another, changing the format from CSV to JSON. After some initialization code, the script includes commands that specify the data source, the mappings, and the target (data sink). Note also the annotations.

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "sample-data", table_name = "taxi_trips", transformation_ctx =
"datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "sample-data",
    table_name = "taxi_trips", transformation_ctx = "datasource0")
## @type: ApplyMapping
## @args: [mapping = [("vendorid", "long", "vendorid", "long"), ("tpep_pickup_datetime",
"string", "tpep_pickup_datetime", "string"), ("tpep_dropoff_datetime", "string",
"tpep_dropoff_datetime", "string"), ("passenger_count", "long", "passenger_count",
"long"), ("trip_distance", "double", "trip_distance", "double"), ("ratecodeid",
"long", "ratecodeid", "long"), ("store_and_fwd_flag", "string", "store_and_fwd_flag",
"string"), ("pulocationid", "long", "pulocationid", "long"), ("dolocationid", "long",
"dolocationid", "long"), ("payment_type", "long", "payment_type", "long"), ("fare_amount",
"double", "fare_amount", "double"), ("extra", "double", "extra", "double"), ("mta_tax",
"double", "mta_tax", "double"), ("tip_amount", "double", "tip_amount", "double"),
("tolls_amount", "double", "tolls_amount", "double"), ("improvement_surcharge", "double",
"improvement_surcharge", "double"), ("total_amount", "double", "total_amount", "double")],
transformation_ctx = "applymapping1"]
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("vendorid",
"long", "vendorid", "long"), ("tpep_pickup_datetime", "string", "tpep_pickup_datetime",
"string"), ("tpep_dropoff_datetime", "string", "tpep_dropoff_datetime", "string"),
("passenger_count", "long", "passenger_count", "long"), ("trip_distance",
"double", "trip_distance", "double"), ("ratecodeid", "long", "ratecodeid", "long"),
("store_and_fwd_flag", "string", "store_and_fwd_flag", "string"), ("pulocationid",
"long", "pulocationid", "long"), ("dolocationid", "long", "dolocationid", "long"),
("payment_type", "long", "payment_type", "long"), ("fare_amount", "double", "fare_amount",
"double"), ("extra", "double", "extra", "double"), ("mta_tax", "double", "mta_tax",
"double"), ("tip_amount", "double", "tip_amount", "double"), ("tolls_amount",
"double", "tolls_amount", "double"), ("improvement_surcharge", "double",
"improvement_surcharge", "double"), ("total_amount", "double", "total_amount", "double")], transformation_ctx =
"applymapping1")
## @type: DataSink
## @args: [connection_type = "s3", connection_options = {"path": "s3://example-data-
destination/taxi-data"}, format = "json", transformation_ctx = "datasink2"]
## @return: datasink2
## @inputs: [frame = applymapping1]
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
    connection_type = "s3", connection_options = {"path": "s3://example-data-destination/taxi-
data"}, format = "json", transformation_ctx = "datasink2")

```

```
job.commit()
```

Working with Scripts on the AWS Glue Console

A script contains the code that performs extract, transform, and load (ETL) work. You can provide your own script, or AWS Glue can generate a script with guidance from you. For information about creating your own scripts, see [Providing Your Own Custom Scripts \(p. 181\)](#).

You can edit a script in the AWS Glue console. When you edit a script, you can add sources, targets, and transforms.

To edit a script

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Then choose the **Jobs** tab.
2. Choose a job in the list, and then choose **Action**, **Edit script** to open the script editor.

You can also access the script editor from the job details page. Choose the **Script** tab, and then choose **Edit script**.

Script Editor

The AWS Glue script editor lets you insert, modify, and delete sources, targets, and transforms in your script. The script editor displays both the script and a diagram to help you visualize the flow of data.

To create a diagram for the script, choose **Generate diagram**. AWS Glue uses annotation lines in the script beginning with `##` to render the diagram. To correctly represent your script in the diagram, you must keep the parameters in the annotations and the parameters in the Apache Spark code in sync.

The script editor lets you add code templates wherever your cursor is positioned in the script. At the top of the editor, choose from the following options:

- To add a source table to the script, choose **Source**.
- To add a target table to the script, choose **Target**.
- To add a target location to the script, choose **Target location**.
- To add a transform to the script, choose **Transform**. For information about the functions that are called in your script, see [Program AWS Glue ETL Scripts in Python \(p. 314\)](#).
- To add a Spigot transform to the script, choose **Spigot**.

In the inserted code, modify the parameters in both the annotations and Apache Spark code. For example, if you add a **Spigot** transform, verify that the path is replaced in both the `@args` annotation line and the output code line.

The **Logs** tab shows the logs that are associated with your job as it runs. The most recent 1,000 lines are displayed.

The **Schema** tab shows the schema of the selected sources and targets, when available in the Data Catalog.

Providing Your Own Custom Scripts

Scripts perform the extract, transform, and load (ETL) work in AWS Glue. A script is created when you automatically generate the source code logic for a job. You can either edit this generated script, or you can provide your own custom script.

Important

Different versions of AWS Glue support different versions of Apache Spark. Your custom script must be compatible with the supported Apache Spark version. For information about AWS Glue versions, see the [Glue version job property \(p. 165\)](#).

To provide your own custom script in AWS Glue, follow these general steps:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose the **Jobs** tab, and then choose **Add job** to start the **Add job** wizard.
3. In the **Job properties** screen, choose the **IAM role** that is required for your custom script to run. For more information, see [Identity and Access Management in AWS Glue \(p. 50\)](#).
4. Under **This job runs**, choose one of the following:
 - An existing script that you provide
 - A new script to be authored by you
5. Choose any connections that your script references. These objects are needed to connect to the necessary JDBC data stores.

An elastic network interface is a virtual network interface that you can attach to an instance in a virtual private cloud (VPC). Choose the elastic network interface that is required to connect to the data store that's used in the script.

6. If your script requires additional libraries or files, you can specify them as follows:

Python library path

Comma-separated Amazon Simple Storage Service (Amazon S3) paths to Python libraries that are required by the script.

Note

Only pure Python libraries can be used. Libraries that rely on C extensions, such as the pandas Python Data Analysis Library, are not yet supported.

Dependent jars path

Comma-separated Amazon S3 paths to JAR files that are required by the script.

Note

Currently, only pure Java or Scala (2.11) libraries can be used.

Referenced files path

Comma-separated Amazon S3 paths to additional files (for example, configuration files) that are required by the script.

7. If you want, you can add a schedule to your job. To change a schedule, you must delete the existing schedule and add a new one.

For more information about adding jobs in AWS Glue, see [Adding Jobs in AWS Glue \(p. 164\)](#).

For step-by-step guidance, see the **Add job** tutorial in the AWS Glue console.

Triggering Jobs in AWS Glue

You decide what triggers an extract, transform, and load (ETL) job to run in AWS Glue. The triggering condition can be based on a schedule (as defined by a cron expression) or on an event. You can also run a job on demand.

Triggering Jobs Based on Schedules or Events

When you create a trigger for a job based on a schedule, you can specify constraints, such as the frequency the job runs, which days of the week it runs, and at what time. These constraints are based on *cron*. When you're setting up a schedule for a trigger, you should consider the features and limitations of *cron*. For example, if you choose to run your crawler on day 31 each month, keep in mind that some months don't have 31 days. For more information about *cron*, see [Time-Based Schedules for Jobs and Crawlers \(p. 221\)](#).

When you create a trigger based on an event, you specify events to watch that cause the trigger to fire, such as when another job succeeded. For a conditional trigger based on a *job events* trigger, you specify a list of jobs that cause a trigger to fire when any or all jobs satisfy the watched job events. In turn, when the trigger fires, it starts a run of any dependent jobs.

Defining Trigger Types

A trigger can be one of the following types:

Schedule

A time-based trigger based on *cron*.

Job events (conditional)

An event-based trigger that fires when a previous job or multiple jobs satisfy a list of conditions. You provide a list of job events to watch for when their run state changes to succeeded, failed, stopped, or timeout. This trigger waits to fire until any or all the conditions are satisfied.

Important

Dependent jobs are only started if the job which completes was started by a trigger (not run ad-hoc). All jobs in a dependency chain must be descendants of a single **schedule** or **on-demand** trigger.

On-demand

The trigger fires when you start it. As jobs complete, any triggers watching for completion are also fired and dependent jobs are started.

So that they are ready to fire as soon as they exist, you can set a flag to enable (activate) **schedule** and **job events (conditional)** triggers when they are created.

For more information about defining triggers using the AWS Glue console, see [Working with Triggers on the AWS Glue Console \(p. 183\)](#).

Working with Triggers on the AWS Glue Console

A trigger controls when an ETL job runs in AWS Glue. To view your existing triggers, sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Then choose the **Triggers** tab.

The **Triggers** list displays properties for each trigger:

Trigger name

The unique name you gave the trigger when you created it.

Tags

Tag your trigger with a **Tag key** and optional **Tag value**. Once created, tag keys are read-only. Use tags on some resources to help you organize and identify them. For more information, see [AWS Tags in AWS Glue \(p. 229\)](#).

Trigger type

Indicates whether the trigger is time-based (**Schedule**), event-based (**Job events**), or started by you (**On-demand**).

Trigger status

Indicates whether the trigger is **Enabled** or **ACTIVATED** and ready to invoke associated jobs when it fires. The trigger can also be **Disabled** or **DEACTIVATED** and paused so that it doesn't determine whether a job is invoked. A **CREATED** trigger exists, but does not factor into whether a job runs.

Trigger parameters

For **Schedule** triggers, this includes the details about the frequency and time to fire the trigger. For **Job events** triggers, it includes the list of jobs to watch that, depending on their run state, might fire the trigger. See the details of the trigger for the watch list of jobs with events.

Jobs to trigger

Lists the jobs associated with the trigger that are invoked when this trigger fires.

Adding and Editing Triggers

To edit, delete, or start a trigger, select the check box next to the trigger in the list, and then choose **Action**. Here you can also disable a trigger to prevent it from starting any associated jobs, or enable it to start associated jobs when it fires.

Choose a trigger in the list to view details for the trigger. Trigger details include the information you defined when you created the trigger.

To add a new trigger, choose **Add trigger**, and follow the instructions in the **Add trigger** wizard.

You provide the following properties:

Name

Give your trigger a unique name.

Trigger type

Specify one of the following:

- **Schedule:** The trigger fires at a specific time.
- **Job events:** The trigger fires when any or all jobs in the list match the selected job event. For the trigger to fire, the watched job must have been started by a trigger. For any job you choose, you can only watch one job event.
- **On-demand:** The trigger fires when it is started from the triggers list page.

For **Schedule** and **Job events** trigger types, you can enable them when they are created.

Jobs to trigger

List of jobs that are started by this trigger.

For more information, see [Triggering Jobs in AWS Glue \(p. 182\)](#).

Developing Scripts Using Development Endpoints

AWS Glue can create an environment—known as a *development endpoint*—that you can use to iteratively develop and test your extract, transform, and load (ETL) scripts. You can create, edit, and delete development endpoints using the AWS Glue console or API.

When you create a development endpoint, you provide configuration values to provision the development environment. These values tell AWS Glue how to set up the network so that you can access the endpoint securely and the endpoint can access your data stores.

You can then create a notebook that connects to the endpoint, and use your notebook to author and test your ETL script. When you're satisfied with the results of your development process, you can create an ETL job that runs your script. With this process, you can add functions and debug your scripts in an interactive manner.

Follow the tutorials in this section to learn how to use your development endpoint with notebooks.

Topics

- [Development Endpoint Workflow \(p. 185\)](#)
- [Adding a Development Endpoint \(p. 186\)](#)
- [Viewing Development Endpoint Properties \(p. 187\)](#)
- [Accessing Your Development Endpoint \(p. 189\)](#)
- [Creating a Notebook Server Hosted on Amazon EC2 \(p. 190\)](#)
- [Tutorial Setup: Prerequisites for the Development Endpoint Tutorials \(p. 192\)](#)
- [Tutorial: Set Up a Local Apache Zeppelin Notebook to Test and Debug ETL Scripts \(p. 195\)](#)
- [Tutorial: Set Up an Apache Zeppelin Notebook Server on Amazon EC2 \(p. 198\)](#)
- [Tutorial: Use an Amazon SageMaker Notebook with Your Development Endpoint \(p. 200\)](#)
- [Tutorial: Use a REPL Shell with Your Development Endpoint \(p. 202\)](#)
- [Tutorial: Set Up PyCharm Professional with a Development Endpoint \(p. 203\)](#)

Development Endpoint Workflow

To use an AWS Glue development endpoint, you can follow this workflow:

1. Create a development endpoint using the console or API. The endpoint is launched in a virtual private cloud (VPC) with your defined security groups.
2. The console or API polls the development endpoint until it is provisioned and ready for work. When it's ready, connect to the development endpoint using one of the following methods to create and test AWS Glue scripts.
 - Install an Apache Zeppelin notebook on your local machine, connect it to a development endpoint, and then develop on it using your browser.
 - Create a Zeppelin notebook server in its own Amazon EC2 instance in your account using the AWS Glue console, and then connect to it using your browser. For more information about how to create a notebook server, see [Creating a Notebook Server Associated with a Development Endpoint \(p. 210\)](#).
 - Create an Amazon SageMaker notebook in your account using the AWS Glue console. For more information about how to create a notebook, see [Working with Notebooks on the AWS Glue Console \(p. 217\)](#).
 - Open a terminal window to connect directly to a development endpoint.
 - If you have the professional edition of the JetBrains [PyCharm Python IDE](#), connect it to a development endpoint and use it to develop interactively. If you insert pydevd statements in your script, PyCharm can support remote breakpoints.
3. When you finish debugging and testing on your development endpoint, you can delete it.

Adding a Development Endpoint

Use development endpoints to iteratively develop and test your extract, transform, and load (ETL) scripts in AWS Glue. You can add a development endpoint using the AWS Glue console or the AWS Command Line Interface (AWS CLI).

To add a development endpoint (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Sign in as a user who has the IAM permission `glue:CreateDevEndpoint`.
2. In the navigation pane, choose **Dev endpoints**, and then choose **Add endpoint**.
3. Follow the steps in the AWS Glue **Add endpoint** wizard to provide the properties that are required to create an endpoint. Specify an IAM role that permits access to your data.

If you choose to provide an SSH public key when you create your development endpoint, save the SSH private key to access the development endpoint later.

4. Choose **Finish** to complete the wizard. Then check the console for development endpoint status. When the status changes to **READY**, the development endpoint is ready to use.

When creating the endpoint, you can provide the following optional information:

Security configuration

To specify at-rest encryption options, add a security configuration to the development endpoint.

Worker type

The type of predefined worker that is allocated to the development endpoint. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory, a 50 GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, and a 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, and a 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.

Known issue: When you create a development endpoint with the G.2X `WorkerType` configuration, the Spark drivers for the development endpoint run on 4 vCPU, 16 GB of memory, and a 64 GB disk.

Number of workers

The number of workers of a defined `workerType` that are allocated to the development endpoint. This field is available only when you choose worker type G.1X or G.2X.

The maximum number of workers you can define is 299 for G.1X, and 149 for G.2X.

Data processing units (DPUs)

The number of DPUs that AWS Glue uses for your development endpoint. The number must be greater than 1.

Python library path

Comma-separated Amazon Simple Storage Service (Amazon S3) paths to Python libraries that are required by your script. Multiple values must be complete paths separated by a comma (,). Only individual files are supported, not a directory path.

Note

You can use only pure Python libraries. Libraries that rely on C extensions, such as the Pandas Python data analysis library, are not yet supported.

Dependent jars path

Comma-separated Amazon S3 paths to JAR files that are required by the script.

Note

Currently, you can use only pure Java or Scala (2.11) libraries.

Glue Version

Specifies the versions of Python and Apache Spark to use. Defaults to AWS Glue version 1.0 (Python version 3 and Spark version 2.4). For more information, see the [Glue version job property \(p. 165\)](#).

Tags

Tag your development endpoint with a **Tag key** and optional **Tag value**. After tag keys are created, they are read-only. Use tags on some resources to help you organize and identify them. For more information, see [AWS Tags in AWS Glue \(p. 229\)](#).

Spark UI

Enable the use of Spark UI for monitoring Spark applications running on this development endpoint. For more information, see [Enabling the Apache Spark Web UI for Development Endpoints \(p. 238\)](#).

Use Glue Data Catalog as the Hive metastore (under Catalog Options)

Enables you to use the AWS Glue Data Catalog as a Spark Hive metastore.

To add a development endpoint (AWS CLI)

1. In a command line window, enter a command similar to the following.

```
aws glue create-dev-endpoint --endpoint-name "endpoint1" --role-arn
    "arn:aws:iam::account-id:role/role-name" --number-of-nodes "3" --glue-version "1.0" --
    arguments '{"GLUE_PYTHON_VERSION": "3"}' --region "region-name"
```

This command specifies AWS Glue version 1.0. Because this version supports both Python 2 and Python 3, you can use the arguments parameter to indicate the desired Python version. If the glue-version parameter is omitted, AWS Glue version 0.9 is assumed. For more information about AWS Glue versions, see the [Glue version job property \(p. 165\)](#).

For information about additional command line parameters, see [create-dev-endpoint](#) in the *AWS CLI Command Reference*.

2. (Optional) Enter the following command to check the development endpoint status. When the status changes to `READY`, the development endpoint is ready to use.

```
aws glue get-dev-endpoint --endpoint-name "endpoint1"
```

Viewing Development Endpoint Properties

You can view development endpoint details using the AWS Glue console. Endpoint details include the information that you defined when you created it using the **Add endpoint** wizard. They also include

information that you need to connect to the endpoint and information about any notebooks that use the endpoint.

To view development endpoint properties

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Sign in as a user who has the IAM permissions `glue:GetDevEndpoints` and `glue:GetDevEndpoint`.
2. In the navigation pane, under **ETL**, choose **Dev endpoints**.
3. On the **Dev endpoints** page, choose the name of the development endpoint.

The following are some of the development endpoint properties:

Endpoint name

The unique name that you give the endpoint when you create it.

Provisioning status

Describes whether the endpoint is being created (**PROVISIONING**), ready to be used (**READY**), in the process of terminating (**TERMINATING**), terminated (**TERMINATED**), or failed (**FAILED**).

Failure reason

The reason for a development endpoint failure.

Private address

The address to connect to the development endpoint. On the Amazon EC2 console, you can view the elastic network interface that is attached to this IP address. This internal address is created if the development endpoint is associated with a virtual private cloud (VPC).

For more information about accessing a development endpoint from a private address, see [the section called "Accessing Your Development Endpoint" \(p. 189\)](#).

Public address

The address to connect to the development endpoint.

Public key contents

The current public SSH keys that are associated with the development endpoint (optional). If you provided a public key when you created the development endpoint, you should have saved the corresponding SSH private key.

IAM role

Specify the IAM role that is used for authorization to resources. If the development endpoint reads AWS KMS encrypted Amazon S3 data, the **IAM role** must have decrypt permission on the KMS key.

For more information about creating an IAM role, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#).

SSH to Python REPL

You can open a terminal window on your computer and enter this command to interact with the development endpoint as a read-eval-print loop (REPL) shell. This field is shown only if the development endpoint contains a public SSH key.

SSH to Scala REPL

You can open a terminal window and enter this command to interact with the development endpoint as a REPL shell. This field is shown only if the development endpoint contains a public SSH key.

SSH tunnel to remote interpreter

You can open a terminal window and enter this command to open a tunnel to the development endpoint. Then open your local Apache Zeppelin notebook and point to the development endpoint as a remote interpreter. When the interpreter is set up, all notes within the notebook can use it. This field is shown only if the development endpoint contains a public SSH key.

Public key update status

The status of completing an update of the public key on the development endpoint. When you update a public key, the new key must be propagated to the development endpoint. Status values include COMPLETED and PENDING.

Last modified time

The last time this development endpoint was modified.

Running for

The amount of time the development endpoint has been provisioned and READY.

Accessing Your Development Endpoint

When you create a development endpoint in a virtual private cloud (VPC), AWS Glue returns only a private IP address. The public IP address field is not populated. When you create a non-VPC development endpoint, AWS Glue returns only a public IP address.

If your development endpoint has a **Public address**, confirm that it is reachable with the SSH private key for the development endpoint, as in the following example.

```
ssh -i dev-endpoint-private-key.pem glue@public-address
```

Suppose that your development endpoint has a **Private address**, your VPC subnet is routable from the public internet, and its security groups allow inbound access from your client. In this case, follow these steps to attach an *Elastic IP address* to a development endpoint to allow access from the internet.

Note

If you want to use Elastic IP addresses, the subnet that is being used requires an internet gateway associated through the route table.

To access a development endpoint by attaching an Elastic IP address

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Dev endpoints**, and navigate to the development endpoint details page. Record the **Private address** for use in the next step.
3. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
4. In the navigation pane, under **Network & Security**, choose **Network Interfaces**.
5. Search for the **Private DNS (IPv4)** that corresponds to the **Private address** on the AWS Glue console development endpoint details page.

You might need to modify which columns are displayed on your Amazon EC2 console. Note the **Network interface ID (ENI)** for this address (for example, eni-12345678).

6. On the Amazon EC2 console, under **Network & Security**, choose **Elastic IPs**.
7. Choose **Allocate new address**, and then choose **Allocate** to allocate a new Elastic IP address.
8. On the **Elastic IPs** page, choose the newly allocated **Elastic IP**. Then choose **Actions, Associate address**.

9. On the **Associate address** page, do the following:
 - For **Resource type**, choose **Network interface**.
 - In the **Network interface** box, enter the **Network interface ID (ENI)** for the private address.
 - Choose **Associate**.
10. Confirm that the newly associated Elastic IP address is reachable with the SSH private key that is associated with the development endpoint, as in the following example.

```
ssh -i dev-endpoint-private-key.pem glue@elastic-ip
```

For information about using a bastion host to get SSH access to the development endpoint's private address, see the AWS Security Blog post [Securely Connect to Linux Instances Running in a Private Amazon VPC](#).

Creating a Notebook Server Hosted on Amazon EC2

You can install an Apache Zeppelin Notebook on your local machine and use it to debug and test ETL scripts on a development endpoint. Alternatively, you can host the Zeppelin Notebook server on an Amazon EC2 instance. For more information, see [the section called "Notebook Server Considerations" \(p. 210\)](#).

In the AWS Glue **Create notebook server** window, you add the properties that are required to create a notebook server to use an Apache Zeppelin notebook.

Note

For any notebook server that you create that is associated with a development endpoint, you manage it. Therefore, if you delete the development endpoint, to delete the notebook server, you must delete the AWS CloudFormation stack on the AWS CloudFormation console.

Important

Before you can use the notebook server hosted on Amazon EC2, you must run a script on the Amazon EC2 instance that does the following actions:

- Sets the Zeppelin notebook password.
- Sets up communication between the notebook server and the development endpoint.
- Verifies or generates a Secure Sockets Layer (SSL) certificate to access the Zeppelin notebook.

For more information, see [the section called "Notebook Server Considerations" \(p. 210\)](#).

You provide the following properties:

CloudFormation stack name

The name of your notebook that is created in the AWS CloudFormation stack on the development endpoint. The name is prefixed with `aws-glue-`. This notebook runs on an Amazon EC2 instance. The Zeppelin HTTP server is started either on public port 443 or localhost port 8080 that can be accessed with an SSH tunnel command.

IAM role

A role with a trust relationship to Amazon EC2 that matches the Amazon EC2 instance profile exactly. Create the role in the IAM console. Choose **Amazon EC2**, and attach a policy for the notebook, such as **AWSGlueServiceNotebookRoleDefault**. For more information, see [Step 5: Create an IAM Role for Notebook Servers \(p. 25\)](#).

For more information about instance profiles, see [Using Instance Profiles](#).

EC2 key pair

The Amazon EC2 key that is used to access the Amazon EC2 instance hosting the notebook server. You can create a key pair on the Amazon EC2 console (<https://console.aws.amazon.com/ec2/>). Save the key files for later use. For more information, see [Amazon EC2 Key Pairs](#).

Attach a public IP to the notebook server EC2 instance

Select this option to attach a public IP which can be used to access the notebook server from the internet. Whether you choose a public or private **Subnet** is a factor when deciding to select this option. In a public subnet, a notebook server requires a public IP to access the internet. If your notebook server is in a private subnet and you do not want a public IP, don't select this option. However, your notebook server still requires a route to the internet such as through a NAT gateway.

Notebook username

The user name that you use to access the Zeppelin notebook. The default is `admin`.

Notebook S3 path

The location where the state of the notebook is stored. The Amazon S3 path to the Zeppelin notebook must follow the format: `s3://bucket-name/username`. Subfolders cannot be included in the path. The default is `s3://aws-glue-notebooks-account-id-region/notebook-username`.

Subnet

The available subnets that you can use with your notebook server. An asterisk (*) indicates that the subnet can be accessed from the internet. The subnet must have a route to the internet through an internet gateway (IGW), NAT gateway, or VPN. For more information, see [Setting Up Your Environment for Development Endpoints \(p. 34\)](#).

Security groups

The available security groups that you can use with your notebook server. The security group must have inbound rules for HTTPS (port 443) and SSH (port 22). Ensure that the rule's source is either 0.0.0.0/0 or the IP address of the machine connecting to the notebook.

S3 AWS KMS key

A key used for client-side KMS encryption of the Zeppelin notebook storage on Amazon S3. This field is optional. Enable access to Amazon S3 by either choosing an AWS KMS key or choose **Enter a key ARN** and provide the Amazon Resource Name (ARN) for the key. Type the ARN in the form `arn:aws:kms:region:account-id:key/key-id`. You can also provide the ARN in the form of a key alias, such as `arn:aws:kms:region:account-id:alias/alias-name`.

Custom AMI ID

A custom Amazon Machine Image (AMI) ID of an encrypted Amazon Elastic Block Storage (EBS) EC2 instance. This field is optional. Provide the AMI ID by either choosing an AMI ID or choose **Enter AMI ID** and type the custom AMI ID. For more information about how to encrypt your notebook server storage, see [Encryption and AMI Copy](#).

Notebook server tags

The AWS CloudFormation stack is always tagged with a key **aws-glue-dev-endpoint** and the value of the name of the development endpoint. You can add more tags to the AWS CloudFormation stack.

EC2 instance

The name of Amazon EC2 instance that is created to host your notebook. This links to the Amazon EC2 console (<https://console.aws.amazon.com/ec2/>) where the instance is tagged with the key **aws-glue-dev-endpoint** and value of the name of the development endpoint.

CloudFormation stack

The name of the AWS CloudFormation stack used to create the notebook server.

SSH to EC2 server command

Type this command in a terminal window to connect to the Amazon EC2 instance that is running your notebook server. The Amazon EC2 address shown in this command is either public or private depending on whether you chose to **Attach a public IP to the notebook server EC2 instance**.

Copy certificate

Example **scp** command to copy the keystore required to set up the Zeppelin notebook server to the Amazon EC2 instance that hosts the notebook server. Run the command from a terminal window in the directory where the Amazon EC2 private key is located. The key to access the Amazon EC2 instance is the parameter to the **-i** option. You provide the **path-to-keystore-file**. The location where the development endpoint private SSH key on the Amazon EC2 server is located is the remaining part of the command.

HTTPS URL

After completing the setup of a notebook server, type this URL in a browser to connect to your notebook using HTTPS.

Tutorial Setup: Prerequisites for the Development Endpoint Tutorials

Development endpoints create an environment where you can interactively test and debug ETL scripts in various ways before you run them as AWS Glue jobs. The tutorials in this section show you how to do this using different IDEs. All of them assume that you have set up a development endpoint and crawled sample data to create tables in your AWS Glue Data Catalog using the steps in the following sections.

Because you're using only Amazon Simple Storage Service (Amazon S3) data in some cases, and a mix of JDBC and Amazon S3 data in others, you will set up one development endpoint that is not in a virtual private cloud (VPC) and one that is.

Crawling the Sample Data Used in the Tutorials

The first step is to create a crawler that can crawl some sample data and record metadata about it in tables in your Data Catalog. The sample data that is used is drawn from <http://everypolitician.org/> and has been modified slightly for purposes of the tutorials. It contains data in JSON format about United States legislators and the seats that they have held in the US House of Representatives and Senate.

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
In the AWS Glue console, choose **Databases** in the navigation pane, and then choose **Add database**. Name the database **legislators**.
2. Choose **Crawlers**, and then choose **Add crawler**. Name the crawler **legislator_crawler**, and then choose **Next**.
3. Accept the default crawler source type (**Data stores**) and click **Next**.
4. Leave **S3** as the data store. Under **Crawl data in**, choose **Specified path in another account**. Then in the **Include path** box, enter **s3://awsglue-datasets/examples/us-legislators/all**. Choose **Next**, and then choose **Next** again to confirm that you don't want to add another data store.
5. Provide an IAM role for the crawler to assume when it runs.

- Provide a role that can access s3://awsglue-datasets/examples/us-legislators/all, or choose **Create an IAM role** and enter a name to create a role that has access to that location.
6. Choose **Next**, and then choose **Next** again to confirm that this crawler will be run on demand.
 7. For **Database**, choose the legislators database. Choose **Next**, and then choose **Finish** to complete the creation of the new crawler.
 8. Choose **Crawlers** in the navigation pane again. Select the check box next to the new legislator_crawler crawler, and choose **Run crawler**.
 9. Choose **Databases** in the navigation pane. Choose the legislators database, and then choose **Tables in legislators**. You should see six tables created by the crawler in your Data Catalog, containing metadata that the crawler retrieved.

Creating a Development Endpoint for Amazon S3 Data

The next thing to do is to create a development endpoint for Amazon S3 data. When you use a JDBC data source or target, the development endpoint must be created with a VPC. However, this isn't necessary in this tutorial if you are only accessing Amazon S3.

1. In the AWS Glue console, choose **Dev endpoints**. Choose **Add endpoint**.
2. Specify an endpoint name, such as **demo-endpoint**.
3. Choose an **IAM role** with permissions similar to the IAM role that you use to run AWS Glue ETL jobs. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#). Choose **Next**.
4. In **Networking**, leave **Skip networking information** selected, and choose **Next**.
5. In **SSH Public Key**, enter a public key generated by an SSH key generator program, such as **ssh-keygen** (do not use an Amazon EC2 key pair). The generated public key will be imported into your development endpoint. Save the corresponding private key to later connect to the development endpoint using SSH. Choose **Next**. For more information, see [ssh-keygen](#) in Wikipedia.

Note

When generating the key on Microsoft Windows, use a current version of PuTTYgen and paste the public key into the AWS Glue console from the PuTTYgen window. Generate an RSA key. Do not upload a file with the public key, instead use the key generated in the field **Public key for pasting into OpenSSH authorized_keys file**. The corresponding private key (.ppk) can be used in PuTTY to connect to the development endpoint. To connect to the development endpoint with SSH on Windows, convert the private key from .ppk format to OpenSSH .pem format using the PuTTYgen **Conversion** menu. For more information, see [Connecting to Your Linux Instance from Windows Using PuTTY](#).

6. In **Review**, choose **Finish**. After the development endpoint is created, wait for its provisioning status to move to **READY**.

Creating an Amazon S3 Location to Use for Output

If you don't already have a bucket, follow the instructions in [Create a Bucket](#) to set one up in Amazon S3 where you can save output from sample ETL scripts.

Creating a Development Endpoint with a VPC

Although not required for this tutorial, a VPC development endpoint is needed if both Amazon S3 and JDBC data stores are accessed by your ETL statements. In this case, when you create a development endpoint you specify network properties of the virtual private cloud (Amazon VPC) that contains your JDBC data stores. Before you start, set up your environment as explained in [Setting Up Your Environment for Development Endpoints \(p. 34\)](#).

1. In the AWS Glue console, choose **Dev endpoints** in the navigation pane. Then choose **Add endpoint**.

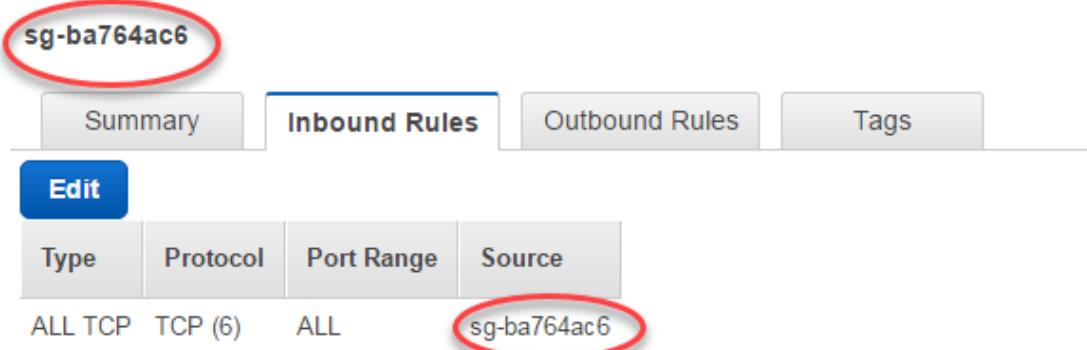
2. Specify an endpoint name, such as **vpc-demo-endpoint**.
3. Choose an **IAM role** with permissions similar to the IAM role that you use to run AWS Glue ETL jobs. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#). Choose **Next**.
4. In **Networking**, specify an Amazon VPC, a subnet, and security groups. This information is used to create a development endpoint that can connect to your data resources securely. Consider the following suggestions when filling in the properties of your endpoint:
 - If you already set up a connection to your data stores, you can use the same connection to determine the Amazon VPC, subnet, and security groups for your endpoint. Otherwise, specify these parameters individually.
 - Ensure that your Amazon VPC has **Edit DNS hostnames** set to **yes**. This parameter can be set in the Amazon VPC console (<https://console.aws.amazon.com/vpc/>). For more information, see [Setting Up DNS in Your VPC \(p. 29\)](#).
 - For this tutorial, ensure that the Amazon VPC you select has an Amazon S3 VPC endpoint. For information about how to create an Amazon S3 VPC endpoint, see [Amazon VPC Endpoints for Amazon S3 \(p. 30\)](#).
 - Choose a public subnet for your development endpoint. You can make a subnet a public subnet by adding a route to an internet gateway. For IPv4 traffic, create a route with **Destination** `0.0.0.0/0` and **Target** the internet gateway ID. Your subnet's route table should be associated with an internet gateway, not a NAT gateway. This information can be set in the Amazon VPC console (<https://console.aws.amazon.com/vpc/>). For example:



Route Table: rtb-547ccc3e	
Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-e07ccc8a

For more information, see [Route tables for Internet Gateways](#). For information about how to create an internet gateway, see [Internet Gateways](#).

- Ensure that you choose a security group that has an inbound self-reference rule. This information can be set in the Amazon VPC console (<https://console.aws.amazon.com/vpc/>). For example:



sg-ba764ac6			
Summary	Inbound Rules	Outbound Rules	Tags
Edit			
Type	Protocol	Port Range	Source
ALL TCP	TCP (6)	ALL	sg-ba764ac6

For more information about how to set up your subnet, see [Setting Up Your Environment for Development Endpoints \(p. 34\)](#).

Choose **Next**.

5. In **SSH Public Key**, enter a public key generated by an SSH key generator program (do not use an Amazon EC2 key pair). Save the corresponding private key to later connect to the development endpoint using SSH. Choose **Next**.

Note

When generating the key on Microsoft Windows, use a current version of PuTTYgen and paste the public key into the AWS Glue console from the PuTTYgen window. Generate an RSA key. Do not upload a file with the public key, instead use the key generated in the field **Public key for pasting into OpenSSH authorized_keys file**. The corresponding private key (.ppk) can be used in PuTTY to connect to the development endpoint. To connect to the development endpoint with SSH on Windows, convert the private key from .ppk format to OpenSSH .pem format using the PuTTYgen **Conversion** menu. For more information, see [Connecting to Your Linux Instance from Windows Using PuTTY](#).

6. In **Review**, choose **Finish**. After the development endpoint is created, wait for its provisioning status to move to **READY**.

You are now ready to try out the tutorials in this section:

- [Tutorial: Set Up a Local Apache Zeppelin Notebook to Test and Debug ETL Scripts \(p. 195\)](#)
- [Tutorial: Set Up an Apache Zeppelin Notebook Server on Amazon EC2 \(p. 198\)](#)
- [Tutorial: Use a REPL Shell with Your Development Endpoint \(p. 202\)](#)

Tutorial: Set Up a Local Apache Zeppelin Notebook to Test and Debug ETL Scripts

In this tutorial, you connect an Apache Zeppelin Notebook on your local machine to a development endpoint so that you can interactively run, debug, and test AWS Glue ETL (extract, transform, and load) scripts before deploying them. This tutorial uses SSH port forwarding to connect your local machine to an AWS Glue development endpoint. For more information, see [Port forwarding](#) in Wikipedia.

The tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 192\)](#).

Installing an Apache Zeppelin Notebook

1. Make sure that you have Java Development Kit 1.7 installed on your local machine (see [the Java home page](#)).

If you are running on Microsoft Windows, make sure that the JAVA_HOME environment variable points to the right Java directory. It's possible to update Java without updating this variable, and if it points to a folder that no longer exists, Zeppelin fails to start.

2. Download the version of Apache Zeppelin with all interpreters from [the Zeppelin download page](#) onto your local machine. Choose the file to download according to the following compatibility table, and follow the download instructions.

Glue Version	Zeppelin Release	File To Download
0.9	0.7.3	zeppelin-0.7.3-bin-all.tgz
1.0	0.8.1	zeppelin-0.8.1-bin-all.tgz

Start Zeppelin in the way that's appropriate for your operating system. Leave the terminal window that starts the notebook server open while you are using Zeppelin. When the server has started successfully, you can see a line in the console that ends with "Done, zeppelin server started."

3. Open Zeppelin in your browser by navigating to `http://localhost:8080`.
4. In Zeppelin in the browser, open the drop-down menu at **anonymous** in the upper-right corner of the page, and choose **Interpreter**. On the interpreters page, search for **spark**, and choose **edit** on the right. Make the following changes:
 - Select the **Connect to existing process** check box, and then set **Host** to `localhost` and **Port** to `9007` (or whatever other port you are using for port forwarding).
 - In **Properties**, set **master** to `yarn-client`.
 - If there is a `spark.executor.memory` property, delete it by choosing the **x** in the **action** column.
 - If there is a `spark.driver.memory` property, delete it by choosing the **x** in the **action** column.

Choose **Save** at the bottom of the page, and then choose **OK** to confirm that you want to update the interpreter and restart it. Use the browser back button to return to the Zeppelin start page.

Initiating SSH Port Forwarding to Connect to Your DevEndpoint

Next, use SSH local port forwarding to forward a local port (here, `9007`) to the remote destination defined by AWS Glue (`169.254.76.1:9007`).

Open a terminal window that gives you access to the SSH secure-shell protocol. On Microsoft Windows, you can use the BASH shell provided by [Git for Windows](#), or install [Cygwin](#).

Run the following SSH command, modified as follows:

- Replace `private-key-file-path` with a path to the `.pem` file that contains the private key corresponding to the public key that you used to create your development endpoint.
- If you are forwarding a different port than `9007`, replace `9007` with the port number that you are actually using locally. The address, `169.254.76.1:9007`, is the remote port and not changed by you.
- Replace `dev-endpoint-public-dns` with the public DNS address of your development endpoint. To find this address, navigate to your development endpoint in the AWS Glue console, choose the name, and copy the **Public address** that's listed in the **Endpoint details** page.

```
ssh -i private-key-file-path -NTL 9007:169.254.76.1:9007 glue@dev-endpoint-public-dns
```

You will likely see a warning message like the following:

```
The authenticity of host 'ec2-xx-xxx-xxx-xx.us-west-2.compute.amazonaws.com
(xx.xxx.xxx.xx)'
can't be established.  ECDSA key fingerprint is SHA256:4e97875Brt+1wKzRko
+JflSnp21X7aTP3BcFnHYLEts.
Are you sure you want to continue connecting (yes/no)?
```

Type **yes** and leave the terminal window open while you use your Zeppelin notebook.

Running a Simple Script Fragment in a Notebook Paragraph

In the Zeppelin start page, choose **Create new note**. Name the new note **Legislators**, and confirm **spark** as the interpreter.

Type the following script fragment into your notebook and run it. It uses the person's metadata in the AWS Glue Data Catalog to create a DynamicFrame from your sample data. It then prints out the item count and the schema of this data.

```
%pyspark
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *

# Create a Glue context
glueContext = GlueContext(SparkContext.getOrCreate())

# Create a DynamicFrame using the 'persons_json' table
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")

# Print out information about this data
print "Count: ", persons_DyF.count()
persons_DyF.printSchema()
```

The output of the script is as follows:

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

Troubleshooting Your Local Notebook Connection

- If you encounter a *connection refused* error, you might be using a development endpoint that is out of date. Try creating a new development endpoint and reconnecting.
- If your connection times out or stops working for any reason, you may need to take the following steps to restore it:

1. In Zeppelin, in the drop-down menu in the upper-right corner of the page, choose **Interpreters**. On the interpreters page, search for **spark**. Choose **edit**, and clear the **Connect to existing process** check box. Choose **Save** at the bottom of the page.
2. Initiate SSH port forwarding as described earlier.
3. In Zeppelin, re-enable the spark interpreter's **Connect to existing process** settings, and then save again.

Resetting the interpreter like this should restore the connection. Another way to accomplish this is to choose **restart** for the Spark interpreter on the **Interpreters** page. Then wait for up to 30 seconds to ensure that the remote interpreter has restarted.

- Ensure your development endpoint has permission to access the remote Zeppelin interpreter. Without the proper networking permissions you might encounter errors such as open failed: connect failed: Connection refused.

Tutorial: Set Up an Apache Zeppelin Notebook Server on Amazon EC2

In this tutorial, you create an Apache Zeppelin Notebook server that is hosted on an Amazon EC2 instance. The notebook connects to one of your development endpoints so that you can interactively run, debug, and test AWS Glue ETL (extract, transform, and load) scripts before deploying them.

The tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 192\)](#).

Creating an Apache Zeppelin Notebook Server on an Amazon EC2 Instance

To create a notebook server on Amazon EC2, you must have permission to create resources in AWS CloudFormation, Amazon EC2, and other services. For more information about required user permissions, see [Step 3: Attach a Policy to IAM Users That Access AWS Glue \(p. 15\)](#).

1. On the AWS Glue console, choose **Dev endpoints** to go to the development endpoints list.
2. Choose an endpoint by selecting the box next to it. Choose an endpoint with an empty SSH public key because the key is generated with a later action on the Amazon EC2 instance. Then choose **Actions**, and choose **Create notebook server**.

To host the notebook server, an Amazon EC2 instance is spun up using an AWS CloudFormation stack on your development endpoint. If you create the Zeppelin server with an SSL certificate, the Zeppelin HTTPS server is started on port 443.

3. Enter an AWS CloudFormation stack server name such as `demo-cf`, using only alphanumeric characters and hyphens.
4. Choose an IAM role that you have set up with a trust relationship to Amazon EC2, as documented in [Step 5: Create an IAM Role for Notebook Servers \(p. 25\)](#).
5. Choose an Amazon EC2 key pair that you have generated on the Amazon EC2 console (<https://console.aws.amazon.com/ec2/>), or choose **Create EC2 key pair** to generate a new one. Remember where you have downloaded and saved the private key portion of the pair. This key pair is different from the SSH key you used when creating your development endpoint (the keys that Amazon EC2 uses are 2048-bit SSH-2 RSA keys). For more information about Amazon EC2 keys, see [Amazon EC2 Key Pairs](#).

It is generally a good practice to ensure that the private-key file is write-protected so that it is not accidentally modified. On macOS and Linux systems, do this by opening a terminal and entering `chmod 400 private-key-file path`. On Windows, open the console and enter `attrib -r private-key-file path`.

6. Choose a user name to access your Zeppelin notebook.
7. Choose an Amazon S3 path for your notebook state to be stored in.
8. Choose **Create**.

You can view the status of the AWS CloudFormation stack in the AWS CloudFormation console **Events** tab (<https://console.aws.amazon.com/cloudformation>). You can view the Amazon EC2 instances created by AWS CloudFormation in the Amazon EC2 console (<https://console.aws.amazon.com/ec2/>). Search for instances that are tagged with the key name **aws-glue-dev-endpoint** and value of the name of your development endpoint.

After the notebook server is created, its status changes to **CREATE_COMPLETE** in the Amazon EC2 console. Details about your server also appear in the development endpoint details page. When the creation is complete, you can connect to a notebook on the new server.

To complete the setup of the Zeppelin notebook server, you must run a script on the Amazon EC2 instance. This tutorial requires that you upload an SSL certificate when you create the Zeppelin server on the Amazon EC2 instance. But there is also an SSH local port forwarding method to connect. For additional setup instructions, see [Creating a Notebook Server Associated with a Development Endpoint \(p. 210\)](#). When the creation is complete, you can connect to a notebook on the new server using HTTPS.

Note

For any notebook server that you create that is associated with a development endpoint, you manage it. Therefore, if you delete the development endpoint, to delete the notebook server, you must delete the AWS CloudFormation stack on the AWS CloudFormation console.

Connecting to Your Notebook Server on Amazon EC2

1. In the AWS Glue console, choose Dev endpoints to navigate to the development endpoints list. Choose the name of the development endpoint for which you created a notebook server. Choosing the name opens its details page.
2. On the **Endpoint details** page, copy the URL labeled **HTTPS URL** for your notebook server.
3. Open a web browser, and paste in the notebook server URL. This lets you access the server using HTTPS on port 443. Your browser may not recognize the server's certificate, in which case you have to override its protection and proceed anyway.
4. Log in to Zeppelin using the user name and password that you provided when you created the notebook server.

Running a Simple Script Fragment in a Notebook Paragraph

1. Choose **Create new note** and name it **Legislators**. Confirm **spark** as the **Default Interpreter**.
2. You can verify that your notebook is now set up correctly by typing the statement `spark.version` and running it. This returns the version of Apache Spark that is running on your notebook server.
3. Type the following script into the next paragraph in your notebook and run it. This script reads metadata from the `persons_json` table that your crawler created, creates a `DynamicFrame` from the underlying data, and displays the number of records and the schema of the data.

```
%pyspark
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
```

```
# Create a Glue context
glueContext = GlueContext(SparkContext.getOrCreate())

# Create a DynamicFrame using the 'persons_json' table
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")

# Print out information about this data
print "Count: ", persons_DyF.count()
persons_DyF.printSchema()
```

The output of the script should be:

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

Tutorial: Use an Amazon SageMaker Notebook with Your Development Endpoint

In AWS Glue, you can create a development endpoint and then create an Amazon SageMaker notebook to help develop your ETL and machine learning scripts. A SageMaker notebook is a fully managed machine learning compute instance running the Jupyter Notebook application.

The tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 192\)](#).

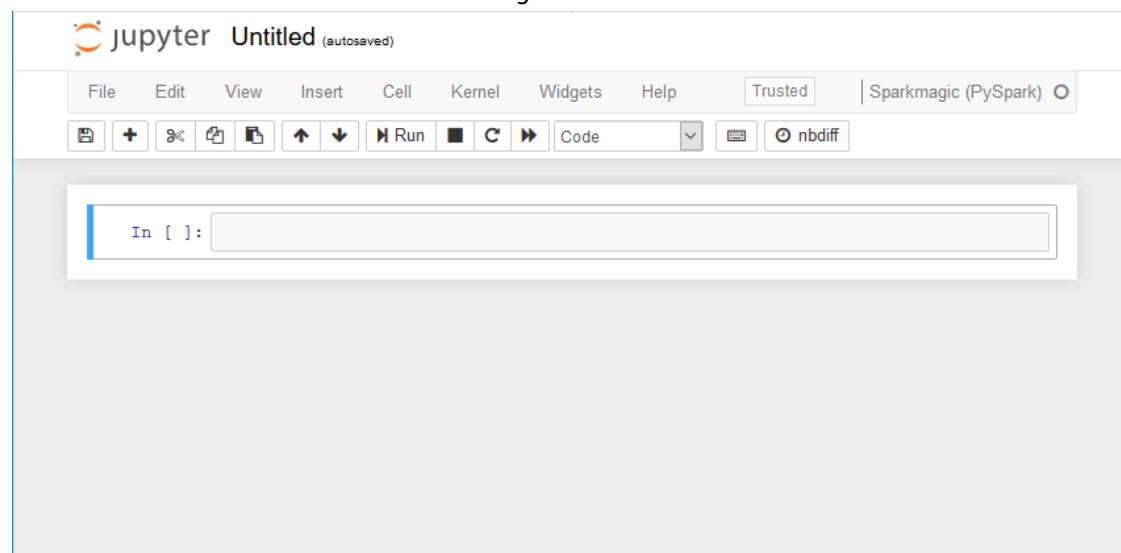
1. In the AWS Glue console, choose **Dev endpoints** to navigate to the development endpoints list.
2. Select the check box next to the name of a development endpoint that you want to use, and on the **Action** menu, choose **Create SageMaker notebook**.
3. Fill out the **Create and configure a notebook** page as follows:

- a. Enter a notebook name.
- b. Under **Attach to development endpoint**, verify the development endpoint.
- c. Create or choose an AWS Identity and Access Management (IAM) role.

Creating a role is recommended. If you use an existing role, ensure that it has the required permissions. For more information, see [the section called “Step 6: Create an IAM Policy for Amazon SageMaker Notebooks” \(p. 26\)](#).

- d. (Optional) Choose a VPC, a subnet, and one or more security groups.
- e. (Optional) Choose an AWS Key Management Service encryption key.
- f. (Optional) Add tags for the notebook instance.
4. Choose **Create notebook**. On the **Notebooks** page, choose the refresh icon at the upper right, and continue until the **Status** shows Ready.
5. Select the check box next to the new notebook name, and then choose **Open notebook**.
6. Create a new notebook: On the **jupyter** page, choose **New**, and then choose **Sparkmagic (PySpark)**.

Your screen should now look like the following:



7. (Optional) At the top of the page, choose **Untitled**, and give the notebook a name.
8. To start a Spark application, enter the following command into the notebook, and then in the toolbar, choose **Run**.

```
spark
```

After a short delay, you should see the following response:

```
In [1]: spark
Starting Spark application
      ID      YARN Application ID   Kind  State  Spark UI  Driver log  Current session?
      0  application_1576209965005_0001  pyspark  idle    Link      Link      ✓
SparkSession available as 'spark'.
<pyspark.sql.session.SparkSession object at 0x7f3d54913550>
```

9. Create a dynamic frame and run a query against it: Copy, paste, and run the following code, which outputs the count and schema of the `persons_json` table.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")
print ("Count: ", persons_DyF.count())
persons_DyF.printSchema()
```

Tutorial: Use a REPL Shell with Your Development Endpoint

In AWS Glue, you can create a development endpoint and then invoke a REPL (Read–Evaluate–Print Loop) shell to run PySpark code incrementally so that you can interactively debug your ETL scripts before deploying them.

The tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 192\)](#).

1. In the AWS Glue console, choose **Dev endpoints** to navigate to the development endpoints list. Choose the name of a development endpoint to open its details page.
2. Copy the SSH command labeled **SSH to Python REPL**, and paste it into a text editor. This field is only shown if the development endpoint contains a public SSH key. Replace the `<private-key.pem>` text with the path to the private-key .pem file that corresponds to the public key that you used to create the development endpoint. Use forward slashes rather than backslashes as delimiters in the path.
3. On your local computer, open a terminal window that can run SSH commands, and paste in the edited SSH command. Run the command.

Assuming that you accepted the default AWS Glue version 1.0 with Python 3 for the development endpoint, the output will look like this:

```
Python 3.6.8 (default, Aug  2 2019, 17:42:44)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/aws/glue/etl/jars/glue-assembly.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/spark/jars/slf4j-log4j12-1.7.16.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
2019-09-23 22:12:23,071 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading
libraries under SPARK_HOME.
2019-09-23 22:12:26,562 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66)) -
Same name resource file:/usr/lib/spark/python/lib/pyspark.zip added multiple times to
distributed cache
2019-09-23 22:12:26,580 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66)) -
Same path resource file:///usr/share/aws/glue/etl/python/PyGlue.zip added multiple
times to distributed cache.
2019-09-23 22:12:26,581 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66)) -
Same path resource file:///usr/lib/spark/python/lib/py4j-src.zip added multiple times
to distributed cache.
```

```
2019-09-23 22:12:26,581 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66)) -  
Same path resource file:///usr/share/aws/glue/libs/pyspark.zip added multiple times to  
distributed cache.  
Welcome to  
  
Using Python version 3.6.8 (default, Aug 2 2019 17:42:44)  
SparkSession available as 'spark'.  
>>>
```

4. Test that the REPL shell is working correctly by typing the statement, `print(spark.version)`. As long as that displays the Spark version, your REPL is now ready to use.
5. Now you can try executing the following simple script, line by line, in the shell:

```
import sys  
from pyspark.context import SparkContext  
from awsglue.context import GlueContext  
from awsglue.transforms import *  
glueContext = GlueContext(SparkContext.getOrCreate())  
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",  
    table_name="persons_json")  
print ("Count: ", persons_DyF.count())  
persons_DyF.printSchema()
```

Tutorial: Set Up PyCharm Professional with a Development Endpoint

This tutorial shows you how to connect the [PyCharm Professional](#) Python IDE running on your local machine to a development endpoint so that you can interactively run, debug, and test AWS Glue ETL (extract, transfer, and load) scripts before deploying them. The instructions and screen captures in the tutorial are based on PyCharm Professional version 2019.3.

To connect to a development endpoint interactively, you must have PyCharm Professional installed. You can't do this using the free edition.

The tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 192\)](#).

Note

The tutorial uses Amazon S3 as a data source. If you want to use a JDBC data source instead, you must run your development endpoint in a virtual private cloud (VPC). To connect with SSH to a development endpoint in a VPC, you must create an SSH tunnel. This tutorial does not include instructions for creating an SSH tunnel. For information on using SSH to connect to a development endpoint in a VPC, see [Securely Connect to Linux Instances Running in a Private Amazon VPC](#) in the AWS security blog.

Topics

- [Connecting PyCharm Professional to a Development Endpoint \(p. 204\)](#)
- [Deploying the Script to Your Development Endpoint \(p. 207\)](#)
- [Configuring a Remote Interpreter \(p. 208\)](#)
- [Running Your Script on the Development Endpoint \(p. 208\)](#)

Connecting PyCharm Professional to a Development Endpoint

1. Create a new pure-Python project in PyCharm named `legislators`.
2. Create a file named `get_person_schema.py` in the project with the following content:

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

def main():
    # Create a Glue context
    glueContext = GlueContext(SparkContext.getOrCreate())

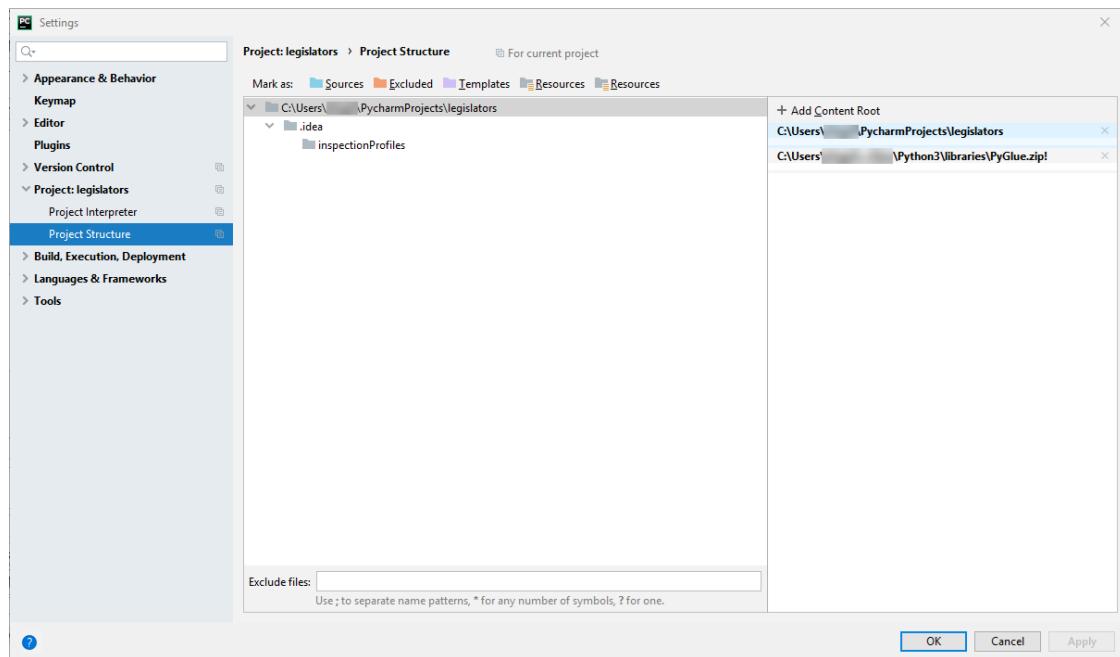
    # Create a DynamicFrame using the 'persons_json' table
    persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")

    # Print out information about this data
    print("Count: ", persons_DyF.count())
    persons_DyF.printSchema()

if __name__ == "__main__":
    main()
```

3. Do one of the following:
 - For Glue version 0.9, download the AWS Glue Python library file, `PyGlue.zip`, from <https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl/python/PyGlue.zip> to a convenient location on your local machine.
 - For Glue version 1.0, download the AWS Glue Python library file, `PyGlue.zip`, from <https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl-1.0/python/PyGlue.zip> to a convenient location on your local machine.
4. Add `PyGlue.zip` as a content root for your project in PyCharm:
 - In PyCharm, choose **File, Settings** to open the **Settings** dialog box. (You can also press **Ctrl+Alt+S**.)
 - Expand the `legislators` project and choose **Project Structure**. Then in the right pane, choose **+ Add Content Root**.
 - Navigate to the location where you saved `PyGlue.zip`, select it, then choose **Apply**.

The **Settings** screen should look something like the following:



Leave the **Settings** dialog box open after you choose **Apply**.

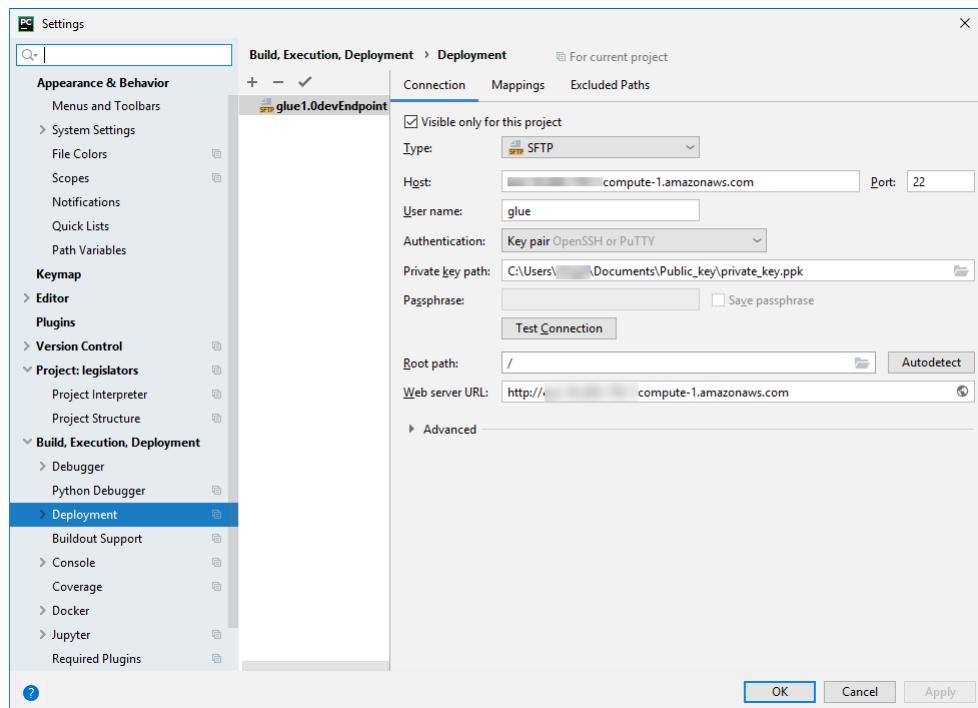
5. Configure deployment options to upload the local script to your development endpoint using SFTP (this capability is available only in PyCharm Professional):

- In the **Settings** dialog box, expand the **Build, Execution, Deployment** section. Choose the **Deployment** subsection.
- Choose the **+** icon at the top of the middle pane to add a new server. Set its **Type** to **SFTP** and give it a name.
- Set the **SFTP host** to the **Public address** of your development endpoint, as listed on its details page. (Choose the name of your development endpoint in the AWS Glue console to display the details page). For a development endpoint running in a VPC, set **SFTP host** to the host address and local port of your SSH tunnel to the development endpoint.
- Set the **User name** to `glue`.
- Set the **Auth type** to **Key pair (OpenSSH or Putty)**. Set the **Private key file** by browsing to the location where your development endpoint's private key file is located. Note that PyCharm only supports DSA, RSA and ECDSA OpenSSH key types, and does not accept keys in Putty's private format. You can use an up-to-date version of `ssh-keygen` to generate a key-pair type that PyCharm accepts, using syntax like the following:

```
ssh-keygen -t rsa -f <key_file_name> -C "<your_email_address>"
```

- Choose **Test connection**, and allow the connection to be tested. If the connection succeeds, choose **Apply**.

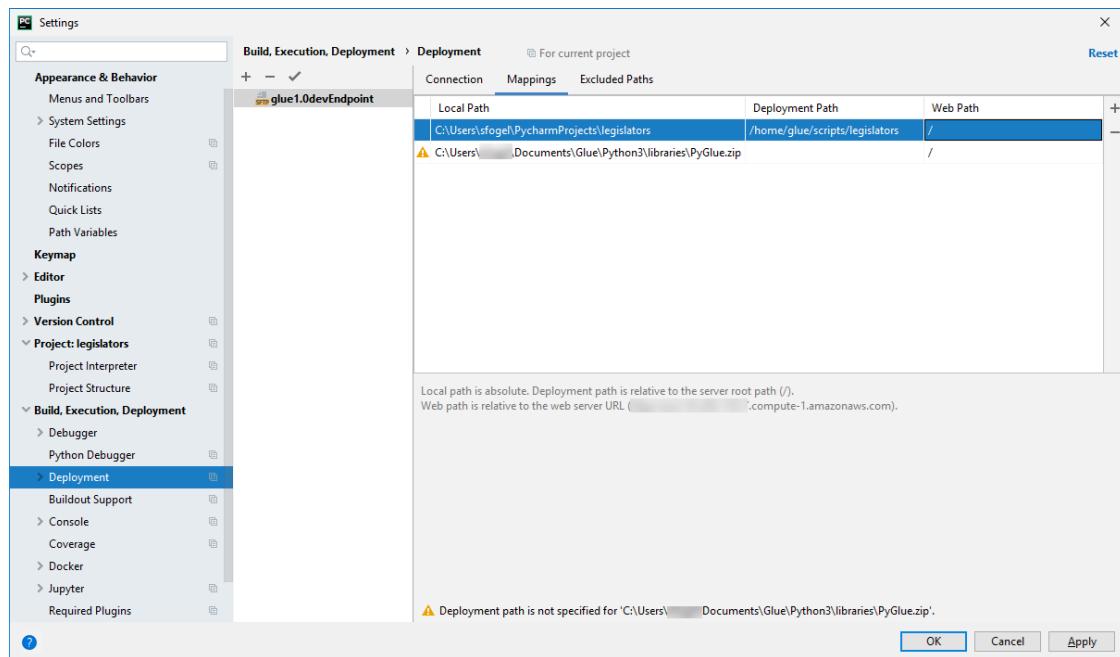
The **Settings** screen should now look something like the following:



Again, leave the **Settings** dialog box open after you choose **Apply**.

6. Map the local directory to a remote directory for deployment:
 - In the right pane of the **Deployment** page, choose the middle tab at the top, labeled **Mappings**.
 - In the **Deployment Path** column, enter a path under `/home/glue/scripts/` for deployment of your project path. For example: `/home/glue/scripts/legislators`.
 - Choose **Apply**.

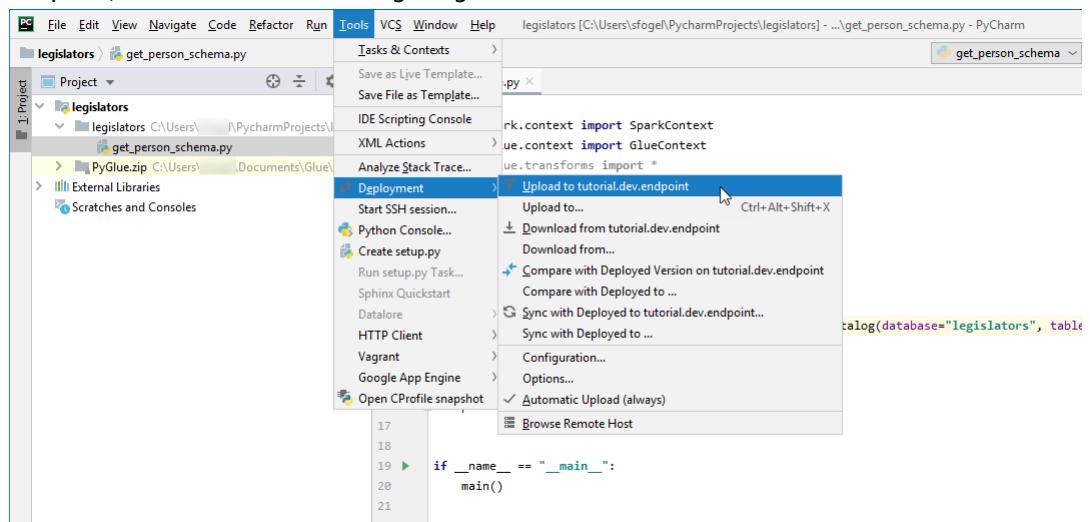
The **Settings** screen should now look something like the following:



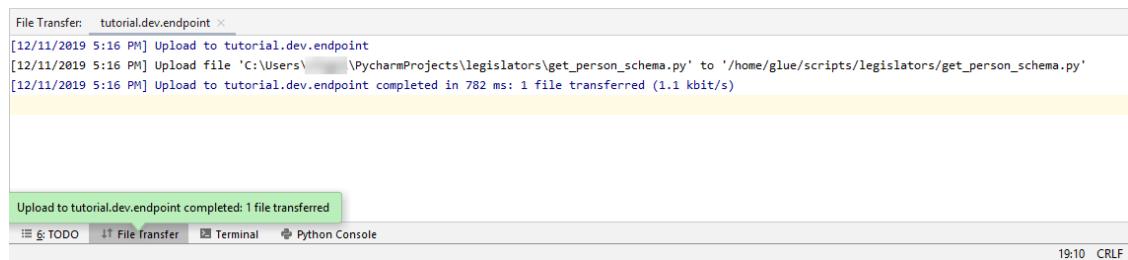
Choose **OK** to close the **Settings** dialog box.

Deploying the Script to Your Development Endpoint

1. Choose **Tools**, **Deployment**, and then choose the name under which you set up your development endpoint, as shown in the following image:



After your script has been deployed, the bottom of the screen should look something like the following:



2. On the menu bar, choose **Tools**, **Deployment**, **Automatic Upload (always)**. Ensure that a check mark appears next to **Automatic Upload (always)**.

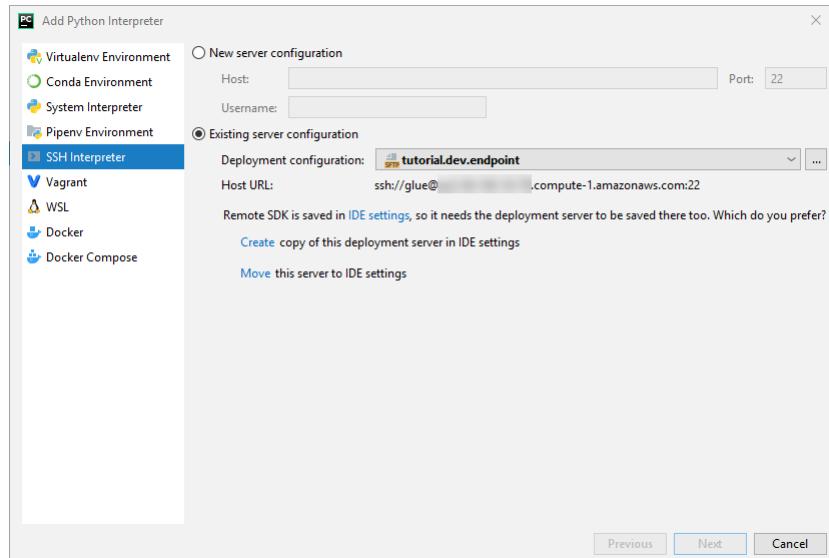
When this option is enabled, PyCharm automatically uploads changed files to the development endpoint.

Configuring a Remote Interpreter

Configure PyCharm to use the Python interpreter on the development endpoint.

1. From the **File** menu, choose **Settings**.
2. Expand the project **legislators** and choose **Project Interpreter**.
3. Choose the gear icon next to the **Project Interpreter** list, and then choose **Add**.
4. In the **Add Python Interpreter** dialog box, in the left pane, choose **SSH Interpreter**.
5. Choose **Existing server configuration**, and in the **Deployment configuration** list, choose your configuration.

Your screen should look something like the following image.



6. Choose **Move this server to IDE settings**, and then choose **Next**.
7. In the **Interpreter** field, change the path to `/usr/bin/gluepython` if you are using Python 2, or to `/usr/bin/gluepython3` if you are using Python 3. Then choose **Finish**.

Running Your Script on the Development Endpoint

To run the script:

- In the left pane, right-click the file name and choose **Run '<filename>'**.

After a series of messages, the final output should show the count and the schema.

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

```
Process finished with exit code 0
```

You are now set up to debug your script remotely on your development endpoint.

Managing Notebooks

A notebook enables interactive development and testing of your ETL (extract, transform, and load) scripts on a development endpoint. AWS Glue provides an interface to Amazon SageMaker notebooks and Apache Zeppelin notebook servers.

- Amazon SageMaker provides an integrated Jupyter authoring notebook instance. With AWS Glue, you create and manage Amazon SageMaker notebooks. You can also open Amazon SageMaker notebooks from the AWS Glue console.

In addition, you can use Apache Spark with Amazon SageMaker on AWS Glue development endpoints which support Amazon SageMaker (but not AWS Glue ETL jobs). SageMaker Spark is an open source Apache Spark library for Amazon SageMaker. For more information, see see [Using Apache Spark with Amazon SageMaker](#).

- Apache Zeppelin notebook servers are run on Amazon EC2 instances. You can create these instances on the AWS Glue console.

For more information about creating and accessing your notebooks using the AWS Glue console, see [Working with Notebooks on the AWS Glue Console \(p. 217\)](#).

For more information about creating development endpoints, see [Viewing Development Endpoint Properties \(p. 187\)](#).

Important

Managing Amazon SageMaker notebooks with AWS Glue development endpoints is available in the following AWS Regions:

Region	Code
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
AWS GovCloud (US-West)	us-gov-west-1

Topics

- [Creating a Notebook Server Associated with a Development Endpoint \(p. 210\)](#)
- [Working with Notebooks on the AWS Glue Console \(p. 217\)](#)

Creating a Notebook Server Associated with a Development Endpoint

One method for testing your ETL code is to use an Apache Zeppelin notebook running on an Amazon Elastic Compute Cloud (Amazon EC2) instance. When you use AWS Glue to create a notebook server on an Amazon EC2 instance, there are several actions you must take to set up your environment securely. The development endpoint is built to be accessed from a single client. To simplify your setup, start by creating a development endpoint that is used from a notebook server on Amazon EC2.

The following sections explain some of the choices to make and the actions to take to create a notebook server securely. These instructions perform the following tasks:

- Create a development endpoint.
- Spin up a notebook server on an Amazon EC2 instance.
- Securely connect a notebook server to a development endpoint.
- Securely connect a web browser to a notebook server.

Choices on the AWS Glue Console

For more information about managing development endpoints using the AWS Glue console, see [Viewing Development Endpoint Properties \(p. 187\)](#).

To create the development endpoint and notebook server

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Dev endpoints** in the navigation pane, and then choose **Add endpoint** to create a development endpoint.
3. Follow the steps in the wizard to create a development endpoint that you plan to associate with one notebook server running on Amazon EC2.

In the **Add SSH public key (optional)** step, leave the public key empty. In a later step, you generate and push a public key to the development endpoint and a corresponding private key to the Amazon EC2 instance that is running the notebook server.

4. When the development endpoint is provisioned, continue with the steps to create a notebook server on Amazon EC2. On the development endpoints list page, choose the development endpoint that you just created. Choose **Action, Create Zeppelin notebook server**, and fill in the information about your notebook server. (For more information, see [the section called "Viewing Development Endpoint Properties" \(p. 187\)](#).)
5. Choose **Finish**. The notebook server is created with an AWS CloudFormation stack. The AWS Glue console provides you with the information you need to access the Amazon EC2 instance.

After the notebook server is ready, you must run a script on the Amazon EC2 instance to complete the setup.

Actions on the Amazon EC2 Instance to Set Up Access

After you create the development endpoint and notebook server, complete the following actions to set up the Amazon EC2 instance for your notebook server.

To set up access to the notebook server

1. If your local desktop is running Windows, you need a way to run commands SSH and SCP to interact with the Amazon EC2 instance. You can find instructions for connecting in the Amazon EC2 documentation. For more information, see [Connecting to Your Linux Instance from Windows Using PuTTY](#).
2. You can connect to your Zeppelin notebook using an HTTPS URL. This requires a Secure Sockets Layer (SSL) certificate on your Amazon EC2 instance. The notebook server must provide web browsers with a certificate to validate its authenticity and to allow encrypted traffic for sensitive data such as passwords.

If you have an SSL certificate from a certificate authority (CA), copy your SSL certificate key store onto the Amazon EC2 instance into a path that the `ec2-user` has write access to, such as `/home/ec2-user/`. See the AWS Glue console notebook server details for the `scp` command to **Copy certificate**. For example, open a terminal window, and enter the following command:

```
scp -i ec2-private-key keystore.jks ec2-user@dns-address-of-ec2-instance:~/keystore.jks
```

The truststore, `keystore.jks`, that is copied to the Amazon EC2 instance must have been created with a password.

The `ec2-private-key` is the key needed to access the Amazon EC2 instance. When you created the notebook server, you provided an Amazon EC2 key pair and saved this EC2 private key to your local machine. You might need to edit the **Copy certificate** command to point to the key file on your local machine. You can also find this key file name on the Amazon EC2 console details for your notebook server.

The `dns-address-of-ec2-instance` is the address of the Amazon EC2 instance where the keystore is copied.

Note

There are many ways to generate an SSL certificate. It is a security best practice to use a certificate generated with a certificate authority (CA). You might need to enlist the help of an administrator in your organization to obtain the certificate. Follow the policies of your organization when you create a keystore for the notebook server. For more information, see [Certificate authority](#) in Wikipedia.

Another method is to generate a self-signed certificate with a script on your notebook server Amazon EC2 instance. However, with this method, each local machine that connects to the notebook server must be configured to trust the certificate generated before connecting to the notebook server. Also, when the generated certificate expires, a new certificate must be generated and trusted on all local machines. For more information about the setup, see [Self-signed certificates \(p. 214\)](#). For more information, see [Self-signed certificate](#) in Wikipedia.

3. Using SSH, connect to the Amazon EC2 instance that is running your notebook server; for example:

```
ssh -i ec2-private-key ec2-user@dns-address-of-ec2-instance
```

The `ec2-private-key` is the key that is needed to access the Amazon EC2 instance. When you created the notebook server, you provided an Amazon EC2 key pair and saved this EC2 private key to your local machine. You might need to edit the **Copy certificate** command to point to the key file on your local machine. You can also find this key file name on the Amazon EC2 console details for your notebook server.

The `dns-address-of-ec2-instance` is the address of the Amazon EC2 instance where the keystore is copied.

4. From the home directory, `/home/ec2-user/`, run the `./setup_notebook_server.py` script. AWS Glue created and placed this script on the Amazon EC2 instance. The script performs the following actions:

- **Asks for a Zeppelin notebook password:** The password is SHA-256 hashed plus salted-and-iterated with a random 128-bit salt kept in the `shiro.ini` file with restricted access. This is the best practice available to Apache Shiro, the authorization package that Apache Zeppelin uses.
- **Generates SSH public and private keys:** The script overwrites any existing SSH public key on the development endpoint that is associated with the notebook server. **As a result, any other notebook servers, Read-Eval-Print Loops (REPLs), or IDEs that connect to this development endpoint can no longer connect.**
- **Verifies or generates an SSL certificate:** Either use an SSL certificate that was generated with a certificate authority (CA) or generate a certificate with this script. If you copied a certificate,

the script asks for the location of the keystore file. Provide the entire path on the Amazon EC2 instance, for example, /home/ec2-user/keystore.jks. The SSL certificate is verified.

The following example output of the `setup_notebook_server.py` script generates a self-signed SSL certificate.

```
Starting notebook server setup. See AWS Glue documentation for more details.  
Press Enter to continue...  
  
Creating password for Zeppelin user admin  
Type the password required to access your Zeppelin notebook:  
Confirm password:  
Updating user credentials for Zeppelin user admin  
  
Zeppelin username and password saved.  
  
Setting up SSH tunnel to devEndpoint for notebook connection.  
Do you want a SSH key pair to be generated on the instance? WARNING this will replace  
any existing public key on the DevEndpoint [y/n] y  
Generating SSH key pair /home/ec2-user/dev.pem  
Generating public/private rsa key pair.  
Your identification has been saved in /home/ec2-user/dev.pem.  
Your public key has been saved in /home/ec2-user/dev.pem.pub.  
The key fingerprint is:  
26:d2:71:74:b8:91:48:06:e8:04:55:ee:a8:af:02:22 ec2-user@ip-10-0-0-142  
The key's randomart image is:  
+--[ RSA 2048 ]--  
| .o.oooo...o . |  
| o. ....+ . |  
| o . . .o . |  
| .o . o. . |  
| . o o S . |  
|E. . o . |  
|= . |  
|.. . |  
|o.. . |  
+-----+  
  
Attempting to reach AWS Glue to update DevEndpoint's public key. This might take a  
while.  
Waiting for DevEndpoint update to complete...  
Waiting for DevEndpoint update to complete...  
Waiting for DevEndpoint update to complete...  
DevEndpoint updated to use the public key generated.  
Configuring Zeppelin server...  
  
*****  
We will configure Zeppelin to be a HTTPS server. You can upload a CA signed certificate  
for the server to consume (recommended). Or you can choose to have a self-signed  
certificate created.  
See AWS Glue documentation for additional information on using SSL/TLS certificates.  
*****  
  
Do you have a JKS keystore to encrypt HTTPS requests? If not, a self-signed certificate  
will be generated. [y/n] n  
Generating self-signed SSL/TLS certificate at /home/ec2-user/  
ec2-192-0-2-0.compute-1.amazonaws.com.jks  
Self-signed certificates successfully generated.  
Exporting the public key certificate to /home/ec2-user/  
ec2-192-0-2-0.compute-1.amazonaws.com.der
```

```
Certificate stored in file /home/ec2-user/ec2-192-0-2-0.compute-1.amazonaws.com.der
Configuring Zeppelin to use the keystore for SSL connection...

Zeppelin server is now configured to use SSL.
SHA256
Fingerprint=53:39:12:0A:2B:A5:4A:37:07:A0:33:34:15:B7:2B:6F:ED:35:59:01:B9:43:AF:B9:50:55:E4:A2:8B:3B:59:E6.

*****
The public key certificate is exported to /home/ec2-user/
ec2-192-0-2-0.compute-1.amazonaws.com.der
The SHA-256 fingerprint for the certificate is
53:39:12:0A:2B:A5:4A:37:07:A0:33:34:15:B7:2B:6F:ED:35:59:01:B9:43:AF:B9:50:55:E4:A2:8B:3B:59:E6.
You may need it when importing the certificate to the client. See AWS Glue
documentation for more details.
*****


Press Enter to continue...


All settings done!


Starting SSH tunnel and Zeppelin...
autossh start/running, process 6074
Done. Notebook server setup is complete. Notebook server is ready.
See /home/ec2-user/zeppelin/logs/ for Zeppelin log files.
```

5. Check for errors with trying to start the Zeppelin server in the log files located at /home/ec2-user/zeppelin/logs/.

Actions on Your Local Computer to Connect to the Zeppelin Server

After you create the development endpoint and notebook server, connect to your Zeppelin notebook. Depending on how you set up your environment, you can connect in one of the following ways.

1. **Connect with a trusted CA certificate.** If you provided an SSL certificate from a certificate authority (CA) when the Zeppelin server was set up on the Amazon EC2 instance, choose this method. To connect with HTTPS on port 443, open a web browser and enter the URL for the notebook server. You can find this URL on the development notebook details page for your notebook server. Enter the contents of the **HTTPS URL** field; for example:

```
https://public-dns-address-of-ec2-instance:443
```

2. **Connect with a self-signed certificate.** If you ran the `setup_notebook_server.py` script to generate an SSL certificate, first trust the connection between your web browser and the notebook server. The details of this action vary by operating system and web browser. The general work flow is as follows:

1. Access the SSL certificate from the local computer. For some scenarios, this requires you to copy the SSL certificate from the Amazon EC2 instance to the local computer; for example:

```
scp -i path-to-ec2-private-key ec2-user@notebook-server-dns:/home/ec2-
user/notebook-server-dns.der notebook-server-dns.der
```

2. Import and view (or view and then import) the certificate into the certificate manager that is used by your operating system and browser. Verify that it matches the certificate generated on the Amazon EC2 instance.

Mozilla Firefox browser:

In Firefox, you might encounter an error like **Your connection is not secure**. To set up the connection, the general steps are as follows (the steps might vary by Firefox version):

1. Find the **Options or Preferences** page, navigate to the page and choose **View Certificates**. This option might appear in the **Privacy, Security, or Advanced** tab.
2. In the **Certificate Manager** window, choose the **Servers** tab, and then choose **Add Exception**.
3. Enter the **HTTPS Location** of the notebook server on Amazon EC2, and then choose **Get Certificate**. Choose **View**.
4. Verify that the **Common Name (CN)** matches the DNS of the notebook server Amazon EC2 instance. Also, verify that the **SHA-256 Fingerprint** matches that of the certificate generated on the Amazon EC2 instance. You can find the SHA-256 fingerprint of the certificate in the output of the `setup_notebook_server.py` script or by running an **openssl** command on the notebook instance.

```
openssl x509 -noout -fingerprint -sha256 -inform der -in path-to-certificate.der
```

5. If the values match, **confirm** to trust the certificate.
6. When the certificate expires, generate a new certificate on the Amazon EC2 instance and trust it on your local computer.

Google Chrome browser on macOS:

When using Chrome on macOS, you might encounter an error like **Your connection is not private**. To set up the connection, the general steps are as follows:

1. Copy the SSL certificate from the Amazon EC2 instance to your local computer.
2. Choose **Preferences or Settings** to find the **Settings** page. Navigate to the **Advanced** section, and then find the **Privacy and security** section. Choose **Manage certificates**.
3. In the **Keychain Access** window, navigate to the **Certificates** and choose **File, Import items** to import the SSL certificate.
4. Verify that the **Common Name (CN)** matches the DNS of the notebook server Amazon EC2 instance. Also, verify that the **SHA-256 Fingerprint** matches that of the certificate generated on the Amazon EC2 instance. You can find the SHA-256 fingerprint of the certificate in the output of the `setup_notebook_server.py` script or by running an **openssl** command on the notebook instance.

```
openssl x509 -noout -fingerprint -sha256 -inform der -in path-to-certificate.der
```

5. Trust the certificate by setting **Always Trust**.
6. When the certificate expires, generate a new certificate on the Amazon EC2 instance and trust it on your local computer.

Chrome browser on Windows:

When using Chrome on Windows, you might encounter an error like **Your connection is not private**. To set up the connection, the general steps are as follows:

1. Copy the SSL certificate from the Amazon EC2 instance to your local computer.
2. Find the **Settings** page, navigate to the **Advanced** section, and then find the **Privacy and security** section. Choose **Manage certificates**.
3. In the **Certificates** window, navigate to the **Trusted Root Certification Authorities** tab, and choose **Import** to import the SSL certificate.
4. Place the certificate in the **Certificate store for Trusted Root Certification Authorities**.
5. Trust by installing the certificate.
6. Verify that the **SHA-1 Thumbprint** that is displayed by the certificate in the browser matches that of the certificate generated on the Amazon EC2 instance. To find the certificate on the browser, navigate to the list of **Trusted Root Certification Authorities**, and choose the certificate **Issued To** the Amazon EC2 instance. Choose to **View** the certificate, choose **Details**, and then view the **Thumbprint** for sha1. You can find the corresponding SHA-1 fingerprint of the certificate by running an **openssl** command on the Amazon EC2 instance.

```
openssl x509 -noout -fingerprint -sha1 -inform der -in path-to-certificate.der
```

7. When the certificate expires, generate a new certificate on the Amazon EC2 instance and trust it on your local computer.

Microsoft Internet Explorer browser on Windows:

When using Internet Explorer on Windows, you might encounter an error like **Your connection is not private**. To set up the connection, the general steps are as follows:

1. Copy the SSL certificate from the Amazon EC2 instance to your local computer.
2. Find the **Internet Options** page, navigate to the **Content** tab, and then find the **Certificates** section.
3. In the **Certificates** window, navigate to the **Trusted Root Certification Authorities** tab, and choose **Import** to import the SSL certificate.
4. Place the certificate in the **Certificate store for Trusted Root Certification Authorities**.
5. Trust by installing the certificate.
6. Verify that the **SHA-1 Thumbprint** that is displayed by the certificate in the browser matches that of the certificate generated on the Amazon EC2 instance. To find the certificate on the browser, navigate to the list of **Trusted Root Certification Authorities**, and choose the certificate **Issued To** the Amazon EC2 instance. Choose to **View** the certificate, choose **Details**, and then view the **Thumbprint** for sha1. You can find the corresponding SHA-1 fingerprint of the certificate by running an **openssl** command on the Amazon EC2 instance.

```
openssl x509 -noout -fingerprint -sha1 -inform der -in path-to-certificate.der
```

7. When the certificate expires, generate a new certificate on the Amazon EC2 instance and trust it on your local computer.

After you trust the certificate, to connect with HTTPS on port 443, open a web browser and enter the URL for the notebook server. You can find this URL on the development notebook details page for your notebook server. Enter the contents of the **HTTPS URL** field; for example:

```
https://public-dns-address-of-ec2-instance:443
```

Working with Notebooks on the AWS Glue Console

A *development endpoint* is an environment that you can use to develop and test your AWS Glue scripts. A *notebook* enables interactive development and testing of your ETL (extract, transform, and load) scripts on a development endpoint.

AWS Glue provides an interface to Amazon SageMaker notebooks and Apache Zeppelin notebook servers. On the AWS Glue notebooks page, you can create Amazon SageMaker notebooks and attach them to a development endpoint. You can also manage Zeppelin notebook servers that you created and attached to a development endpoint. To create a Zeppelin notebook server, see [Creating a Notebook Server Hosted on Amazon EC2 \(p. 190\)](#).

The **Notebooks** page on the AWS Glue console lists all the Amazon SageMaker notebooks and Zeppelin notebook servers in your AWS Glue environment. You can use the console to perform several actions on your notebooks. To display details for a notebook or notebook server, choose the notebook in the list. Notebook details include the information that you defined when you created it using the **Create SageMaker notebook** or **Create Zeppelin Notebook server** wizard.

You can switch an Amazon SageMaker notebook that is attached to a development endpoint to another development endpoint as needed. The switch development endpoint action is only supported for Amazon SageMaker notebooks that were created after November 21, 2019.

To switch an Amazon SageMaker notebook to another development endpoint

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Notebooks**.
3. Choose the notebook in the list. Choose **Action**, and then **Switch Dev Endpoint**.
4. Choose an available development endpoint, and then choose **Apply**.

Certain IAM roles are required for this action. For more information, see [Create an IAM Policy for Amazon SageMaker Notebooks](#).

An Amazon SageMaker notebook periodically checks whether it is connected to the attached development endpoint. If it isn't connected, the notebook tries to reconnect automatically.

Amazon SageMaker Notebooks on the AWS Glue Console

The following are some of the properties for Amazon SageMaker notebooks. The console displays some of these properties when you view the details of a notebook.

Important

AWS Glue only manages Amazon SageMaker notebooks in certain AWS Regions. For more information, see [Managing Notebooks \(p. 209\)](#).

Before you begin, ensure that you have permissions to manage Amazon SageMaker notebooks on the AWS Glue console. For more information, see **AWSGlueConsoleSageMakerNotebookFullAccess** in [Step 3: Attach a Policy to IAM Users That Access AWS Glue \(p. 15\)](#).

Notebook name

The unique name of the Amazon SageMaker notebook.

Development endpoint

The name of the development endpoint that this notebook is attached to.

Important

This development endpoint must have been created after August 15, 2018.

Status

The provisioning status of the notebook and whether it is **Ready**, **Failed**, **Starting**, **Stopping**, or **Stopped**.

Failure reason

If the status is **Failed**, the reason for the notebook failure.

Instance type

The type of the instance used by the notebook.

IAM role

The IAM role that was used to create the Amazon SageMaker notebook.

This role has a trust relationship to Amazon SageMaker. You create this role on the AWS Identity and Access Management (IAM) console. When you are creating the role, choose **Amazon SageMaker**, and then attach a policy for the notebook, such as **AWSGlueServiceSageMakerNotebookRoleDefault**. For more information, see [Step 7: Create an IAM Role for Amazon SageMaker Notebooks \(p. 28\)](#).

Zeppelin Notebook Servers on the AWS Glue Console

The following are some of the properties for Apache Zeppelin notebook servers. The console displays some of these properties when you view the details of a notebook.

Notebook server name

The unique name of the Zeppelin notebook server.

Development endpoint

The unique name that you give the endpoint when you create it.

Provisioning status

Describes whether the notebook server is **CREATE_COMPLETE** or **ROLLBACK_COMPLETE**.

Failure reason

If the status is **Failed**, the reason for the notebook failure.

CloudFormation stack

The name of the AWS CloudFormation stack that was used to create the notebook server.

EC2 instance

The name of Amazon EC2 instance that is created to host your notebook. This links to the Amazon EC2 console (<https://console.aws.amazon.com/ec2/>), where the instance is tagged with the key **aws-glue-dev-endpoint** and value of the name of the development endpoint.

SSH to EC2 server command

Enter this command in a terminal window to connect to the Amazon EC2 instance that is running your notebook server. The Amazon EC2 address shown in this command is either public or private, depending on whether you chose to **Attach a public IP to the notebook server EC2 instance**.

Copy certificate

Example `scp` command to copy the keystore that is required to set up the Zeppelin notebook server to the Amazon EC2 instance that hosts the notebook server. Run the command from a terminal window in the directory where the Amazon EC2 private key is located. The key to access the Amazon

EC2 instance is the parameter to the `-i` option. You provide the `path-to-keystore-file`. The remaining part of the command is the location where the development endpoint private SSH key on the Amazon EC2 server is located.

HTTPS URL

After setting up a notebook server, enter this URL in a browser to connect to your notebook using HTTPS.

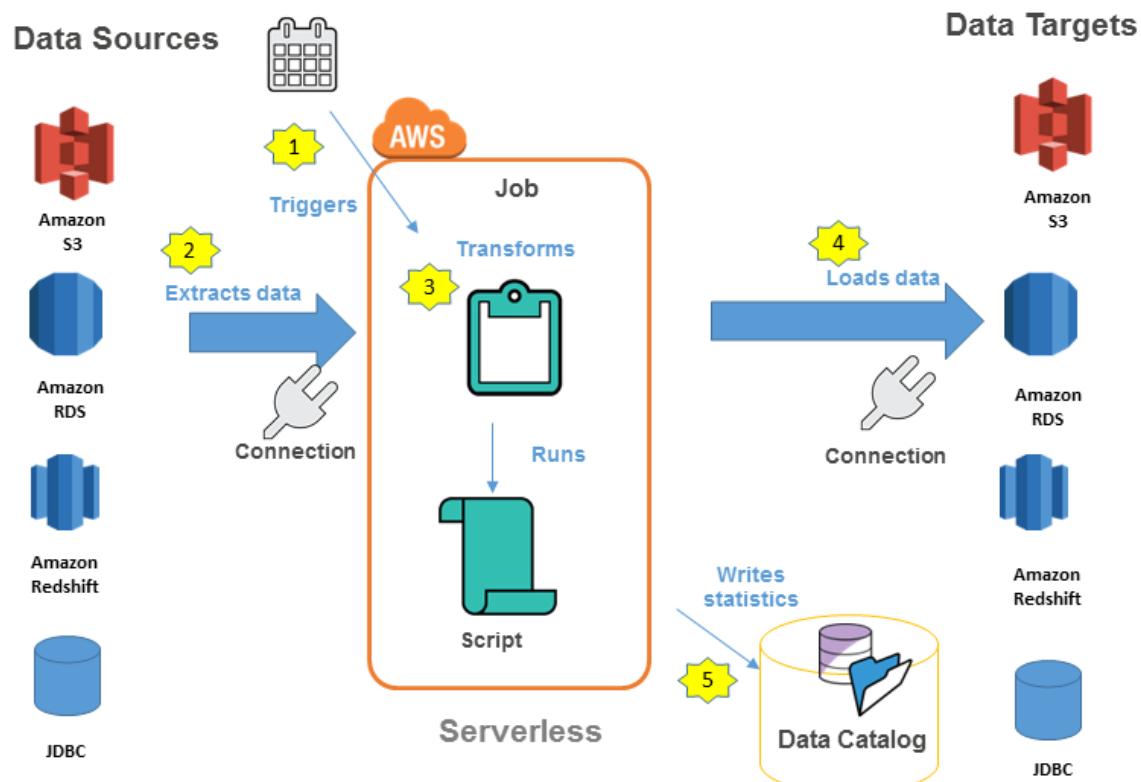
Running and Monitoring AWS Glue

You can automate the running of your ETL (extract, transform, and load) jobs. AWS Glue also provides metrics for crawlers and jobs that you can monitor. After you set up the AWS Glue Data Catalog with the required metadata, AWS Glue provides statistics about the health of your environment. You can automate the invocation of crawlers and jobs with a time-based schedule based on cron. You can also trigger jobs when an event-based trigger fires.

The main objective of AWS Glue is to provide an easier way to extract and transform your data from source to target. To accomplish this objective, an ETL job follows these typical steps (as shown in the diagram that follows):

1. A trigger fires to initiate a job run. This event can be set up on a recurring schedule or to satisfy a dependency.
2. The job extracts data from your source. If required, connection properties are used to access your source.
3. The job transforms your data using a script that you created and the values of any arguments. The script contains the Scala or PySpark Python code that transforms your data.
4. The transformed data is loaded to your data targets. If required, connection properties are used to access the target.
5. Statistics are collected about the job run and are written to your Data Catalog.

The following diagram shows the ETL workflow containing these five steps.



Topics

- [Automated Monitoring Tools \(p. 221\)](#)
- [Time-Based Schedules for Jobs and Crawlers \(p. 221\)](#)
- [Tracking Processed Data Using Job Bookmarks \(p. 223\)](#)
- [AWS Tags in AWS Glue \(p. 229\)](#)
- [Automating AWS Glue with CloudWatch Events \(p. 231\)](#)
- [Monitoring Jobs Using the Apache Spark Web UI \(p. 233\)](#)
- [Monitoring with Amazon CloudWatch \(p. 242\)](#)
- [Job Monitoring and Debugging \(p. 259\)](#)
- [Logging AWS Glue API Calls with AWS CloudTrail \(p. 280\)](#)

Automated Monitoring Tools

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Glue and your other AWS solutions. AWS provides monitoring tools that you can use to watch AWS Glue, report when something is wrong, and take action automatically when appropriate:

You can use the following automated monitoring tools to watch AWS Glue and report when something is wrong:

- **Amazon CloudWatch Events** delivers a near real-time stream of system events that describe changes in AWS resources. CloudWatch Events enables automated event-driven computing. You can write rules that watch for certain events and trigger automated actions in other AWS services when these events occur. For more information, see the [Amazon CloudWatch Events User Guide](#).
- **Amazon CloudWatch Logs** enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts call AWS, the source IP address from which the calls are made, and when the calls occur. For more information, see the [AWS CloudTrail User Guide](#).

Time-Based Schedules for Jobs and Crawlers

You can define a time-based schedule for your crawlers and jobs in AWS Glue. The definition of these schedules uses the Unix-like **cron** syntax. You specify time in **Coordinated Universal Time (UTC)**, and the minimum precision for a schedule is 5 minutes.

Cron Expressions

Cron expressions have six required fields, which are separated by white space.

Syntax

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Fields	Values	Wildcards
Minutes	0–59	, - * /

Fields	Values	Wildcards
Hours	0–23	, - * /
Day-of-month	1–31	, - * ? / L W
Month	1–12 or JAN–DEC	, - * /
Day-of-week	1–7 or SUN–SAT	, - * ? / L
Year	1970–2199	, - * /

Wildcards

- The **,** (comma) wildcard includes additional values. In the Month field, JAN, FEB, MAR would include January, February, and March.
- The **-** (dash) wildcard specifies ranges. In the Day field, 1–15 would include days 1 through 15 of the specified month.
- The ***** (asterisk) wildcard includes all values in the field. In the Hours field, * would include every hour.
- The **/** (forward slash) wildcard specifies increments. In the Minutes field, you could enter **1/10** to specify every 10th minute, starting from the first minute of the hour (for example, the 11th, 21st, and 31st minute).
- The **?** (question mark) wildcard specifies one or another. In the Day-of-month field you could enter **7**, and if you didn't care what day of the week the seventh was, you could enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, **3W** specifies the day closest to the third weekday of the month.

Limits

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value in one of the fields, you must use a **?** (question mark) in the other.
- Cron expressions that lead to rates faster than 5 minutes are not supported.

Examples

When creating a schedule, you can use the following sample cron strings.

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	10	*	*	?	*	Run at 10:00 am (UTC) every day
15	12	*	*	?	*	Run at 12:15 pm (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 pm (UTC) every

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
						Monday through Friday
0	8	1	*	?	*	Run at 8:00 am (UTC) every first day of the month
0/15	*	*	*	?	*	Run every 15 minutes
0/10	*	?	*	MON-FRI	*	Run every 10 minutes Monday through Friday
0/5	8-17	?	*	MON-FRI	*	Run every 5 minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

For example to run on a schedule of every day at 12:15 UTC, specify:

```
cron(15 12 * * ? *)
```

Tracking Processed Data Using Job Bookmarks

AWS Glue tracks data that has already been processed during a previous run of an ETL job by persisting state information from the job run. This persisted state information is called a *job bookmark*. Job bookmarks help AWS Glue maintain state information and prevent the reprocessing of old data. With job bookmarks, you can process new data when rerunning on a scheduled interval. A job bookmark is composed of the states for various elements of jobs, such as sources, transformations, and targets. For example, your ETL job might read new partitions in an Amazon S3 file. AWS Glue tracks which partitions the job has processed successfully to prevent duplicate processing and duplicate data in the job's target data store.

Job bookmarks are implemented for JDBC data sources, the Relationalize transform, and some Amazon Simple Storage Service (Amazon S3) sources. The following table lists the Amazon S3 source formats that AWS Glue supports for job bookmarks.

AWS Glue version	Amazon S3 source formats
Version 0.9	JSON, CSV, Apache Avro, XML

AWS Glue version	Amazon S3 source formats
Version 1.0 and later	JSON, CSV, Apache Avro, XML, Parquet, ORC

For information about Glue versions, see [Defining Job Properties \(p. 165\)](#).

For JDBC sources, the following rules apply:

- For each table, AWS Glue uses one or more columns as bookmark keys to determine new and processed data. The bookmark keys combine to form a single compound key.
- You can specify the columns to use as bookmark keys. If you don't specify bookmark keys, AWS Glue by default uses the primary key as the bookmark key, provided that it is sequentially increasing or decreasing (with no gaps).
- If user-defined bookmarks keys are used, they must be strictly monotonically increasing or decreasing. Gaps are permitted.

Topics

- [Using Job Bookmarks in AWS Glue \(p. 224\)](#)
- [Using Job Bookmarks with the AWS Glue Generated Script \(p. 225\)](#)
- [Tracking Files Using Modification Timestamps \(p. 228\)](#)

Using Job Bookmarks in AWS Glue

The job bookmark option is passed as a parameter when the job is started. The following table describes the options for setting job bookmarks on the AWS Glue console.

Job bookmark	Description
Enable	Causes the job to update the state after a run to keep track of previously processed data. If your job has a source with job bookmark support, it will keep track of processed data, and when a job runs, it processes new data since the last checkpoint.
Disable	Job bookmarks are not used, and the job always processes the entire dataset. You are responsible for managing the output from previous job runs. This is the default.
Pause	<p>Process incremental data since the last successful run or the data in the range identified by the following sub-options, without updating the state of last bookmark. You are responsible for managing the output from previous job runs. The two sub-options are:</p> <ul style="list-style-type: none"> • job-bookmark-from <from-value> is the run ID which represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input is ignored. • job-bookmark-to <to-value> is the run ID which represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input excluding the input identified by the <from-value> is processed by the job. Any input later than this input is also excluded for processing. <p>The job bookmark state is not updated when this option set is specified.</p>

Job bookmark	Description
	The sub-options are optional, however when used both the sub-options needs to be provided.

For details about the parameters passed to a job on the command line, and specifically for job bookmarks, see [Special Parameters Used by AWS Glue \(p. 292\)](#).

For Amazon S3 input sources, AWS Glue job bookmarks check the last modified time of the objects to verify which objects need to be reprocessed. If your input source data has been modified since your last job run, the files are reprocessed when you run the job again.

You can rewind your job bookmarks for your Glue Spark ETL jobs to any previous job run. You can support data backfilling scenarios better by rewinding your job bookmarks to any previous job run, resulting in the subsequent job run reprocessing data only from the bookmarked job run.

If you intend to reprocess all the data using the same job, reset the job bookmark. To reset the job bookmark state, use the AWS Glue console, the [ResetJobBookmark Action \(Python: reset_job_bookmark\) \(p. 560\)](#) API operation, or the AWS CLI. For example, enter the following command using the AWS CLI:

```
aws glue reset-job-bookmark --job-name my-job-name
```

AWS Glue keeps track of job bookmarks by job. If you delete a job, the job bookmark is deleted.

In some cases, you might have enabled AWS Glue job bookmarks but your ETL job is reprocessing data that was already processed in an earlier run. For information about resolving common causes of this error, see [Troubleshooting Errors in AWS Glue \(p. 626\)](#).

Transformation Context

Many of the AWS Glue PySpark dynamic frame methods include an optional parameter named `transformation_ctx`, which is a unique identifier for the ETL operator instance. The `transformation_ctx` parameter is used to identify state information within a job bookmark for the given operator. Specifically, AWS Glue uses `transformation_ctx` to index the key to the bookmark state.

For job bookmarks to work properly, enable the job bookmark parameter and set the `transformation_ctx` parameter. If you don't pass in the `transformation_ctx` parameter, then job bookmarks are not enabled for a dynamic frame or a table used in the method. For example, if you have an ETL job that reads and joins two Amazon S3 sources, you might choose to pass the `transformation_ctx` parameter only to those methods that you want to enable bookmarks. If you reset the job bookmark for a job, it resets all transformations that are associated with the job regardless of the `transformation_ctx` used.

For more information about the `DynamicFrameReader` class, see [DynamicFrameReader Class \(p. 350\)](#). For more information about PySpark extensions, see [AWS Glue PySpark Extensions Reference \(p. 332\)](#).

Using Job Bookmarks with the AWS Glue Generated Script

This section describes more of the operational details of using job bookmarks. It also provides an example of a script that you can generate from AWS Glue when you choose a source and destination and run a job.

Job bookmarks store the states for a job. Each instance of the state is keyed by a job name and a version number. When a script invokes `job.init`, it retrieves its state and always gets the latest version. Within a state, there are multiple state elements, which are specific to each source, transformation, and sink instance in the script. These state elements are identified by a transformation context that is attached to the corresponding element (source, transformation, or sink) in the script. The state elements are saved atomically when `job.commit` is invoked from the user script. The script gets the job name and the control option for the job bookmarks from the arguments.

The state elements in the job bookmark are source, transformation, or sink-specific data. For example, suppose that you want to read incremental data from an Amazon S3 location that is being constantly written to by an upstream job or process. In this case, the script must determine what has been processed so far. The job bookmark implementation for the Amazon S3 source saves information so that when the job runs again, it can filter only the new objects using the saved information and recompute the state for the next run of the job. A timestamp is used to filter the new files.

In addition to the state elements, job bookmarks have a *run number*, an *attempt number*, and a *version number*. The run number tracks the run of the job, and the attempt number records the attempts for a job run. The job run number is a monotonically increasing number that is incremented for every successful run. The attempt number tracks the attempts for each run, and is only incremented when there is a run after a failed attempt. The version number increases monotonically and tracks the updates to a job bookmark.

Example

The following is an example of a generated script for an Amazon S3 data source. The portions of the script that are required for using job bookmarks are shown in bold and italics. For more information about these elements see the [GlueContext Class \(p. 352\)](#) API, and the [DynamicFrameWriter Class \(p. 349\)](#) API.

```
# Sample Script
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "database", table_name = "relatedqueries_csv", transformation_ctx =
"datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "database",
    table_name = "relatedqueries_csv", transformation_ctx = "datasource0")
## @type: ApplyMapping
## @args: [mapping = [("col0", "string", "name", "string"), ("col1", "string", "number",
    "string")], transformation_ctx = "applymapping1"]
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("col0", "string",
    "name", "string"), ("col1", "string", "number", "string")], transformation_ctx = "applymapping1")
## @type: DataSink
## @args: [connection_type = "s3", connection_options = {"path": "s3://input_path"}, format
= "json", transformation_ctx = "datasink2"]
```

```
## @return: datasink2
## @inputs: [frame = applymapping1]
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
    connection_type = "s3", connection_options = {"path": "s3://input_path"}, format =
    "json", transformation_ctx = "datasink2")

job.commit()
```

Example

The following is an example of a generated script for a JDBC source. The source table is an employee table with the empno column as the primary key. Although by default the job uses a sequential primary key as the bookmark key if no bookmark key is specified, because empno is not necessarily sequential—there could be gaps in the values—it does not qualify as a default bookmark key. Therefore, the script explicitly designates empno as the bookmark key. That portion of the code is shown in bold and italics.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "hr", table_name = "emp", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "hr", table_name
    = "emp", transformation_ctx = "datasource0", additional_options = {"jobBookmarkKeys":
["empno"], "jobBookmarksKeysSortOrder": "asc"})
## @type: ApplyMapping
## @args: [mapping = [("ename", "string", "ename", "string"), ("hrly_rate",
    "decimal(38,0)", "hrly_rate", "decimal(38,0)", "comm", "decimal(7,2)", "comm",
    "decimal(7,2)", "hiredate", "timestamp", "hiredate", "timestamp"), ("empno",
    "decimal(5,0)", "empno", "decimal(5,0)", "mgr", "decimal(5,0)", "mgr",
    "decimal(5,0)", "photo", "string", "photo", "string"), ("job", "string", "job",
    "string", "deptno", "decimal(3,0)", "deptno", "decimal(3,0)", "ssn",
    "decimal(9,0)", "ssn", "decimal(9,0)", "sal", "decimal(7,2)", "sal",
    "decimal(7,2))], transformation_ctx = "applymapping1"]
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("ename", "string",
    "ename", "string"), ("hrly_rate", "decimal(38,0)", "hrly_rate", "decimal(38,0)", "comm",
    "decimal(7,2)", "comm", "decimal(7,2)", "hiredate", "timestamp", "hiredate",
    "timestamp"), ("empno", "decimal(5,0)", "empno", "decimal(5,0)", "mgr", "decimal(5,0)", "mgr",
    "decimal(5,0)", "photo", "string", "photo", "string"), ("job", "string", "job",
    "string", "deptno", "decimal(3,0)", "deptno", "decimal(3,0)", "ssn",
    "decimal(9,0)", "ssn", "decimal(9,0)", "sal", "decimal(7,2)", "sal",
    "decimal(7,2))], transformation_ctx = "applymapping1")
## @type: DataSink
## @args: [connection_type = "s3", connection_options = {"path": "s3://hr/employees"}, format =
    "csv", transformation_ctx = "datasink2"]
## @return: datasink2
## @inputs: [frame = applymapping1]
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
    connection_type = "s3", connection_options = {"path": "s3://hr/employees"}, format =
    "csv", transformation_ctx = "datasink2")
```

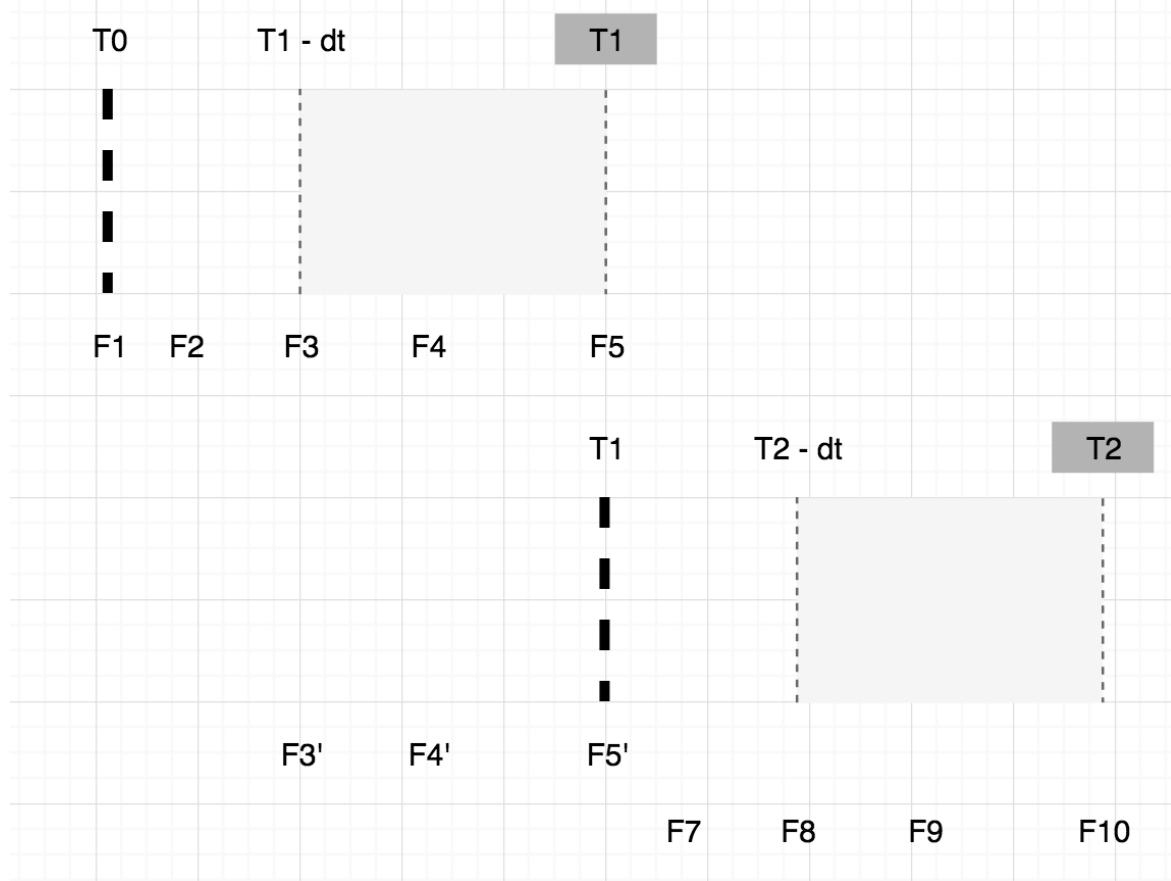
```
job.commit()
```

For more information about connection options related to job bookmarks, see [the section called “JDBC connections” \(p. 296\)](#).

Tracking Files Using Modification Timestamps

For Amazon S3 input sources, AWS Glue job bookmarks check the last modified time of the files to verify which objects need to be reprocessed.

Consider the following example. In the diagram, the X axis is a time axis, from left to right, with the left-most point being T0. The Y axis is list of files observed at time T. The elements representing the list are placed in the graph based on their modification time.



In this example, when a job starts at modification timestamp 1 (T1), it looks for files that have a modification time greater than T0 and less than or equal to T1. Those files are F2, F3, F4, and F5. The job bookmark stores the timestamps T0 and T1 as the low and high timestamps respectively.

When the job reruns at T2, it filters files that have a modification time greater than T1 and less than or equal to T2. Those files are F7, F8, F9, and F10. It thereby misses the files F3', F4', and F5'. The reason that the files F3', F4', and F5', which have a modification time less than or equal to T1, show up after T1 is because of Amazon S3 list consistency.

To account for Amazon S3 eventual consistency, AWS Glue includes a list of files (or path hash) in the job bookmark. AWS Glue assumes that the Amazon S3 file list is only inconsistent up to a finite period (dt) before the current time. That is, the file list for files with a modification time between T1 - dt and T1

when listing is done at T1 is inconsistent. However, the list of files with a modification time less than or equal to T1 - d1 is consistent at a time greater than or equal to T1.

You specify the period of time in which AWS Glue will save files (and where the files are likely to be consistent) by using the `MaxBatch` option in the AWS Glue connection options. The default value is 900 seconds (15 minutes). For more information about this property, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

When the job reruns at timestamp 2 (T2), it lists the files in the following ranges:

- T1 - dt (exclusive) to T1 (inclusive). This list includes F4, F5, F4', and F5'. This list is a consistent range. However, this range is inconsistent for a listing at T1 and has a list of files F3, F4, and F5 saved. For getting the files to be processed at T2, the files F3, F4, and F5 will be removed.
- T1 (exclusive) to T2 - dt (inclusive). This list includes F7 and F8. This list is a consistent range.
- T2 - dt (exclusive) - T2 (inclusive). This list includes F9 and F10. This list is an inconsistent range.

The resultant list of files is F3', F4', F5', F7, F8, F9, and F10.

The new files in the inconsistent list are F9 and F10, which are saved in the filter for the next run.

For more information about Amazon S3 eventual consistency, see [Introduction to Amazon S3 in the Amazon Simple Storage Service Developer Guide](#).

Job Run Failures

A job run version increments when a job fails. For example, if a job run at timestamp 1 (T1) fails, and it is rerun at T2, it advances the high timestamp to T2. Then, when the job is run at a later point T3, it advances the high timestamp to Amazon S3.

If a job run fails before the `job.commit()` (at T1), the files are processed in a subsequent run, in which AWS Glue processes the files from T0 to T2.

AWS Tags in AWS Glue

To help you manage your AWS Glue resources, you can optionally assign your own tags to some AWS Glue resource types. A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define. You can use tags in AWS Glue to organize and identify your resources. Tags can be used to create cost accounting reports and restrict access to resources. If you're using AWS Identity and Access Management, you can control which users in your AWS account have permission to create, edit, or delete tags. For more information, see [Identity-Based Policies \(IAM Policies\) with Tags \(p. 57\)](#).

In AWS Glue, you can tag the following resources:

- Crawler
- Job
- Trigger
- Development endpoint

Note

As a best practice, to allow tagging of these AWS Glue resources, always include the `glue:TagResource` action in your policies.

Consider the following when using tags with AWS Glue.

- A maximum of 50 tags are supported per entity.
- In AWS Glue, you specify tags as a list of key-value pairs in the format `{"string": "string" ...}`
- When you create a tag on an object, the tag key is required, and the tag value is optional.
- The tag key and tag value are case sensitive.
- The tag key and the tag value must not contain the prefix `aws`. No operations are allowed on such tags.
- The maximum tag key length is 128 Unicode characters in UTF-8. The tag key must not be empty or null.
- The maximum tag value length is 256 Unicode characters in UTF-8. The tag value may be empty or null.

Examples

The following examples create a job with assigned tags.

AWS CLI

```
aws glue create-job --name job-test-tags --role MyJobRole --command
Name=glueetl,ScriptLocation=S3://aws-glue-scripts//prod-job1 --tags '{"key1" : "value1",
"key2" : "value2"}'
```

AWS CloudFormation JSON

```
{
  "Description": "AWS Glue Job Test Tags",
  "Resources": {
    "MyJobRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "glue.amazonaws.com"
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
        "Policies": [
          {
            "PolicyName": "root",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": "*",
                  "Resource": "*"
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```

```
        ],
    },
},
"MyJob": {
    "Type": "AWS::Glue::Job",
    "Properties": {
        "Command": {
            "Name": "glueetl",
            "ScriptLocation": "s3://aws-glue-scripts//prod-job1"
        },
        "DefaultArguments": {
            "--job-bookmark-option": "job-bookmark-enable"
        },
        "ExecutionProperty": {
            "MaxConcurrentRuns": 2
        },
        "MaxRetries": 0,
        "Name": "cf-job1",
        "Role": {
            "Ref": "MyJobRole",
        },
        "Tags": {
            "key1": "value1", "key2": "value2"
        }
    }
}
}
```

For more information, see [AWS Tagging Strategies](#).

Note

Currently, AWS Glue does not support the Resource Groups Tagging API.

For information about how to control access using tags, see [Identity-Based Policies \(IAM Policies\) with Tags \(p. 57\)](#).

Automating AWS Glue with CloudWatch Events

You can use Amazon CloudWatch Events to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

Some examples of using CloudWatch Events with AWS Glue include the following:

- Activating a Lambda function when an ETL job succeeds
- Notifying an Amazon SNS topic when an ETL job fails

The following CloudWatch Events are generated by AWS Glue.

- Events for "detail-type": "Glue Job State Change" are generated for SUCCEEDED, FAILED, TIMEOUT, and STOPPED.
- Events for "detail-type": "Glue Job Run Status" are generated for RUNNING, STARTING, and STOPPING job runs when they exceed the job delay notification threshold.
- Events for "detail-type": "Glue Crawler State Change" are generated for Started, Succeeded, and Failed.
- Events for "detail-type": "Glue Data Catalog Database State Change" are generated for CreateDatabase, DeleteDatabase, CreateTable, DeleteTable and BatchDeleteTable. For example, if a table is created or deleted, a notification is sent to CloudWatch Events. Note that you cannot write a program that depends on the order or existence of notification events, as they might be out of sequence or missing. Events are emitted on a best effort basis. In the details of the notification:
 - The typeOfChange contains the name of the API operation.
 - The databaseName contains the name of the affected database.
 - The changedTables contains up to 100 names of affected tables per notification. When table names are long, multiple notifications might be created.
- Events for "detail-type": "Glue Data Catalog Table State Change" are generated for UpdateTable, CreatePartition, BatchCreatePartition, DeletePartition and BatchDeletePartition. For example, if a table or partition is updated, a notification is sent to CloudWatch Events. Note that you cannot write a program that depends on the order or existence of notification events, as they might be out of sequence or missing. Events are emitted on a best effort basis. In the details of the notification:
 - The typeOfChange contains the name of the API operation.
 - The databaseName contains the name of the database that contains the affected resources.
 - The tableName contains the name of the affected table.
 - The changedPartitions specifies up to 100 affected partitions in one notification. When partition names are long, multiple notifications might be created.

For example if there are two partition keys, Year and Month, then "2018,01", "2018,02" modifies the partition where "Year=2018" and "Month=01" and the partition where "Year=2018" and "Month=02".

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Glue Data Catalog Table State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
  "region": "us-west-2",
  "resources": ["arn:aws:glue:us-west-2:123456789012:database/default/foo"],
  "detail": {
    "changedPartitions": [
      "2018,01",
      "2018,02"
    ],
    "databaseName": "default",
    "tableName": "foo",
    "typeOfChange": "BatchCreatePartition"
  }
}
```

For more information, see the [Amazon CloudWatch Events User Guide](#). For events specific to AWS Glue, see [AWS Glue Events](#).

Monitoring Jobs Using the Apache Spark Web UI

You can use the Apache Spark web UI to monitor and debug AWS Glue ETL jobs running on the AWS Glue job system, and also Spark applications running on AWS Glue development endpoints. The Spark UI enables you to check the following for each job:

- The event timeline of each Spark stage
- A directed acyclic graph (DAG) of the job
- Physical and logical plans for SparkSQL queries
- The underlying Spark environmental variables for each job

You can enable the Spark UI using the AWS Glue console or the AWS Command Line Interface (AWS CLI). When you enable the Spark UI, AWS Glue ETL jobs and Spark applications on AWS Glue development endpoints can persist Spark event logs to a location that you specify in Amazon Simple Storage Service (Amazon S3). AWS Glue also provides a sample AWS CloudFormation template to start the Spark history server and show the Spark UI using the event logs. The persisted event logs in Amazon S3 can be used with the Spark UI both in real time as the job is executing and after the job is complete.

The following is an example of a Spark application which reads from two data sources, performs a join transform, and writes it out to Amazon S3 in Parquet format.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import count, when, expr, col, sum, isnull
from pyspark.sql.functions import countDistinct
from awsglue.dynamicframe import DynamicFrame

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'])

df_persons = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
persons.json")
df_memberships = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
memberships.json")

df_joined = df_persons.join(df_memberships, df_persons.id == df_memberships.person_id,
    'fullouter')
df_joined.write.parquet("s3://aws-glue-demo-sparkui/output/")

job.commit()
```

The following DAG visualization shows the different stages in this Spark job.

APACHE  2.2.1

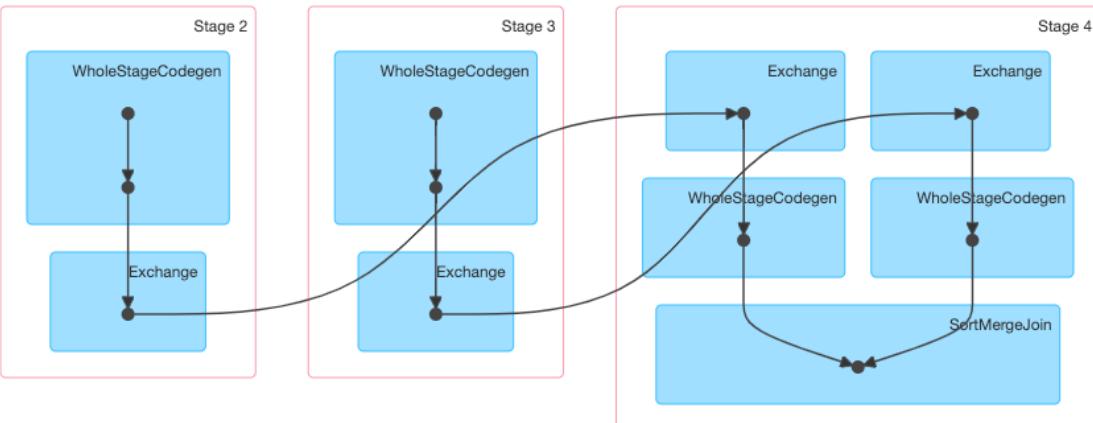
tape-sparksq... application UI

Jobs Stages Storage Environment Executors SQL

Details for Job 2

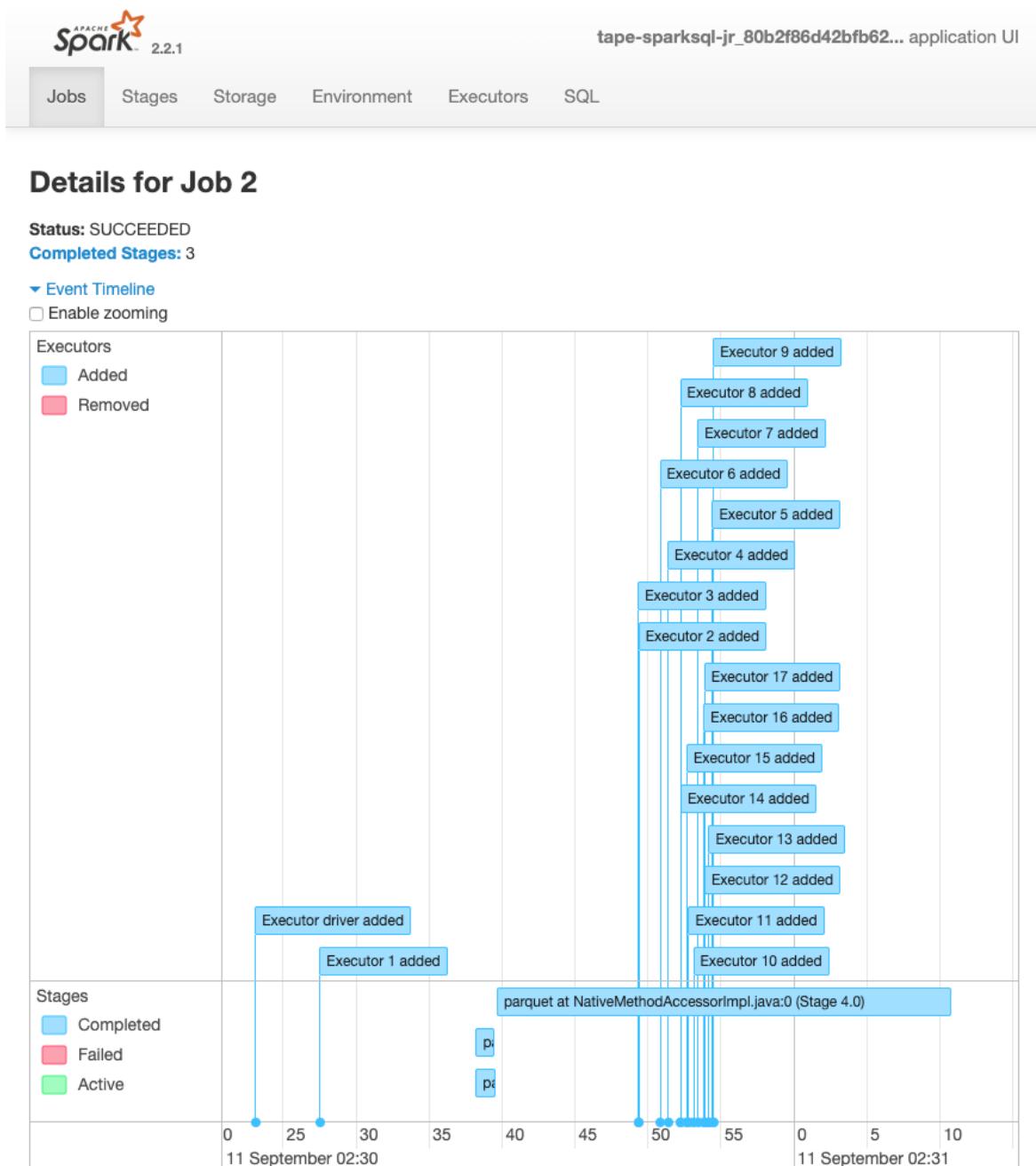
Status: SUCCEEDED
Completed Stages: 3

▶ Event Timeline
▼ DAG Visualization



▶ Completed Stages (3)

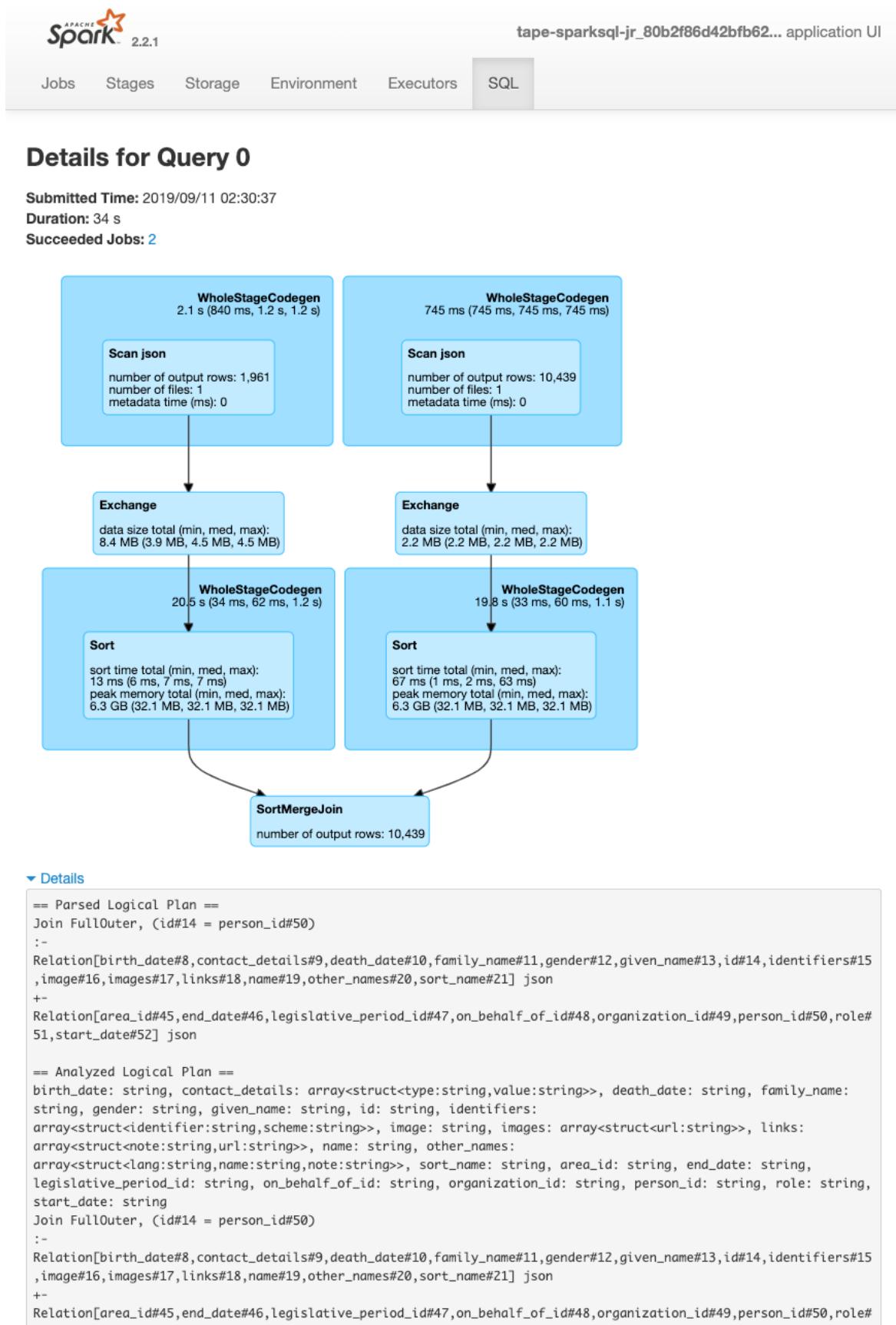
The following event timeline for a job shows the start, execution, and termination of different Spark executors.



- ▶ DAG Visualization
- ▶ Completed Stages (3)

The following screen shows the details of the SparkSQL query plans:

- Parsed logical plan
- Analyzed logical plan
- Optimized logical plan
- Physical plan for execution



You can still use AWS Glue continuous logging to view the Spark application log streams for Spark driver and executors. For more information, see [Continuous Logging for AWS Glue Jobs](#).

Topics

- [Enabling the Apache Spark Web UI for AWS Glue Jobs \(p. 237\)](#)
- [Enabling the Apache Spark Web UI for Development Endpoints \(p. 238\)](#)
- [Launching the Spark History Server \(p. 239\)](#)

Enabling the Apache Spark Web UI for AWS Glue Jobs

You can use the Apache Spark web UI to monitor and debug AWS Glue ETL jobs running on the AWS Glue job system. You can configure the Spark UI using the AWS Glue console or the AWS Command Line Interface (AWS CLI).

Topics

- [Configuring the Spark UI \(Console\) \(p. 237\)](#)
- [Configuring the Spark UI \(AWS CLI\) \(p. 238\)](#)

Configuring the Spark UI (Console)

Follow these steps to configure the Spark UI using the AWS Management Console.

To create a job with the Spark UI enabled

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Choose **Add job**.
4. In **Configure the job properties**, open the **Monitoring options**.
5. In the **Spark UI** tab, choose **Enable**.
6. Specify an Amazon S3 path for storing the Spark event logs for the job.

To edit an existing job to enable the Spark UI

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Choose an existing job in the job list.
4. Choose **Action**, and then choose **Edit job**.
5. Open the **Monitoring options**.
6. In the **Spark UI** tab, choose **Enable**.
7. Enter an Amazon S3 path for storing the Spark event logs for the job.

To set up user preferences for new jobs to enable the Spark UI

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the upper-right corner, choose **User preferences**.
3. Open the **Monitoring options**.

4. In the **Spark UI** tab, choose **Enable**.
5. Specify an Amazon S3 path for storing the Spark event logs for the job.

To set up the job run options to enable the Spark UI

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Choose an existing job in the job lists.
4. Choose **Scripts** and **Edit Job**. You navigate to the code pane.
5. Choose **Run job**.
6. Open the **Monitoring options**.
7. In the **Spark UI** tab, choose **Enable**.
8. Specify an Amazon S3 path for storing the Spark event logs for the job.

Configuring the Spark UI (AWS CLI)

To enable the Spark UI feature using the AWS CLI, pass in the following job parameters to AWS Glue jobs. For more information, see [Special Parameters Used by AWS Glue](#).

```
'--enable-spark-ui': 'true',
'--spark-event-logs-path': 's3://s3-event-log-path'
```

Every 30 seconds, AWS Glue flushes the Spark event logs to the Amazon S3 path that you specify.

Enabling the Apache Spark Web UI for Development Endpoints

Use the Apache Spark web UI to monitor and debug Spark applications running on AWS Glue development endpoints. You can configure the development endpoints with the Spark Web UI using the AWS Glue console or the AWS Command Line Interface (AWS CLI).

Topics

- [Enabling the Spark UI \(Console\) \(p. 238\)](#)
- [Enabling the Spark UI \(AWS CLI\) \(p. 238\)](#)

Enabling the Spark UI (Console)

Follow these steps to configure the Spark UI using the AWS Management Console.

To create a development endpoint with the Spark UI enabled

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Dev endpoints**.
3. Choose **Add endpoint**.
4. In **Configuration**, open the **Spark UI options**.
5. In the **Spark UI** tab, choose **Enable**.

- Specify an Amazon S3 path for storing the Spark event logs.

Enabling the Spark UI (AWS CLI)

To create a development endpoint with the Spark UI enabled using the AWS CLI, pass in the following arguments in JSON syntax.

```
{  
    "EndpointName": "Name",  
    "RoleArn": "role_ARN",  
    "PublicKey": "public_key_contents",  
    "NumberOfNodes": 2,  
    "Arguments": {  
        "--enable-spark-ui": "true",  
        "--spark-event-logs-path": "s3://s3-event-log-path"  
    }  
}
```

Launching the Spark History Server

You can launch the Spark history server using a AWS CloudFormation template that hosts the server on an EC2 instance, or launch locally using Docker.

Topics

- [Launching the Spark History Server and Viewing the Spark UI Using AWS CloudFormation \(p. 239\)](#)
- [Launching the Spark History Server and Viewing the Spark UI Using Docker \(p. 241\)](#)

Launching the Spark History Server and Viewing the Spark UI Using AWS CloudFormation

You can use an AWS CloudFormation template to start the Apache Spark history server and view the Spark web UI. These templates are samples that you should modify to meet your requirements.

To start the Spark history server and view the Spark UI using AWS CloudFormation

- Choose one of the **Launch Stack** buttons in the following table. This launches the stack on the AWS CloudFormation console.

Region	View	Launch
US East (Ohio)	View	Launch Stack
US East (N. Virginia)	View	Launch Stack
US West (N. California)	View	Launch Stack
US West (Oregon)	View	Launch Stack
Asia Pacific (Hong Kong)	View	Launch Stack

Region	View	Launch
Asia Pacific (Mumbai)	View	Launch Stack 
Asia Pacific (Seoul)	View	Launch Stack 
Asia Pacific (Singapore)	View	Launch Stack 
Asia Pacific (Sydney)	View	Launch Stack 
Asia Pacific (Tokyo)	View	Launch Stack 
Canada (Central)	View	Launch Stack 
Europe (Frankfurt)	View	Launch Stack 
Europe (Ireland)	View	Launch Stack 
Europe (London)	View	Launch Stack 
Europe (Paris)	View	Launch Stack 
Europe (Stockholm)	View	Launch Stack 
South America (São Paulo)	View	Launch Stack 

2. On the **Specify template** page, choose **Next**.
3. On the **Specify stack details** page, enter the **Stack name**. Choose **Parameters**, and then choose **Next**.

a. **Spark UI Configuration**

Provide the following information:

- **IP address range** — The IP address range that can be used to view the Spark UI. If you want to restrict access from a specific IP address range, you should use a custom value.
- **History server port** — The port for the Spark UI. You can use the default value.
- **Event log directory** — Choose the location where Spark event logs are stored from the AWS Glue job or development endpoints. You must use `s3a://` for the event logs path scheme.
- **Spark package location** — You can use the default value.
- **Keystore path** — SSL/TLS keystore path for HTTPS. If you want to use a custom keystore file, you can specify the S3 path `s3://path_to_your_keystore_file` here. If you leave this parameter empty, a self-signed certificate based keystore is generated and used.

Note

With a self-signed certificate based keystore, each local machine that connects to the Spark UI must be configured to trust the certificate generated before connecting to the Spark UI. Also, when the generated certificate expires, a new certificate must be generated and trusted on all local machines. For more information about the

setup, see [Self-signed certificates](#). For more information, see [Self-signed certificate in Wikipedia](#).

- **Keystore password** — SSL/TLS keystore password for HTTPS.
- b. **EC2 Instance Configuration**

Provide the following information:

- **Instance type** — The type of Amazon EC2 instance that hosts the Spark history server. Because this template launches Amazon EC2 instance in your account, Amazon EC2 cost will be charged in your account separately.
 - **Latest AMI ID** — The AMI ID of Amazon Linux 2 for the Spark history server instance. You can use the default value.
 - **VPC ID** — The virtual private cloud (VPC) ID for the Spark history server instance. You can use any of the VPCs available in your account. Using a default VPC with a [default Network ACL](#) is not recommended. For more information, see [Default VPC and Default Subnets](#) and [Creating a VPC in the Amazon VPC User Guide](#).
 - **Subnet ID** — The ID for the Spark history server instance. You can use any of the subnets in your VPC. You must be able to reach the network from your client to the subnet. If you want to access via the internet, you must use a public subnet that has the internet gateway in the route table.
4. On the **Configure stack options** page, choose **Next**.
 5. On the **Review** page, review the template. Select **I acknowledge that AWS CloudFormation might create IAM resources**, and then choose **Create stack**.
 6. Wait for the stack to be created.
 7. Open the **Outputs** tab.
 - a. Copy the URL of **SparkUiPublicUrl** if you are using a public subnet.
 - b. Copy the URL of **SparkUiPrivateUrl** if you are using a private subnet.
 8. Open a web browser, and paste in the URL. This lets you access the server using HTTPS on the specified port. Your browser may not recognize the server's certificate, in which case you have to override its protection and proceed anyway.

Launching the Spark History Server and Viewing the Spark UI Using Docker

If you prefer local access (not to have an EC2 instance for the Apache Spark history server), you can also use Docker to start the Apache Spark history server and view the Spark UI locally. This Dockerfile is a sample that you should modify to meet your requirements.

Prerequisites

For information about how to install Docker on your laptop see the [Docker Engine community](#).

To start the Spark history server and view the Spark UI locally using Docker

1. Download files from GitHub.
 - Download the Dockerfile and pom.xml from [AWS Glue code samples](#).
2. Run the following commands:
 - a. Replace the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` with your valid AWS credentials.
 - b. Replace the `s3a://path_to_eventlog` with your event log directory.

```
$ docker build -t glue/sparkui:latest .
$ docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS
-Dspark.history.fs.logDirectory=s3a:/path_to_eventlog
-Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY" -p
18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
org.apache.spark.deploy.history.HistoryServer"
```

3. Open <http://localhost:18080> in your browser to view the Spark UI locally.

Monitoring with Amazon CloudWatch

You can monitor AWS Glue using Amazon CloudWatch, which collects and processes raw data from AWS Glue into readable, near-real-time metrics. These statistics are recorded for a period of two weeks so that you can access historical information for a better perspective on how your web application or service is performing. By default, AWS Glue metrics data is sent to CloudWatch automatically. For more information, see [What Is Amazon CloudWatch?](#) in the *Amazon CloudWatch User Guide*, and [AWS Glue Metrics \(p. 243\)](#).

AWS Glue also supports real-time continuous logging for AWS Glue jobs. When continuous logging is enabled for a job, you can view the real-time logs on the AWS Glue console or the CloudWatch console dashboard. For more information, see [Continuous Logging for AWS Glue Jobs \(p. 256\)](#).

Topics

- [Monitoring AWS Glue Using Amazon CloudWatch Metrics \(p. 242\)](#)
- [Setting Up Amazon CloudWatch Alarms on AWS Glue Job Profiles \(p. 256\)](#)
- [Continuous Logging for AWS Glue Jobs \(p. 256\)](#)

Monitoring AWS Glue Using Amazon CloudWatch Metrics

You can profile and monitor AWS Glue operations using AWS Glue job profiler. It collects and processes raw data from AWS Glue jobs into readable, near real-time metrics stored in Amazon CloudWatch. These statistics are retained and aggregated in CloudWatch so that you can access historical information for a better perspective on how your application is performing.

AWS Glue Metrics Overview

When you interact with AWS Glue, it sends metrics to CloudWatch. You can view these metrics using the AWS Glue console (the preferred method), the CloudWatch console dashboard, or the AWS Command Line Interface (AWS CLI).

To view metrics using the AWS Glue console dashboard

You can view summary or detailed graphs of metrics for a job, or detailed graphs for a job run. For details about the graphs and metrics you can access in the AWS Glue console dashboard, see [Working with Jobs on the AWS Glue Console \(p. 174\)](#).

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Select a job from the **Jobs** list.

4. Choose the **Metrics** tab.
5. Choose **View additional metrics** to see more detailed metrics.

To view metrics using the CloudWatch console dashboard

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **Glue** namespace.

To view metrics using the AWS CLI

- At a command prompt, use the following command.

```
aws cloudwatch list-metrics --namespace "Glue"
```

AWS Glue reports metrics to CloudWatch every 30 seconds, and the CloudWatch metrics dashboards are configured to display them every minute. The AWS Glue metrics represent delta values from the previously reported values. Where appropriate, metrics dashboards aggregate (sum) the 30-second values to obtain a value for the entire last minute. AWS Glue metrics are enabled at initialization of a `GlueContext` in a script and are generally updated only at the end of an Apache Spark task. They represent the aggregate values across all completed Spark tasks so far.

However, the Spark metrics that AWS Glue passes on to CloudWatch are generally absolute values representing the current state at the time they are reported. AWS Glue reports them to CloudWatch every 30 seconds, and the metrics dashboards generally show the average across the data points received in the last 1 minute.

AWS Glue metrics names are all preceded by one of the following types of prefix:

- `glue.driver.` – Metrics whose names begin with this prefix either represent AWS Glue metrics that are aggregated from all executors at the Spark driver, or Spark metrics corresponding to the Spark driver.
- `glue.executorId.` – The `executorId` is the number of a specific Spark executor. It corresponds with the executors listed in the logs.
- `glue.ALL.` – Metrics whose names begin with this prefix aggregate values from all Spark executors.

AWS Glue Metrics

AWS Glue profiles and sends the following metrics to CloudWatch every 30 seconds, and the AWS Glue Metrics Dashboard report them once a minute:

Metric	Description
<code>glue.driver.aggregate.bytesRead</code>	The number of bytes read from all data sources by all completed Spark tasks running in all executors.. Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID. or <code>ALL</code>), and <code>Type</code> (count).

Metric	Description
	<p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Bytes</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Bytes read. • Job progress. • JDBC data sources. • Job Bookmark Issues. • Variance across Job Runs. <p>This metric can be used the same way as the <code>glue.ALL.s3.filesystem.read_bytes</code> metric, with the difference that this metric is updated at the end of a Spark task and captures non-S3 data sources as well.</p>
<code>glue.driver.aggregate.elapsedTime</code>	<p>The ETL elapsed time in milliseconds (does not include the job bootstrap times).</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Milliseconds</p> <p>Can be used to determine how long it takes a job run to run on average.</p> <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Set alarms for stragglers. • Measure variance across job runs.

Metric	Description
<code>glue.driver.aggregate.numCompletedStages</code>	<p>The number of completed stages in the job.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Job progress. • Per-stage timeline of job execution, when correlated with other metrics. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Identify demanding stages in the execution of a job. • Set alarms for correlated spikes (demanding stages) across job runs.
<code>glue.driver.aggregate.numCompletedTasks</code>	<p>The number of completed tasks in the job.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Job progress. • Parallelism within a stage.

Metric	Description
<code>glue.driver.aggregate.numFailedTasks</code>	<p>The number of failed tasks.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Data abnormalities that cause job tasks to fail. • Cluster abnormalities that cause job tasks to fail. • Script abnormalities that cause job tasks to fail. <p>The data can be used to set alarms for increased failures that might suggest abnormalities in data, cluster or scripts.</p>
<code>glue.driver.aggregate.numKilledTasks</code>	<p>The number of tasks killed.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Abnormalities in Data Skew that result in exceptions (OOMs) that kill tasks. • Script abnormalities that result in exceptions (OOMs) that kill tasks. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Set alarms for increased failures indicating data abnormalities. • Set alarms for increased failures indicating cluster abnormalities. • Set alarms for increased failures indicating script abnormalities.

Metric	Description
<code>glue.driver.aggregate.recordsRead</code>	<p>The number of records read from all data sources by all completed Spark tasks running in all executors.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> Records read. Job progress. JDBC data sources. Job Bookmark Issues. Skew in Job Runs over days. <p>This metric can be used in a similar way to the <code>glue.ALL.s3.filesystem.read_bytes</code> metric, with the difference that this metric is updated at the end of a Spark task.</p>
<code>glue.driver.aggregate.shuffleBytesWritten</code>	<p>The number of bytes written by all executors to shuffle data between them since the previous report (aggregated by the AWS Glue Metrics Dashboard as the number of bytes written for this purpose during the previous minute).</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Bytes</p> <p>Can be used to monitor: Data shuffle in jobs (large joins, <code>groupBy</code>, <code>repartition</code>, <code>coalesce</code>).</p> <p>Some ways to use the data:</p> <ul style="list-style-type: none"> Repartition or decompress large input files before further processing. Repartition data more uniformly to avoid hot keys. Pre-filter data before joins or <code>groupBy</code> operations.

Metric	Description
<code>glue.driver.aggregate.shuffleLocalBytesRead</code>	<p>The number of bytes read by all executors to shuffle data between them since the previous report (aggregated by the AWS Glue Metrics Dashboard as the number of bytes read for this purpose during the previous minute).</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID. or <code>ALL</code>), and <code>Type</code> (count).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard, a SUM statistic is used for aggregation.</p> <p>Unit: Bytes</p> <p>Can be used to monitor: Data shuffle in jobs (large joins, groupBy, repartition, coalesce).</p> <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Repartition or decompress large input files before further processing. • Repartition data more uniformly using hot keys. • Pre-filter data before joins or groupBy operations.
<code>glue.driver.BlockManager.disk.diskSpaceUsed</code>	<p>The <code>diskSpaceUsed</code> megabytes of disk space used across all executors.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID. or <code>ALL</code>), and <code>Type</code> (gauge).</p> <p>Valid Statistics: Average. This is a Spark metric, reported as an absolute value.</p> <p>Unit: Megabytes</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Disk space used for blocks that represent cached RDD partitions. • Disk space used for blocks that represent intermediate shuffle outputs. • Disk space used for blocks that represent broadcasts. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Identify job failures due to increased disk usage. • Identify large partitions resulting in spilling or shuffling. • Increase provisioned DPU capacity to correct these issues.

Metric	Description
<code>glue.driver.ExecutorAllocationManager.validDimensionsCountName</code> (<code>gauge</code>)	<p>The number of actively running job executors.</p> <p>Valid Statistics: Average. This is a Spark metric, reported as an absolute value.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Job activity. • Straggling executors (with a few executors running only) • Current executor-level parallelism. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Repartition or decompress large input files beforehand if cluster is under-utilized. • Identify stage or job execution delays due to straggler scenarios. • Compare with <code>numberMaxNeededExecutors</code> to understand backlog for provisioning more DPUs.

Metric	Description
<code>glue.driver.ExecutorAllocationManager.leadExecutors.numberMaxNeededExecutors</code>	<p>The number of maximum (actively running and pending) job executors needed to satisfy the current lead.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (gauge).</p> <p>Valid Statistics: Maximum. This is a Spark metric, reported as an absolute value.</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Job activity. • Current executor-level parallelism and backlog of pending tasks not yet scheduled because of unavailable executors due to DPU capacity or killed/failed executors. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • Identify pending/backlog of scheduling queue. • Identify stage or job execution delays due to straggler scenarios. • Compare with <code>numberAllExecutors</code> to understand backlog for provisioning more DPUs. • Increase provisioned DPU capacity to correct the pending executor backlog.

Metric	Description
<pre>glue.driver.jvm.heap.usage</pre> <pre>glue.executorId.jvm.heap.usage</pre> <pre>glue.ALL.jvm.heap.usage</pre>	<p>The fraction of memory used by the JVM heap for this driver (scale: 0-1) for driver, executor identified by executorId, or ALL executors.</p> <p>Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (gauge).</p> <p>Valid Statistics: Average. This is a Spark metric, reported as an absolute value.</p> <p>Unit: Percentage</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> Driver out-of-memory conditions (OOM) using <code>glue.driver.jvm.heap.usage</code>. Executor out-of-memory conditions (OOM) using <code>glue.ALL.jvm.heap.usage</code>. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> Identify memory-consuming executor ids and stages. Identify straggling executor ids and stages. Identify a driver out-of-memory condition (OOM). Identify an executor out-of-memory condition (OOM) and obtain the corresponding executor ID so as to be able to get a stack trace from the executor log. Identify files or partitions that may have data skew resulting in stragglers or out-of-memory conditions (OOMs).

Metric	Description
<code>glue.driver.jvm.heap.used</code>	The number of memory bytes used by the JVM heap for the driver, the executor identified by <i>executorId</i> , or ALL executors.
<code>glue.executorId.jvm.heap.used</code>	Valid dimensions: <code>JobName</code> (the name of the AWS Glue Job), <code>JobRunId</code> (the JobRun ID, or <code>ALL</code>), and <code>Type</code> (gauge).
<code>glue.ALL.jvm.heap.used</code>	<p>Valid Statistics: Average. This is a Spark metric, reported as an absolute value.</p> <p>Unit: Bytes</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> Driver out-of-memory conditions (OOM). Executor out-of-memory conditions (OOM). <p>Some ways to use the data:</p> <ul style="list-style-type: none"> Identify memory-consuming executor ids and stages. Identify straggling executor ids and stages. Identify a driver out-of-memory condition (OOM). Identify an executor out-of-memory condition (OOM) and obtain the corresponding executor ID so as to be able to get a stack trace from the executor log. Identify files or partitions that may have data skew resulting in stragglers or out-of-memory conditions (OOMs).

Metric	Description
<pre>glue.driver.s3.filesystem.read_bytes glue.executorId.s3.filesystem.read_bytes glue.ALL.s3.filesystem.read_bytes</pre>	<p>The number of bytes read from Amazon S3 by the driver, an executor identified by <code>executorId</code>, or ALL executors since the previous report (aggregated by the AWS Glue Metrics Dashboard as the number of bytes read during the previous minute).</p> <p>Valid dimensions: <code>JobName</code>, <code>JobRunId</code>, and <code>Type</code> (gauge).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard a SUM statistic is used for aggregation. The area under the curve on the AWS Glue Metrics Dashboard can be used to visually compare bytes read by two different job runs.</p> <p>Unit: Bytes.</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • ETL data movement. • Job progress. • Job bookmark issues (data processed, reprocessed, and skipped). • Comparison of reads to ingestion rate from external data sources. • Variance across job runs. <p>Resulting data can be used for:</p> <ul style="list-style-type: none"> • DPU capacity planning. • Setting alarms for large spikes or dips in data read for job runs and job stages.

Metric	Description
<pre>glue.driver.s3.filesystem.write_bytes glue.executorId.s3.filesystem.write_bytes glue.ALL.s3.filesystem.write_bytes</pre>	<p>The number of bytes written to Amazon S3 by the driver, an executor identified by <code>executorId</code>, or ALL executors since the previous report (aggregated by the AWS Glue Metrics Dashboard as the number of bytes written during the previous minute).</p> <p>Valid dimensions: <code>JobName</code>, <code>JobRunId</code>, and <code>Type</code> (gauge).</p> <p>Valid Statistics: SUM. This metric is a delta value from the last reported value, so on the AWS Glue Metrics Dashboard a SUM statistic is used for aggregation. The area under the curve on the AWS Glue Metrics Dashboard can be used to visually compare bytes written by two different job runs.</p> <p>Unit: Bytes</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • ETL data movement. • Job progress. • Job bookmark issues (data processed, reprocessed, and skipped). • Comparison of reads to ingestion rate from external data sources. • Variance across job runs. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • DPU capacity planning. • Setting alarms for large spikes or dips in data read for job runs and job stages.

Metric	Description
<pre>glue.driver.system.cpuSystemLoad</pre> <pre>glue.executorId.system.cpuSystemLoad</pre> <pre>glue.ALL.system.cpuSystemLoad</pre>	<p>The fraction of CPU system load used (scale: 0-1) by the driver, an executor identified by <i>executorId</i>, or ALL executors.</p> <p>Valid dimensions: <i>JobName</i> (the name of the AWS Glue Job), <i>JobRunId</i> (the JobRun ID, or ALL), and <i>Type</i> (gauge).</p> <p>Valid Statistics: Average. This metric is reported as an absolute value.</p> <p>Unit: Percentage</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Driver CPU load. • Executor CPU load. • Detecting CPU-bound or IO-bound executors or stages in a Job. <p>Some ways to use the data:</p> <ul style="list-style-type: none"> • DPU capacity Planning along with IO Metrics (Bytes Read/Shuffle Bytes, Task Parallelism) and the number of maximum needed executors metric. • Identify the CPU/IO-bound ratio. This allows for repartitioning and increasing provisioned capacity for long-running jobs with splittable datasets having lower CPU utilization.

Dimensions for AWS Glue Metrics

AWS Glue metrics use the AWS Glue namespace and provide metrics for the following dimensions:

Dimension	Description
<i>JobName</i>	This dimension filters for metrics of all job runs of a specific AWS Glue job.
<i>JobRunId</i>	This dimension filters for metrics of a specific AWS Glue job run by a JobRun ID, or ALL.
<i>Type</i>	This dimension filters for metrics by either count (an aggregate number) or gauge (a value at a point in time).

For more information, see the [Amazon CloudWatch User Guide](#).

Setting Up Amazon CloudWatch Alarms on AWS Glue Job Profiles

AWS Glue metrics are also available in Amazon CloudWatch. You can set up alarms on any AWS Glue metric for scheduled jobs.

A few common scenarios for setting up alarms are as follows:

- Jobs running out of memory (OOM): Set an alarm when the memory usage exceeds the normal average for either the driver or an executor for an AWS Glue job.
- Straggling executors: Set an alarm when the number of executors falls below a certain threshold for a large duration of time in an AWS Glue job.
- Data backlog or reprocessing: Compare the metrics from individual jobs in a workflow using a CloudWatch math expression. You can then trigger an alarm on the resulting expression value (such as the ratio of bytes written by a job and bytes read by a following job).

For detailed instructions on setting alarms, see [Create or Edit a CloudWatch Alarm](#) in the *Amazon CloudWatch Events User Guide*.

For monitoring and debugging scenarios using CloudWatch, see [Job Monitoring and Debugging](#) (p. 259).

Continuous Logging for AWS Glue Jobs

AWS Glue provides real-time, continuous logging for AWS Glue jobs. You can view real-time Apache Spark job logs in Amazon CloudWatch, including driver logs, executor logs, and an Apache Spark job progress bar. Viewing real-time logs provides you with a better perspective on the running job.

When you start an AWS Glue job, it sends the real-time logging information to CloudWatch (every 5 seconds and before each executor termination) after the Spark application starts running. You can view the logs on the AWS Glue console or the CloudWatch console dashboard.

The continuous logging feature includes the following capabilities:

- Continuous logging with a default filter to reduce high verbosity in the logs
- Continuous logging with no filter
- A custom script logger to log application-specific messages
- A console progress bar to track the running status of the current AWS Glue job

Topics

- [Enabling Continuous Logging for AWS Glue Jobs](#) (p. 256)
- [Viewing Continuous Logging for AWS Glue Jobs](#) (p. 259)

Enabling Continuous Logging for AWS Glue Jobs

You can enable continuous logging using the AWS Glue console or through the AWS Command Line Interface (AWS CLI).

You can enable continuous logging with either a standard filter or no filter when you create a new job, edit an existing job, or enable it through the AWS CLI. Choosing the **Standard filter** prunes out non-

useful Apache Spark driver/executor and Apache Hadoop YARN heartbeat log messages. Choosing **No filter** gives you all the log messages.

You can also specify custom configuration options such as the AWS CloudWatch log group name, CloudWatch log stream prefix before the AWS Glue job run ID driver/executor ID, and log conversion pattern for log messages. These configurations help you to set aggregate logs in custom CloudWatch log groups with different expiration policies, and analyze them further with custom log stream prefixes and conversions patterns.

Topics

- [Using the AWS Management Console \(p. 257\)](#)
- [Logging Application-Specific Messages Using the Custom Script Logger \(p. 258\)](#)
- [Enabling the Progress Bar to Show Job Progress \(p. 259\)](#)

Using the AWS Management Console

Follow these steps to use the console to enable continuous logging when creating or editing an AWS Glue job.

To create a new AWS Glue job with continuous logging

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Choose **Add job**.
4. In **Configure the job properties**, choose **Monitoring options**.
5. In the **Continuous logging** tab, choose **Enable**.
6. Choose **Standard filter** or **No filter**.

To enable continuous logging for an existing AWS Glue job

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Choose an existing job from the **Jobs** list.
4. Choose **Action, Edit job**.
5. Choose **Monitoring options**.
6. In the **Continuous logging** tab, choose **Enable**.
7. Choose **Standard filter** or **No filter**.

To enable continuous logging for all newly created AWS Glue jobs

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. In the upper-right corner, choose **User preferences**.
4. Choose **Monitoring options**.
5. In the **Continuous logging** tab, choose **Enable**.
6. Choose **Standard filter** or **No filter**.

These user preferences are applied to all new jobs unless you override them explicitly when creating an AWS Glue job or by editing an existing job as described previously.

Using the AWS CLI

To enable continuous logging, you pass in job parameters to an AWS Glue job. When you want to use the standard filter, pass the following special job parameters similar to other AWS Glue job parameters. For more information, see [Special Parameters Used by AWS Glue \(p. 292\)](#).

```
--enable-continuous-cloudwatch-log': 'true'
```

When you want no filter, use the following.

```
--enable-continuous-cloudwatch-log': 'true',
--enable-continuous-log-filter': 'false'
```

You can specify a custom AWS CloudWatch log group name. If not specified, the default log group name is /aws-glue/jobs/logs-v2/.

```
--continuous-log-logGroup': 'custom_log_group_name'
```

You can specify a custom AWS CloudWatch log stream prefix. If not specified, the default log stream prefix is the job run ID.

```
--continuous-log-logStreamPrefix': 'custom_log_stream_prefix'
```

You can specify a custom continuous logging conversion pattern. If not specified, the default conversion pattern is %d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n. Note that the conversion pattern only applies to driver logs and executor logs. It does not affect the Glue progress bar.

```
--continuous-log-conversionPattern': 'custom_log_conversion_pattern'
```

Logging Application-Specific Messages Using the Custom Script Logger

You can use the AWS Glue logger to log any application-specific messages in the script that are sent in real time to the driver log stream.

The following example shows a Python script.

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

The following example shows a Scala script.

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
    def main(sysArgs: Array[String]) {
        val logger = new GlueLogger
        logger.info("info message")
        logger.warn("warn message")
```

```
    logger.error("error message")
}
```

Enabling the Progress Bar to Show Job Progress

AWS Glue provides a real-time progress bar under the `JOB_RUN_ID-progress-bar` log stream to check AWS Glue job run status. Currently it supports only jobs that initialize `glueContext`. If you run a pure Spark job without initializing `glueContext`, the AWS Glue progress bar does not appear.

The progress bar shows the following progress update every 5 seconds.

```
Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /  
totalNumberOfTasksInThisStage]
```

Viewing Continuous Logging for AWS Glue Jobs

You can view real-time logs using the AWS Glue console or the Amazon CloudWatch console.

To view real-time logs using the AWS Glue console dashboard

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Jobs**.
3. Add or start an existing job. Choose **Action**, **Run job**.

When you start running a job, you navigate to a page that contains information about the running job:

- The **Logs** tab shows the older aggregated application logs.
 - The **Continuous logging** tab shows a real-time progress bar when the job is running with `glueContext` initialized.
 - The **Continuous logging** tab also contains the **Driver logs**, which capture real-time Apache Spark driver logs, and application logs from the script logged using the AWS Glue application logger when the job is running.
4. For older jobs, you can also view the real-time logs under the **Job History** view by choosing **Logs**. This action takes you to the CloudWatch console that shows all Spark driver, executor, and progress bar log streams for that job run.

To view real-time logs using the CloudWatch console dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log**.
3. Choose the `/aws-glue/jobs/logs-v2/` log group.
4. In the **Filter** box, paste the job run ID.

You can view the driver logs, executor logs, and progress bar (if using the **Standard filter**).

Job Monitoring and Debugging

You can collect metrics about AWS Glue jobs and visualize them on the AWS Glue and Amazon CloudWatch consoles to identify and fix issues. Profiling your AWS Glue jobs requires the following steps:

1. Enable the **Job metrics** option in the job definition. You can enable profiling in the AWS Glue console or as a parameter to the job. For more information see [Defining Job Properties \(p. 165\)](#) or [Special Parameters Used by AWS Glue \(p. 292\)](#).
2. Confirm that the job script initializes a `GlueContext`. For example, the following script snippet initializes a `GlueContext` and shows where profiled code is placed in the script. This general format is used in the debugging scenarios that follow.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

...
...
code-to-profile
...
...

job.commit()
```

3. Run the job.
4. Visualize the metrics on the AWS Glue console and identify abnormal metrics for the `driver` or an `executor`.
5. Narrow down the root cause using the identified metric.
6. Optionally, confirm the root cause using the log stream of the identified driver or job executor.

Topics

- [Debugging OOM Exceptions and Job Abnormalities \(p. 260\)](#)
- [Debugging Demanding Stages and Straggler Tasks \(p. 267\)](#)
- [Monitoring the Progress of Multiple Jobs \(p. 271\)](#)
- [Monitoring for DPU Capacity Planning \(p. 275\)](#)

Debugging OOM Exceptions and Job Abnormalities

You can debug out-of-memory (OOM) exceptions and job abnormalities in AWS Glue. The following sections describe scenarios for debugging out-of-memory exceptions of the Apache Spark driver or a Spark executor.

- [Debugging a Driver OOM Exception \(p. 261\)](#)
- [Debugging an Executor OOM Exception \(p. 263\)](#)

Debugging a Driver OOM Exception

In this scenario, a Spark job is reading a large number of small files from Amazon Simple Storage Service (Amazon S3). It converts the files to Apache Parquet format and then writes them out to Amazon S3. The Spark driver is running out of memory. The input Amazon S3 data has more than 1 million files in different Amazon S3 partitions.

The profiled code is as follows:

```
data = spark.read.format("json").option("inferSchema", False).load("s3://input_path")
data.write.format("parquet").save(output_path)
```

Visualize the Profiled Metrics on the AWS Glue Console

The following graph shows the memory usage as a percentage for the driver and executors. This usage is plotted as one data point that is averaged over the values reported in the last minute. You can see in the memory profile of the job that the [driver memory \(p. 251\)](#) crosses the safe threshold of 50 percent usage quickly. On the other hand, the [average memory usage \(p. 251\)](#) across all executors is still less than 4 percent. This clearly shows abnormality with driver execution in this Spark job.



The job run soon fails, and the following error appears in the [History](#) tab on the AWS Glue console: Command Failed with Exit Code 1. This error string means that the job failed due to a systemic error—which in this case is the driver running out of memory.

e2e-metrics		python		s3://aws-glue-scripts-6569...		7 June 2018 7:37 PM UTC-7		Disable		
History	Details	Script	Metrics							
Show										
Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time
jr_651bfc34...	-	Failed	! ...	Logs	Error logs	2 mins	2880 mins			7 June 18
jr_5731b225...	-	Failed	Command failed with exit code 1				1 mins	2880 mins		7 June 18

On the console, choose the **Error logs** link on the **History** tab to confirm the finding about driver OOM from the CloudWatch Logs. Search for "**Error**" in the job's error logs to confirm that it was indeed an OOM exception that failed the job:

```
# java.lang.OutOfMemoryError: Java heap space
# -XX:OnOutOfMemoryError="kill -9 %p"
# Executing /bin/sh -c "kill -9 12039"...
```

On the **History** tab for the job, choose **Logs**. You can find the following trace of driver execution in the CloudWatch Logs at the beginning of the job. The Spark driver tries to list all the files in all the directories, constructs an `InMemoryFileIndex`, and launches one task per file. This in turn results in the Spark driver having to maintain a large amount of state in memory to track all the tasks. It caches the complete list of a large number of files for the in-memory index, resulting in a driver OOM.

Fix the Processing of Multiple Files Using Grouping

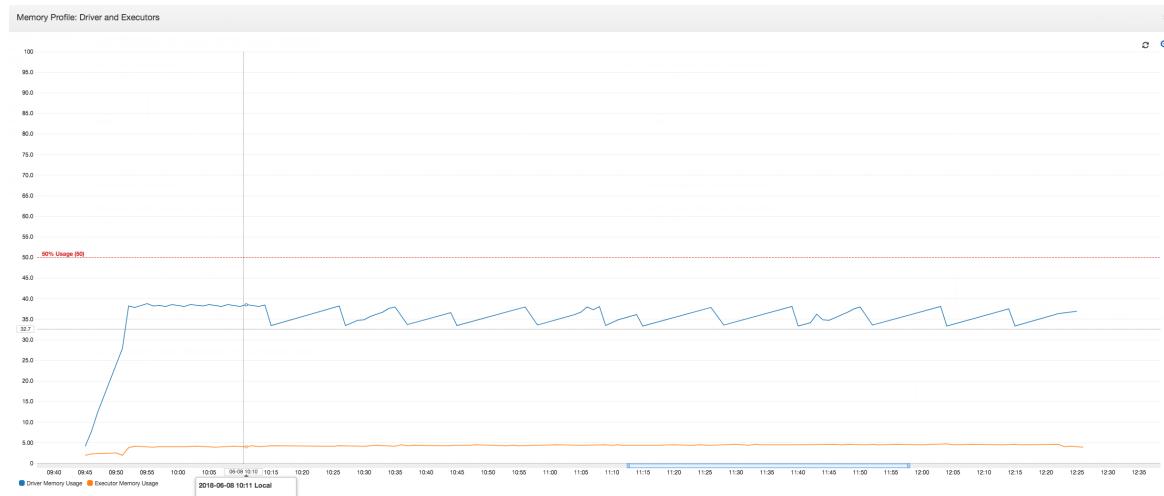
You can fix the processing of the multiple files by using the *grouping* feature in AWS Glue. Grouping is automatically enabled when you use dynamic frames and when the input dataset has a large number of files (more than 50,000). Grouping allows you to coalesce multiple files together into a group, and it allows a task to process the entire group instead of a single file. As a result, the Spark driver stores significantly less state in memory to track fewer tasks. For more information about manually enabling grouping for your dataset, see [Reading Input Files in Larger Groups \(p. 303\)](#).

To check the memory profile of the AWS Glue job, profile the following code with grouping enabled:

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
    "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3",
    connection_options = {"path": output_path}, format = "parquet", transformation_ctx =
    "datasink")
```

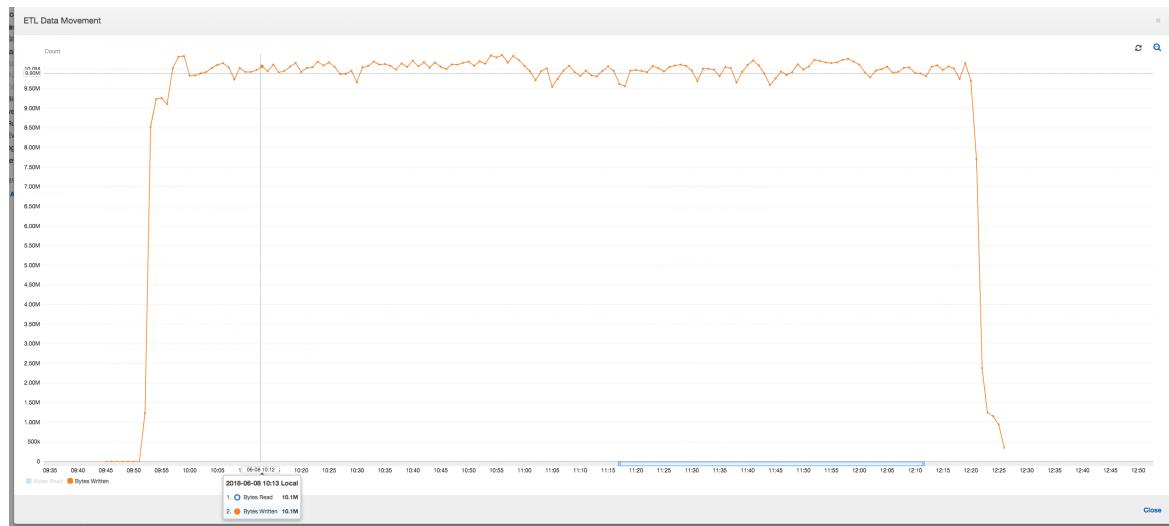
You can monitor the memory profile and the ETL data movement in the AWS Glue job profile.

The driver executes below the threshold of 50 percent memory usage over the entire duration of the AWS Glue job. The executors stream the data from Amazon S3, process it, and write it out to Amazon S3. As a result, they consume less than 5 percent memory at any point in time.



The data movement profile below shows the total number of Amazon S3 bytes that are [read \(p. 253\)](#) and [written \(p. 254\)](#) in the last minute by all executors as the job progresses. Both follow a similar

pattern as the data is streamed across all the executors. The job finishes processing all one million files in less than three hours.



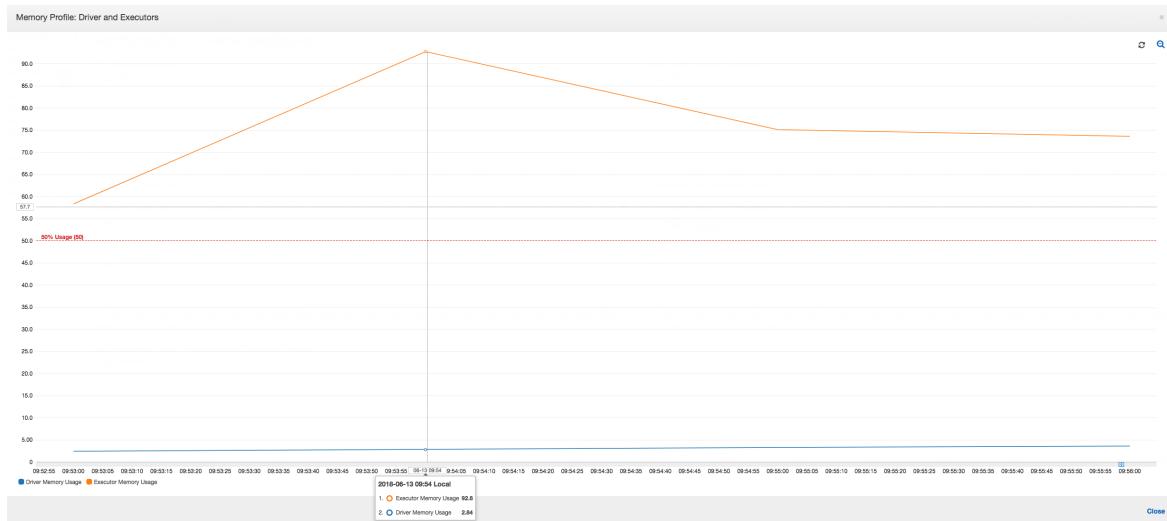
Debugging an Executor OOM Exception

In this scenario, you can learn how to debug OOM exceptions that could occur in Apache Spark executors. The following code uses the Spark MySQL reader to read a large table of about 34 million rows into a Spark dataframe. It then writes it out to Amazon S3 in Parquet format. You can provide the connection properties and use the default Spark configurations to read the table.

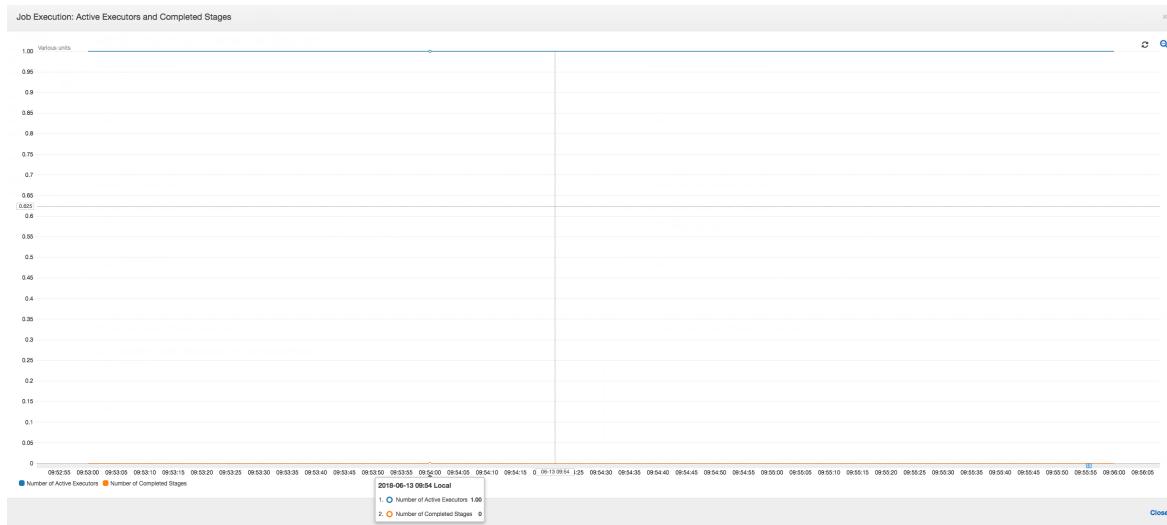
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

Visualize the Profiled Metrics on the AWS Glue Console

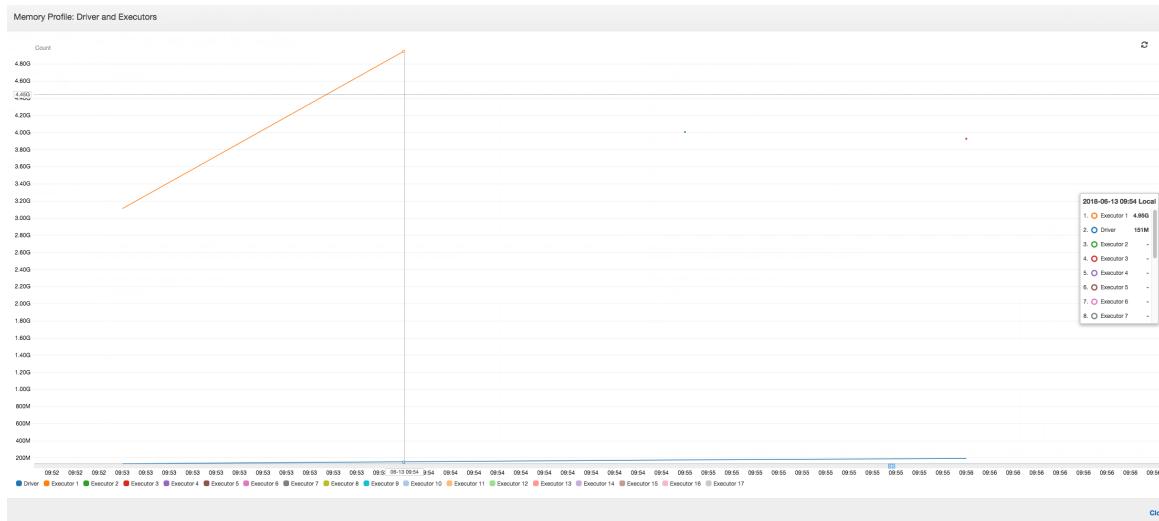
If the slope of the memory usage graph is positive and crosses 50 percent, then if the job fails before the next metric is emitted, then memory exhaustion is a good candidate for the cause. The following graph shows that within a minute of execution, the [average memory usage \(p. 251\)](#) across all executors spikes up quickly above 50 percent. The usage reaches up to 92 percent and the container running the executor is terminated ("killed") by Apache Hadoop YARN.



As the following graph shows, there is always a [single executor \(p. 249\)](#) running until the job fails. This is because a new executor is launched to replace the killed executor. The JDBC data source reads are not parallelized by default because it would require partitioning the table on a column and opening multiple connections. As a result, only one executor reads in the complete table sequentially.



As the following graph shows, Spark tries to launch a new task four times before failing the job. You can see the [memory profile \(p. 252\)](#) of three executors. Each executor quickly uses up all of its memory. The fourth executor runs out of memory, and the job fails. As a result, its metric is not reported immediately.



You can confirm from the error string on the AWS Glue console that the job failed due to OOM exceptions, as shown in the following image.

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
y_467d27235b534900d9e29...	-	Failed	WARN YarnAllocator: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.			0 secs	2880 mins			13 June 2018 9:48 AM UT...	13 June 2018 9:50 AM UT...
y_d10a9298957898152d8...	-	Succeeded				2 mins	2880 mins			13 June 2018 9:32 AM UT...	13 June 2018 9:44 AM UT...
y_dcd507823082bef919162...	-	Succeeded				2 mins	2880 mins			13 June 2018 8:57 AM UT...	13 June 2018 8:59 AM UT...
y_7a0d55298839b0c3b9e74...	-	Failed	WARN YarnAllocator: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.			1 hr, 8 mins	2880 mins			12 June 2018 5:15 PM UT...	12 June 2018 6:31 PM UT...

Job output logs: To further confirm your finding of an executor OOM exception, look at the CloudWatch Logs. When you search for **Error**, you find the four executors being killed in roughly the same time windows as shown on the metrics dashboard. All are terminated by YARN as they exceed their memory limits.

Executor 1

```

18/06/13 16:54:29 WARN YarnAllocator: Container killed by YARN for exceeding
memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN
for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 ERROR YarnClusterScheduler: Lost executor 1 on
ip-10-1-2-175.ec2.internal: Container killed by YARN for exceeding memory limits. 5.5 GB
of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0,
ip-10-1-2-175.ec2.internal, executor 1): ExecutorLostFailure (executor 1
exited caused by one of the running tasks) Reason: Container killed by YARN for
exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.

```

Executor 2

```

18/06/13 16:55:35 WARN YarnAllocator: Container killed by YARN for exceeding
memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN
for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 ERROR YarnClusterScheduler: Lost executor 2 on ip-10-1-2-16.ec2.internal:
Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory
used. Consider boosting spark.yarn.executor.memoryOverhead.

```

```
18/06/13 16:55:35 WARN TaskSetManager: Lost task 0.1 in stage 0.0 (TID 1,
ip-10-1-2-16.ec2.internal, executor 2): ExecutorLostFailure (executor 2 exited
caused by one of the running tasks) Reason: Container killed by YARN for
exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
```

Executor 3

```
18/06/13 16:56:37 WARN YarnAllocator: Container killed by YARN for exceeding
memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN
for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 ERROR YarnClusterScheduler: Lost executor 3 on
ip-10-1-2-189.ec2.internal: Container killed by YARN for exceeding memory limits. 5.8 GB
of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN TaskSetManager: Lost task 0.2 in stage 0.0 (TID 2,
ip-10-1-2-189.ec2.internal, executor 3): ExecutorLostFailure (executor 3
exited caused by one of the running tasks) Reason: Container killed by YARN for
exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
```

Executor 4

```
18/06/13 16:57:18 WARN YarnAllocator: Container killed by YARN for exceeding
memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN
for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 ERROR YarnClusterScheduler: Lost executor 4 on ip-10-1-2-96.ec2.internal:
Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory
used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN TaskSetManager: Lost task 0.3 in stage 0.0 (TID 3,
ip-10-1-2-96.ec2.internal, executor 4): ExecutorLostFailure (executor 4 exited
caused by one of the running tasks) Reason: Container killed by YARN for
exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
```

Fix the Fetch Size Setting Using AWS Glue Dynamic Frames

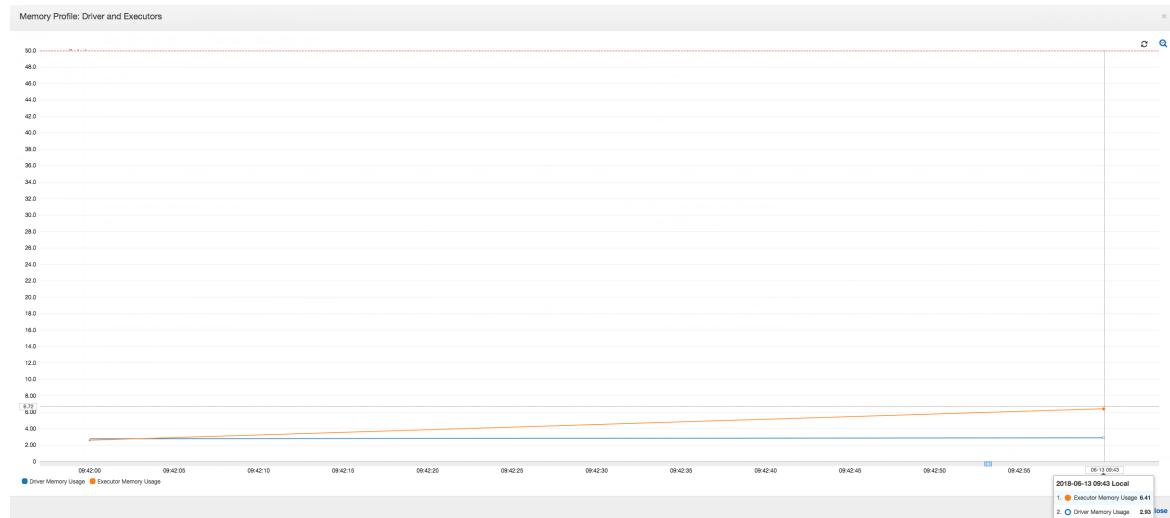
The executor ran out of memory while reading the JDBC table because the default configuration for the Spark JDBC fetch size is zero. This means that the JDBC driver on the Spark executor tries to fetch the 34 million rows from the database together and cache them, even though Spark streams through the rows one at a time. With Spark, you can avoid this scenario by setting the fetch size parameter to a non-zero default value.

You can also fix this issue by using AWS Glue dynamic frames instead. By default, dynamic frames use a fetch size of 1,000 rows that is a typically sufficient value. As a result, the executor does not take more than 7 percent of its total memory. The AWS Glue job finishes in less than two minutes with only a single executor. While using AWS Glue dynamic frames is the recommended approach, it is also possible to set the fetch size using the Apache Spark `fetchsize` property. See the [Spark SQL, DataFrames and Datasets Guide](#).

```
val (url, database, tableName) = {
  ("jdbc_url", "db_name", "table_name")
}
val source = glueContext.getSource(format, sourceJson)
```

```
val df = source.getDynamicFrame
glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3",
connection_options = {"path": output_path}, format = "parquet", transformation_ctx =
"datasink")
```

Normal profiled metrics: The [executor memory \(p. 251\)](#) with AWS Glue dynamic frames never exceeds the safe threshold, as shown in the following image. It streams in the rows from the database and caches only 1,000 rows in the JDBC driver at any point in time. An out of memory exception does not occur.



Debugging Demanding Stages and Straggler Tasks

You can use AWS Glue job profiling to identify demanding stages and straggler tasks in your extract, transform, and load (ETL) jobs. A straggler task takes much longer than the rest of the tasks in a stage of an AWS Glue job. As a result, the stage takes longer to complete, which also delays the total execution time of the job.

Coalescing Small Input Files into Larger Output Files

A straggler task can occur when there is a non-uniform distribution of work across the different tasks, or a data skew results in one task processing more data.

You can profile the following code—a common pattern in Apache Spark—to coalesce a large number of small files into larger output files. For this example, the input dataset is 32 GB of JSON Gzip compressed files. The output dataset has roughly 190 GB of uncompressed JSON files.

The profiled code is as follows:

```
datasource0 = spark.read.format("json").load("s3://input_path")
df = datasource0.coalesce(1)
df.write.format("json").save(output_path)
```

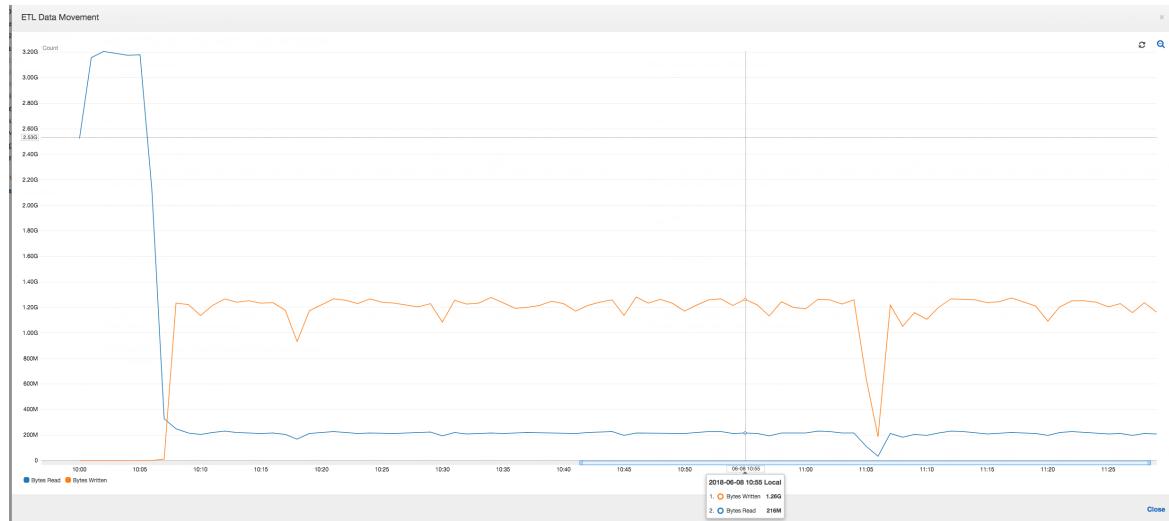
Visualize the Profiled Metrics on the AWS Glue Console

You can profile your job to examine four different sets of metrics:

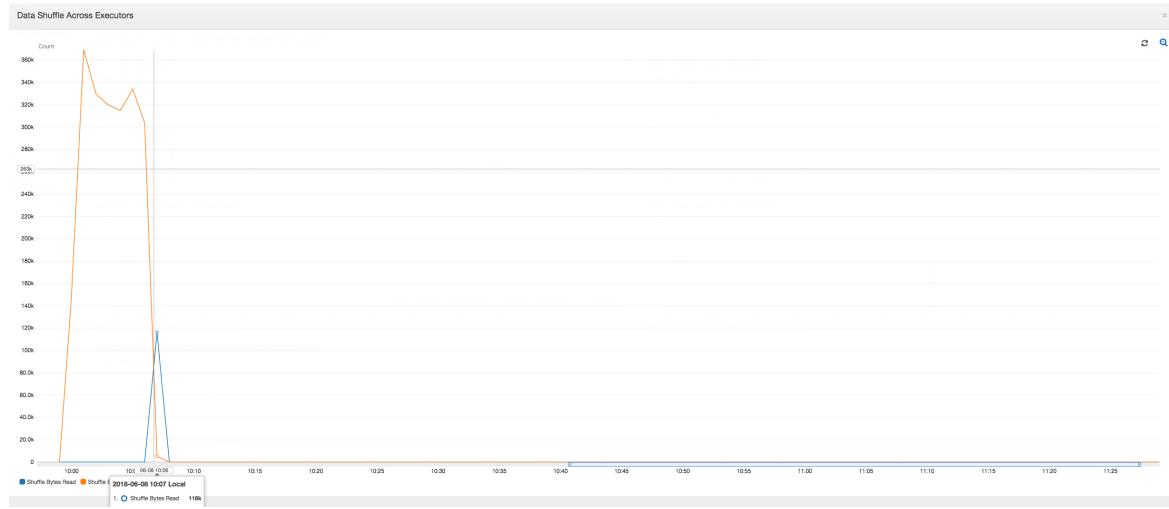
- ETL data movement
- Data shuffle across executors

- Job execution
- Memory profile

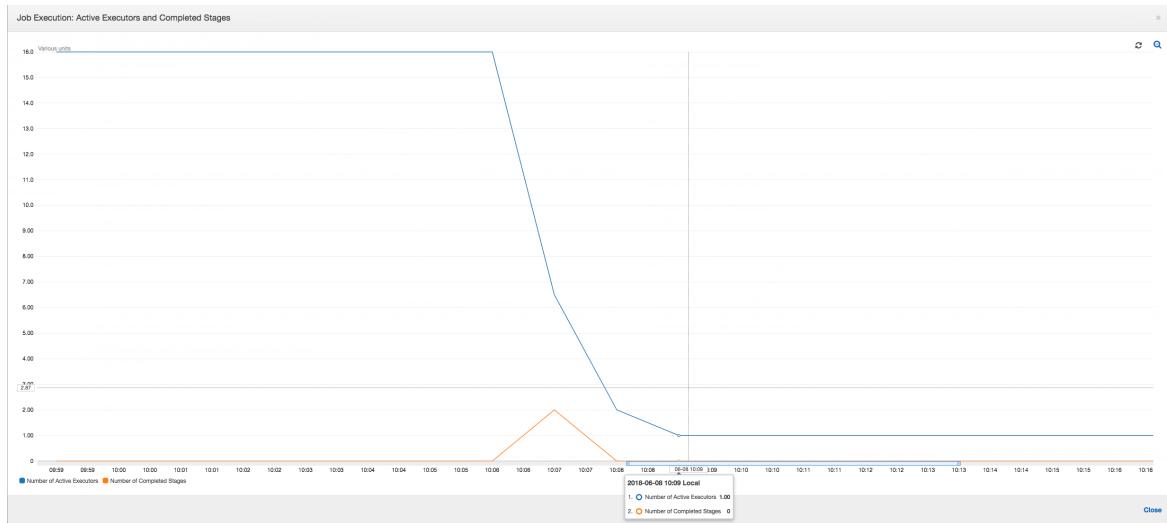
ETL data movement: In the **ETL Data Movement** profile, the bytes are [read \(p. 253\)](#) fairly quickly by all the executors in the first stage that completes within the first six minutes. However, the total job execution time is around one hour, mostly consisting of the data [writes \(p. 254\)](#).



Data shuffle across executors: The number of bytes [read \(p. 248\)](#) and [written \(p. 247\)](#) during shuffling also shows a spike before Stage 2 ends, as indicated by the **Job Execution** and **Data Shuffle** metrics. After the data shuffles from all executors, the reads and writes proceed from executor number 3 only.



Job execution: As shown in the graph below, all other executors are idle and are eventually relinquished by the time 10:09. At that point, the total number of executors decreases to only one. This clearly shows that executor number 3 consists of the straggler task that is taking the longest execution time and is contributing to most of the job execution time.



Memory profile: After the first two stages, only [executor number 3 \(p. 252\)](#) is actively consuming memory to process the data. The remaining executors are simply idle or have been relinquished shortly after the completion of the first two stages.



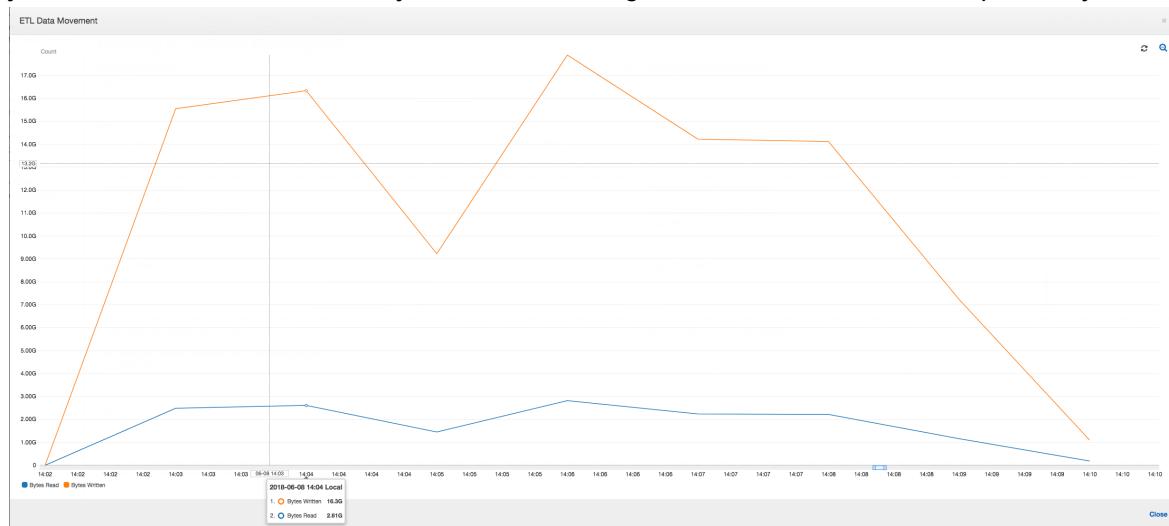
Fix Straggling Executors Using Grouping

You can avoid straggling executors by using the *grouping* feature in AWS Glue. Use grouping to distribute the data uniformly across all the executors and coalesce files into larger files using all the available executors on the cluster. For more information, see [Reading Input Files in Larger Groups \(p. 303\)](#).

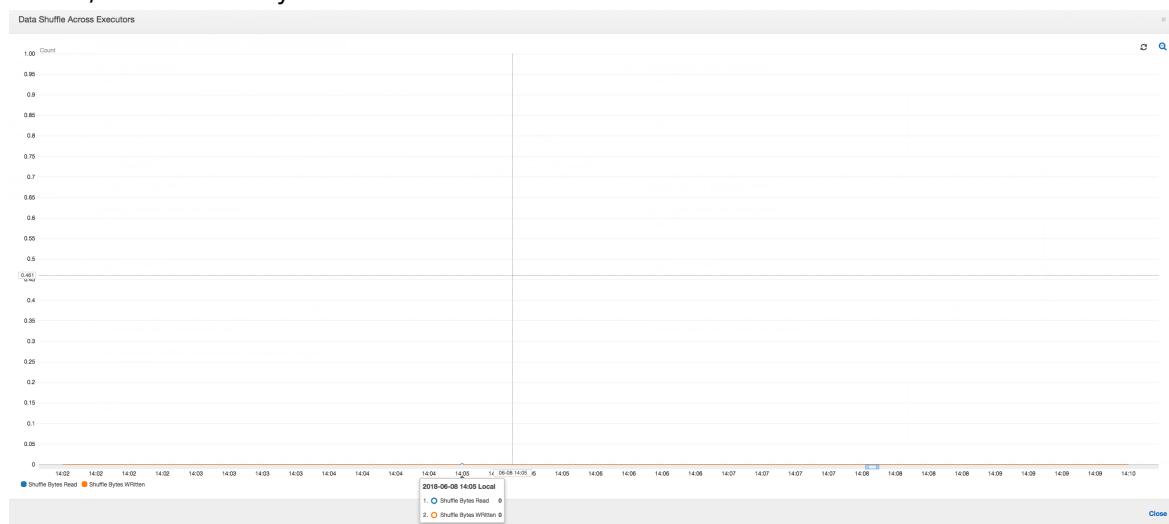
To check the ETL data movements in the AWS Glue job, profile the following code with grouping enabled:

```
df = glueContext.create_dynamic_frame_from_options("s3", {"paths": ["s3://input_path"]}, "recurse":True, 'groupFiles': 'inPartition', format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3", connection_options = {"path": "output_path"}, format = "json", transformation_ctx = "datasink4")
```

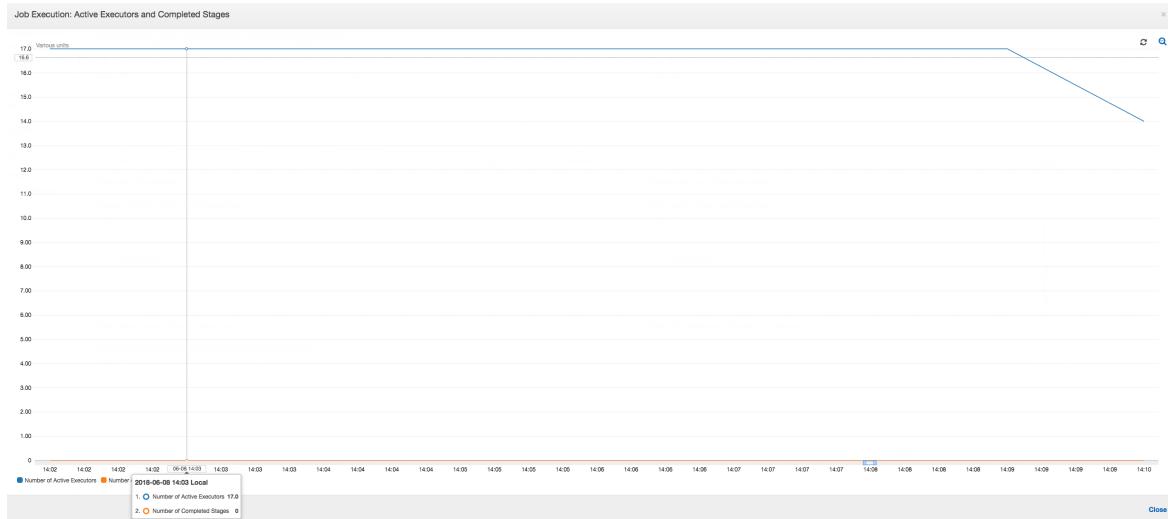
ETL data movement: The data writes are now streamed in parallel with the data reads throughout the job execution time. As a result, the job finishes within eight minutes—much faster than previously.



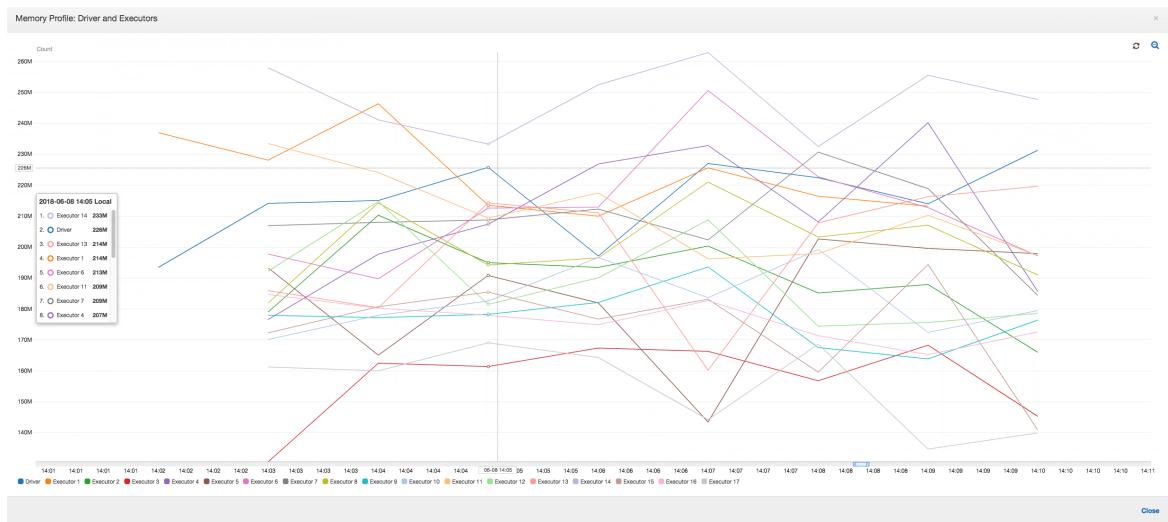
Data shuffle across executors: As the input files are coalesced during the reads using the grouping feature, there is no costly data shuffle after the data reads.



Job execution: The job execution metrics show that the total number of active executors running and processing data remains fairly constant. There is no single straggler in the job. All executors are active and are not relinquished until the completion of the job. Because there is no intermediate shuffle of data across the executors, there is only a single stage in the job.



Memory profile: The metrics show the [active memory consumption \(p. 252\)](#) across all executors—reconfirming that there is activity across all executors. As data is streamed in and written out in parallel, the total memory footprint of all executors is roughly uniform and well below the safe threshold for all executors.



Monitoring the Progress of Multiple Jobs

You can profile multiple AWS Glue jobs together and monitor the flow of data between them. This is a common workflow pattern, and requires monitoring for individual job progress, data processing backlog, data reprocessing, and job bookmarks.

Topics

- [Profiled Code \(p. 272\)](#)
- [Visualize the Profiled Metrics on the AWS Glue Console \(p. 272\)](#)
- [Fix the Processing of Files \(p. 274\)](#)

Profiled Code

In this workflow, you have two jobs: an Input job and an Output job. The Input job is scheduled to run every 30 minutes using a periodic trigger. The Output job is scheduled to run after each successful run of the Input job. These scheduled jobs are controlled using job triggers.

Triggers A trigger starts a job when it fires.					Showing: 1 - 2
Action	Trigger name	Trigger type	Trigger status	Trigger parameters	Jobs to trigger
Add trigger	e2e-bookmark-input	Schedule	ACTIVATED	Every 15 minutes	e2e-bookmark-input
Action	e2e-bookmark-output	Job events	ACTIVATED	Job events: e2ebookmark-input	e2e-bookmark

Input job: This job reads in data from an Amazon Simple Storage Service (Amazon S3) location, transforms it using `ApplyMapping`, and writes it to a staging Amazon S3 location. The following code is profiled code for the Input job:

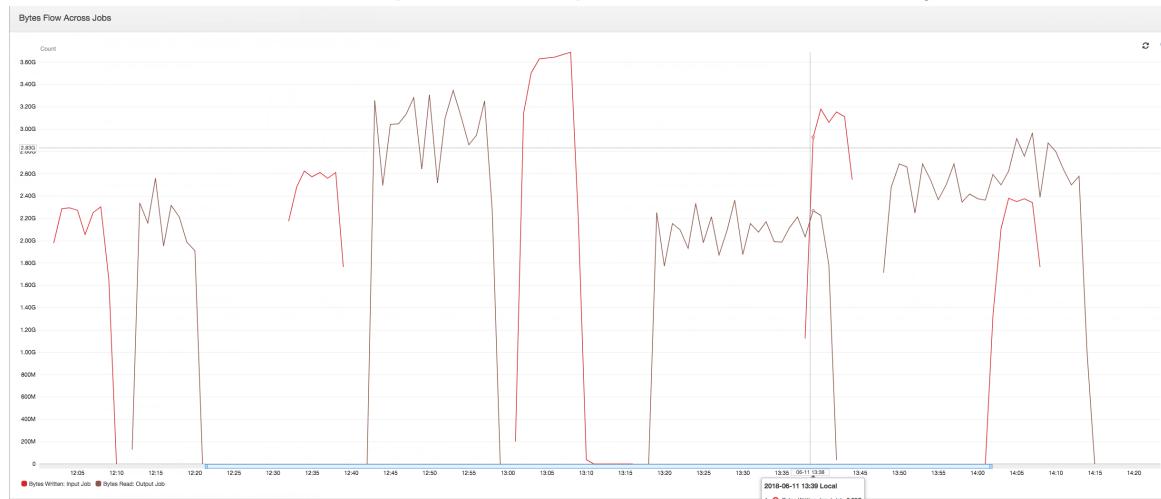
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
connection_options = {"paths": ["s3://input_path"],
"useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
connection_type = "s3", connection_options = {"path": staging_path, "compression": "gzip"}, format = "json")
```

Output job: This job reads the output of the Input job from the staging location in Amazon S3, transforms it again, and writes it to a destination:

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
connection_options = {"paths": [staging_path],
"useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
connection_type = "s3", connection_options = {"path": output_path}, format = "json")
```

Visualize the Profiled Metrics on the AWS Glue Console

The following dashboard superimposes the Amazon S3 bytes written metric from the Input job onto the Amazon S3 bytes read metric on the same timeline for the Output job. The timeline shows different job runs of the Input and Output jobs. The Input job (shown in red) starts every 30 minutes. The Output Job (shown in brown) starts at the completion of the Input Job, with a Max Concurrency of 1.



In this example, [job bookmarks](#) are not enabled. No transformation contexts are used to enable job bookmarks in the script code.

Job History: The Input and Output jobs have multiple runs, as shown on the **History** tab, starting from 12:00 PM.

The Input job on the AWS Glue console looks like this:

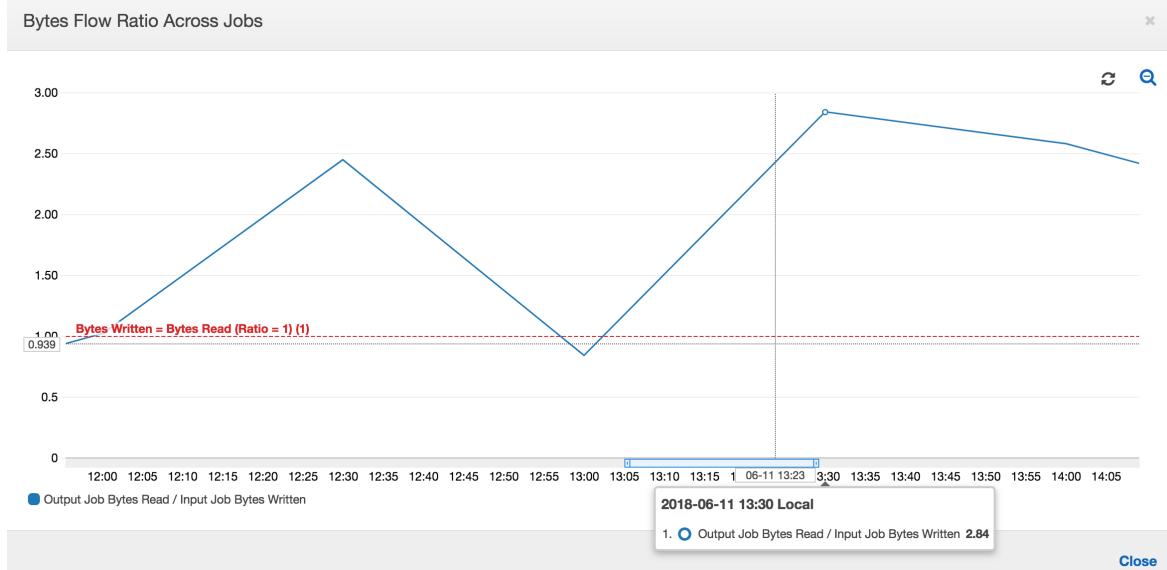
Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
j_r_0ce47fb1a561051fdccae95e...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:30 PM UT...	11 June 2018 2:40 PM UT...
j_r_1b49ec0f3607611cc0e24274...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:30 PM UT...	11 June 2018 2:10 PM UT...
j_r_1cfe65350cbe16e89099681...	-	Succeeded		Logs		7 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:30 PM UT...	11 June 2018 1:46 PM UT...
j_r_f52949097744bd2ebfb55b61...	-	Succeeded		Logs		15 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:00 PM UT...	11 June 2018 1:16 PM UT...
...	Logs	

The following image shows the Output job:

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
j_r_0e6ba78770743d73dbdd63...	-	Failed	Max conc...	Logs	Error logs	0 secs	2880 mins		e2e-bookmark-output	11 June 2018 2:11 PM UT...	
j_r_3a242babab084ddcd95d2d23...	-	Succeeded		Logs		27 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:47 PM UT...	11 June 2018 2:15 PM UT...
j_r_c98cc0031be79462b3a8047b...	-	Succeeded		Logs		24 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:17 PM UT...	11 June 2018 1:43 PM UT...
j_r_02943d966c395d595e9f65...	-	Succeeded		Logs		17 mins	2880 mins		e2e-bookmark-output	11 June 2018 12:41 PM UT...	11 June 2018 12:59 PM UT...
...	Logs	

First job runs: As shown in the Data Bytes Read and Written graph below, the first job runs of the Input and Output jobs between 12:00 and 12:30 show roughly the same area under the curves. Those areas represent the Amazon S3 bytes written by the Input job and the Amazon S3 bytes read by the Output job. This data is also confirmed by the ratio of Amazon S3 bytes written (summed over 30 minutes – the job trigger frequency for the Input job). The data point for the ratio for the Input job run that started at 12:00PM is also 1.

The following graph shows the data flow ratio across all the job runs:



Second job runs: In the second job run, there is a clear difference in the number of bytes read by the Output job compared to the number of bytes written by the Input job. (Compare the area under the curve across the two job runs for the Output job, or compare the areas in the second run of the Input and Output jobs.) The ratio of the bytes read and written shows that the Output Job read about 2.5x the data written by the Input job in the second span of 30 minutes from 12:30 to 13:00. This is because the Output Job reprocessed the output of the first job run of the Input job because job bookmarks were not enabled. A ratio above 1 shows that there is an additional backlog of data that was processed by the Output job.

Third job runs: The Input job is fairly consistent in terms of the number of bytes written (see the area under the red curves). However, the third job run of the Input job ran longer than expected (see the long tail of the red curve). As a result, the third job run of the Output job started late. The third job run processed only a fraction of the data accumulated in the staging location in the remaining 30 minutes between 13:00 and 13:30. The ratio of the bytes flow shows that it only processed 0.83 of data written by the third job run of the Input job (see the ratio at 13:00).

Overlap of Input and Output jobs: The fourth job run of the Input job started at 13:30 as per the schedule, before the third job run of the Output job finished. There is a partial overlap between these two job runs. However, the third job run of the Output job captures only the files that it listed in the staging location of Amazon S3 when it began around 13:17. This consists of all data output from the first job runs of the Input job. The actual ratio at 13:30 is around 2.75. The third job run of the Output job processed about 2.75x of data written by the fourth job run of the Input job from 13:30 to 14:00.

As these images show, the Output job is reprocessing data from the staging location from all prior job runs of the Input job. As a result, the fourth job run for the Output job is the longest and overlaps with the entire fifth job run of the Input job.

Fix the Processing of Files

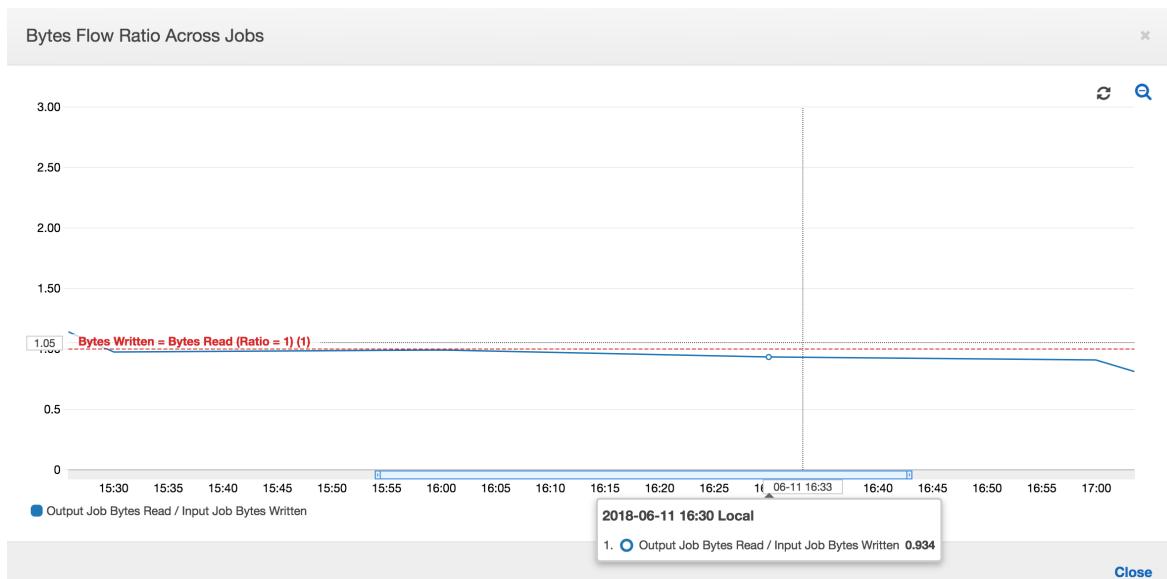
You should ensure that the Output job processes only the files that haven't been processed by previous job runs of the Output job. To do this, enable job bookmarks and set the transformation context in the Output job, as follows:

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
    connection_options = {"paths": [staging_path],
    "useS3ListImplementation":True,"recurse":True}, format="json", transformation_ctx =
"bookmark_ctx")
```

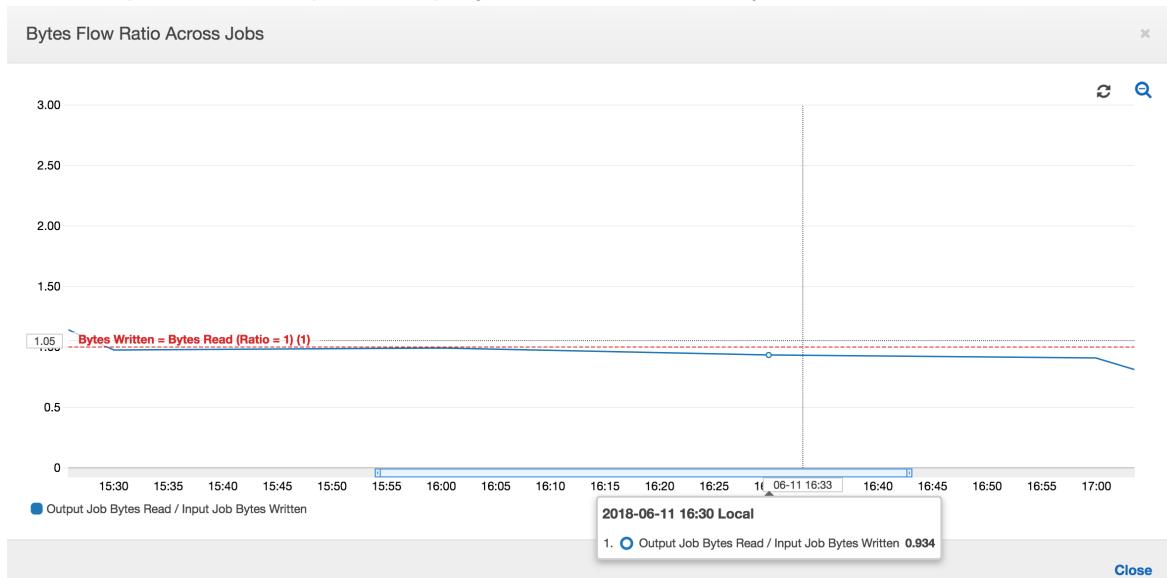
With job bookmarks enabled, the Output job doesn't reprocess the data in the staging location from all the previous job runs of the Input job. In the following image showing the data read and written, the area under the brown curve is fairly consistent and similar with the red curves.



The ratios of byte flow also remain roughly close to 1 because there is no additional data processed.



A job run for the Output job starts and captures the files in the staging location before the next Input job run starts putting more data into the staging location. As long as it continues to do this, it processes only the files captured from the previous Input job run, and the ratio stays close to 1.



Suppose that the Input job takes longer than expected, and as a result, the Output job captures files in the staging location from two Input job runs. The ratio is then higher than 1 for that Output job run. However, the following job runs of the Output job don't process any files that are already processed by the previous job runs of the Output job.

Monitoring for DPU Capacity Planning

You can use job metrics in AWS Glue to estimate the number of data processing units (DPUs) that can be used to scale out an AWS Glue job.

Topics

- [Profiled Code \(p. 276\)](#)

- [Visualize the Profiled Metrics on the AWS Glue Console \(p. 276\)](#)
- [Determine the Optimal DPU Capacity \(p. 278\)](#)

Profiled Code

The following script reads an Amazon Simple Storage Service (Amazon S3) partition containing 428 gzipped JSON files. The script applies a mapping to change the field names, and converts and writes them to Amazon S3 in Apache Parquet format. You provision 10 DPUs as per the default and execute this job.

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
    connection_options = {"paths": [input_path],
    "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [(map_spec)])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
    connection_type = "s3", connection_options = {"path": output_path}, format = "parquet")
```

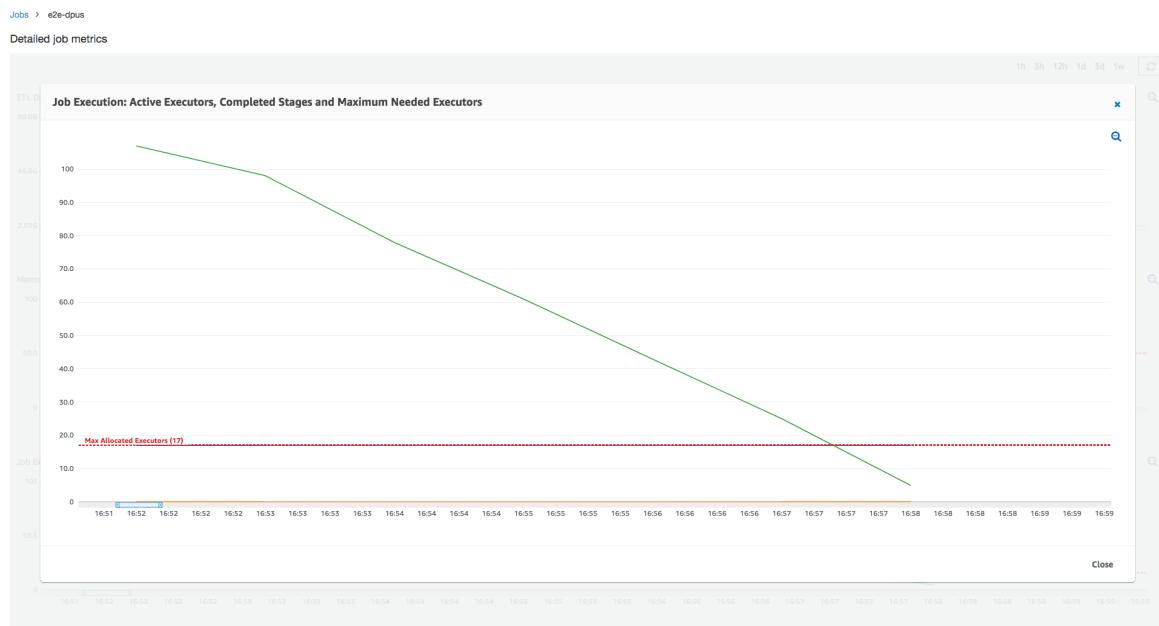
Visualize the Profiled Metrics on the AWS Glue Console

Job Run 1: In this job run we show how to find if there are under-provisioned DPUs in the cluster. The job execution functionality in AWS Glue shows the total [number of actively running executors \(p. 249\)](#), the [number of completed stages \(p. 245\)](#), and the [number of maximum needed executors \(p. 250\)](#).

The number of maximum needed executors is computed by adding the total number of running tasks and pending tasks, and dividing by the tasks per executor. This result is a measure of the total number of executors required to satisfy the current load.

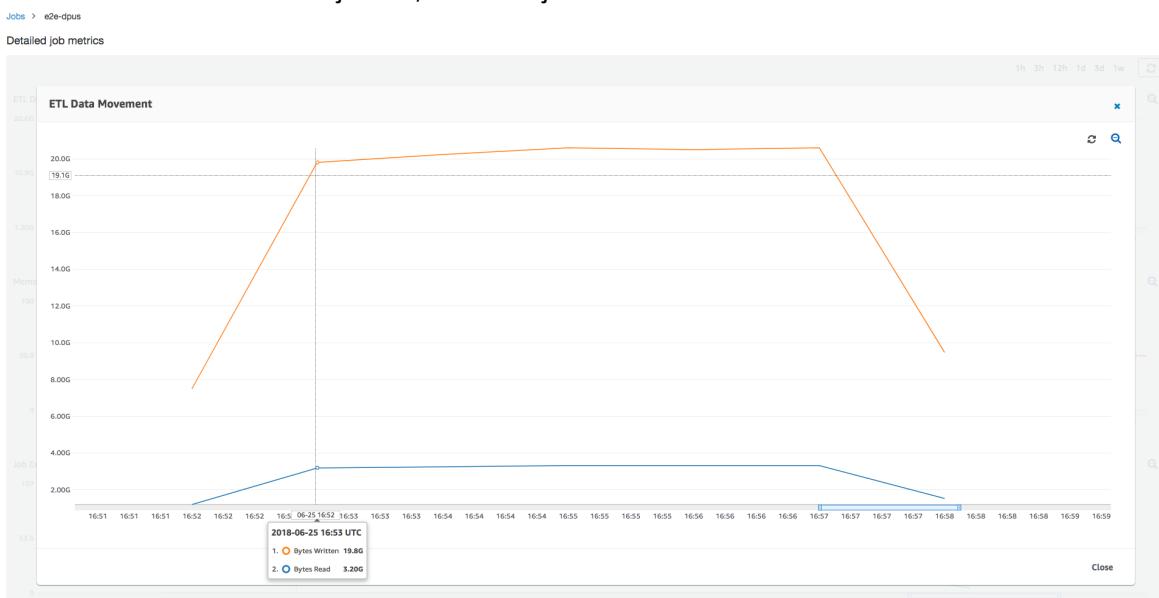
In contrast, the number of actively running executors measures how many executors are running active Apache Spark tasks. As the job progresses, the maximum needed executors can change and typically goes down towards the end of the job as the pending task queue diminishes.

The horizontal red line in the following graph shows the number of maximum allocated executors, which depends on the number of DPUs that you allocate for the job. In this case, you allocate 10 DPUs for the job run. One DPU is reserved for the master. Nine DPUs run two executors each and one executor is reserved for the Spark driver. The Spark driver runs inside the application master. So, the number of maximum allocated executors is $2 \times 9 - 1 = 17$ executors.



As the graph shows, the number of maximum needed executors starts at 107 at the beginning of the job, whereas the number of active executors remains 17. This is the same as the number of maximum allocated executors with 10 DPU. The ratio between the maximum needed executors and maximum allocated executors (adding 1 to both for the Spark driver) gives you the under-provisioning factor: $108/18 = 6x$. You can provision 6 (under provisioning ratio) * 9 (current DPU capacity - 1) + 1 DPU = 55 DPU to scale out the job to run it with maximum parallelism and finish faster.

The AWS Glue console displays the detailed job metrics as a static line representing the original number of maximum allocated executors. The console computes the maximum allocated executors from the job definition for the metrics. By contrast, for detailed job run metrics, the console computes the maximum allocated executors from the job run configuration, specifically the DPU allocated for the job run. To view metrics for an individual job run, select the job run and choose **View run metrics**.

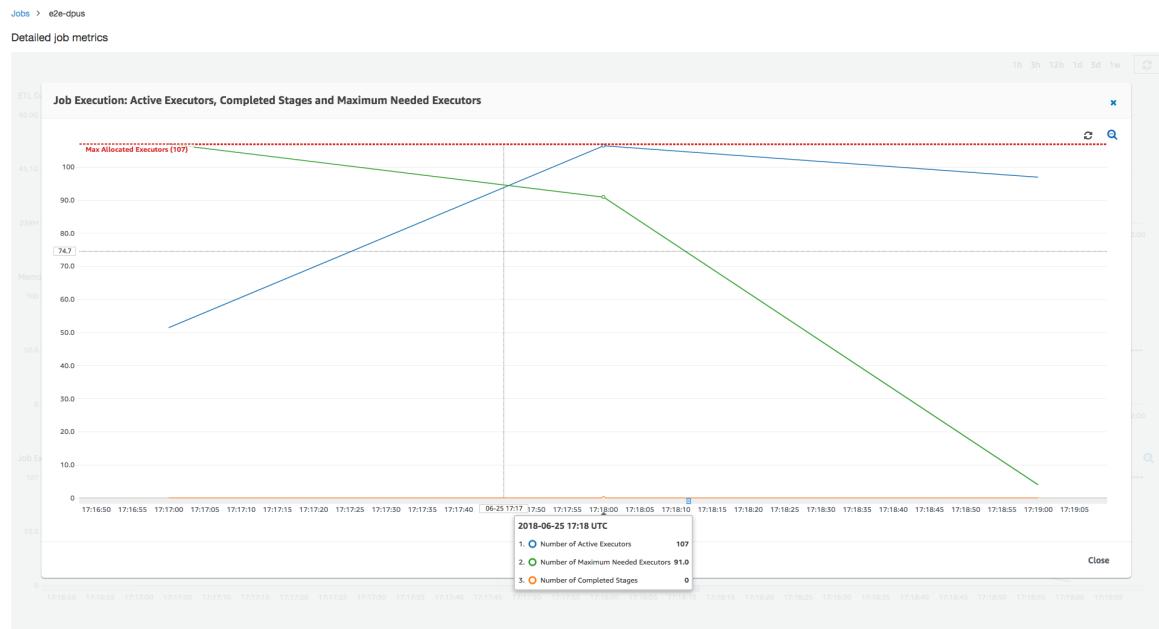


Looking at the Amazon S3 bytes [read \(p. 253\)](#) and [written \(p. 254\)](#), notice that the job spends all six minutes streaming in data from Amazon S3 and writing it out in parallel. All the cores on the allocated

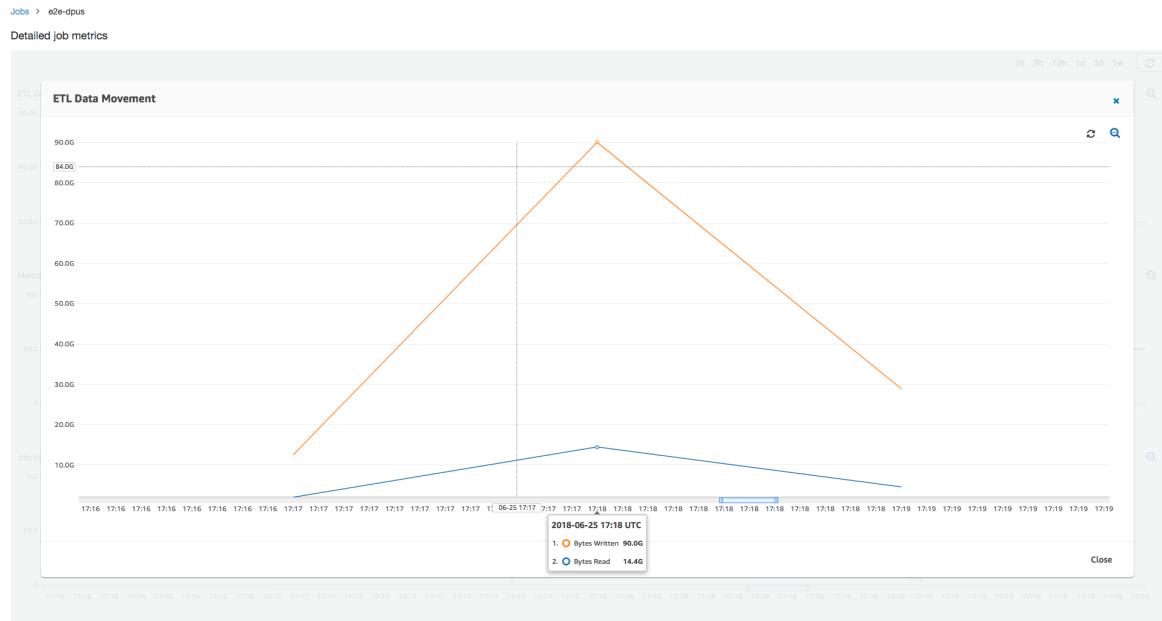
DPUs are reading and writing to Amazon S3. The maximum number of needed executors being 107 also matches the number of files in the input Amazon S3 path—428. Each executor can launch four Spark tasks to process four input files (JSON gzipped).

Determine the Optimal DPU Capacity

Based on the results of the previous job run, you can increase the total number of allocated DPUs to 55, and see how the job performs. The job finishes in less than three minutes—half the time it required previously. The job scale-out is not linear in this case because it is a short running job. Jobs with long-lived tasks or a large number of tasks (a large number of max needed executors) benefit from a close-to-linear DPU scale-out performance speedup.



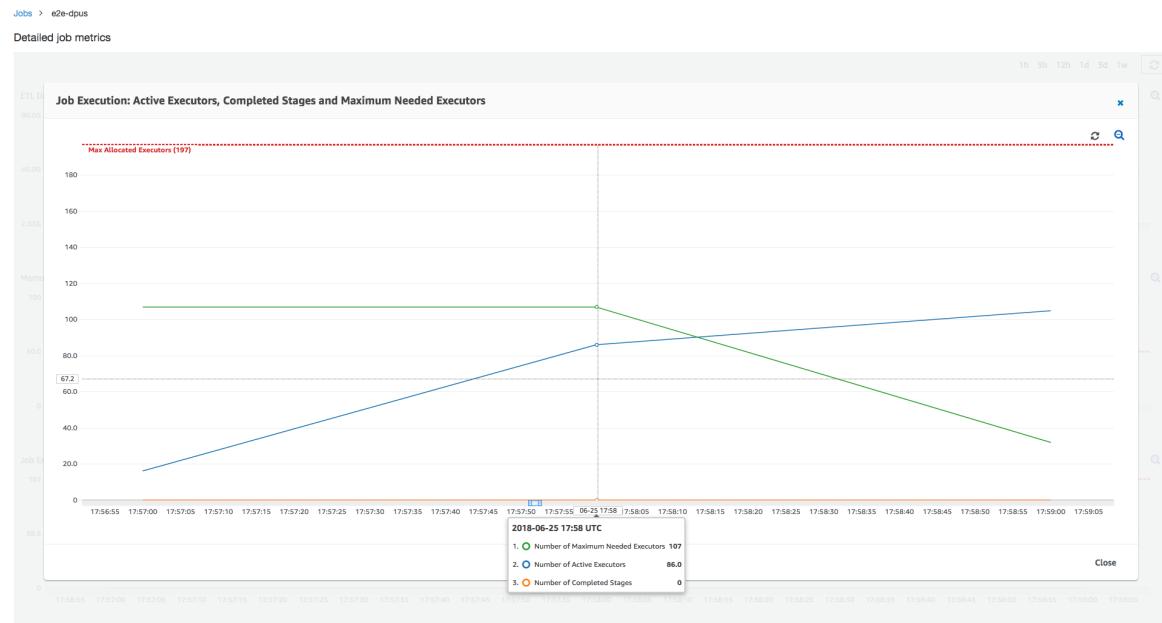
As the above image shows, the total number of active executors reaches the maximum allocated—107 executors. Similarly, the maximum needed executors is never above the maximum allocated executors. The maximum needed executors number is computed from the actively running and pending task counts, so it might be smaller than the number of active executors. This is because there can be executors that are partially or completely idle for a short period of time and are not yet decommissioned.



This job run uses 6x more executors to read and write from Amazon S3 in parallel. As a result, this job run uses more Amazon S3 bandwidth for both reads and writes, and finishes faster.

Identify Overprovisioned DPUs

Next, you can determine whether scaling out the job with 100 DPUs ($99 * 2 = 198$ executors) helps to scale out any further. As the following graph shows, the job still takes three minutes to finish. Similarly, the job does not scale out beyond 107 executors (55 DPUs configuration), and the remaining 91 executors are overprovisioned and not used at all. This shows that increasing the number of DPUs might not always improve performance, as evident from the maximum needed executors.



Compare Time Differences

The three job runs shown in the following table summarize the job execution times for 10 DPUs, 55 DPUs, and 100 DPUs. You can find the DPU capacity to improve the job execution time using the estimates you established by monitoring the first job run.

Job ID	Number of DPUs	Execution Time
jr_c894524c8ef5048a4d9...	10	6 min.
jr_1a466cf2575e7ffe6856...	55	3 min.
jr_34fa1ed4c6aa9ff0a814...	100	3 min.

Logging AWS Glue API Calls with AWS CloudTrail

AWS Glue is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Glue. CloudTrail captures all API calls for AWS Glue as events. The calls captured include calls from the AWS Glue console and code calls to the AWS Glue API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Glue. 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 AWS Glue, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

AWS Glue Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Glue, 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 AWS Glue, 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 AWS Glue actions are logged by CloudTrail and are documented in the [AWS Glue API \(p. 452\)](#). For example, calls to the `CreateDatabase`, `CreateTable` and `CreateScript` 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](#).

However, CloudTrail doesn't log all information regarding calls. For example, it doesn't log certain sensitive information, such as the `ConnectionProperties` used in connection requests, and it logs a `null` instead of the responses returned by the following APIs:

BatchGetPartition	GetCrawlers	GetJobs	GetTable
CreateScript	GetCrawlerMetrics	GetJobRun	GetTables
GetCatalogImportStatus	GetDatabase	GetJobRuns	GetTableVersions
GetClassifier	GetDataBases	GetMapping	GetTrigger
GetClassifiers	GetDataflowGraph	GetObject	GetTriggers
GetConnection	GetDevEndpoint	GetPartition	GetUserDefinedFunction
GetConnections	GetDevEndpoints	GetPartitions	GetUserDefinedFunctions
GetCrawler	GetJob	GetPlan	

Understanding AWS Glue Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `DeleteCrawler` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-11T22:29:49Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "DeleteCrawler",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.64",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
  "requestParameters": {
    "name": "tes-alpha"
  },
  "responseElements": null,
  "requestID": "b16f4050-aed3-11e7-b0b3-75564a46954f",
  "eventID": "e73dd117-cfd1-47d1-9e2f-d1271cad838c",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

This example shows a CloudTrail log entry that demonstrates a `CreateConnection` action.

```
{
```

```
"eventVersion": "1.05",
"userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
},
"eventTime": "2017-10-13T00:19:19Z",
"eventSource": "glue.amazonaws.com",
"eventName": "CreateConnection",
"awsRegion": "us-east-1",
"sourceIPAddress": "72.21.198.66",
"userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
"requestParameters": {
    "connectionInput": {
        "name": "test-connection-alpha",
        "connectionType": "JDBC",
        "physicalConnectionRequirements": {
            "subnetId": "subnet-323232",
            "availabilityZone": "us-east-1a",
            "securityGroupIdList": [
                "sg-12121212"
            ]
        }
    }
},
"responseElements": null,
"requestID": "27136ebc-afac-11e7-a7d6-ab217e5c3f19",
"eventID": "e8b3baeb-c511-4597-880f-c16210c60a4a",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

Performing Complex ETL Activities Using Workflows in AWS Glue

A *workflow* is a container for a set of related jobs, crawlers, and triggers in AWS Glue. Using a workflow, you can design a complex multi-job extract, transform, and load (ETL) activity that AWS Glue can execute and track as single entity.

You can create workflows using the AWS Management Console or the AWS Glue API. The console enables you to visualize the components and flow of a workflow with a graph.

Topics

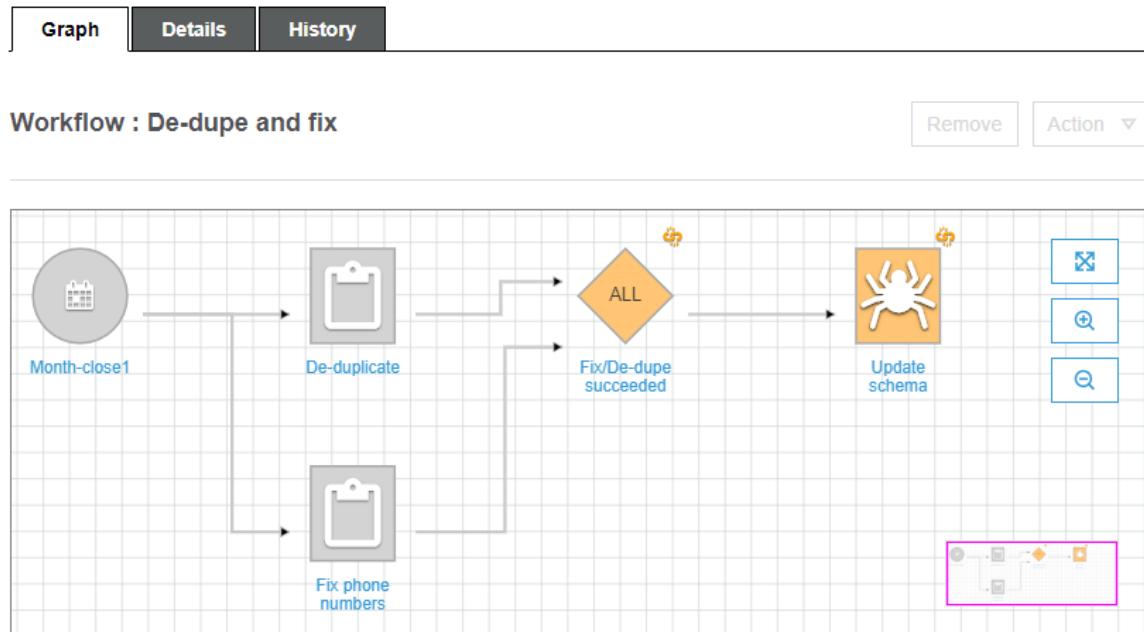
- [Overview of Workflows \(p. 283\)](#)
- [Creating and Running Workflows \(p. 284\)](#)
- [Getting and Setting Workflow Run Properties \(p. 287\)](#)
- [Querying Workflows Using the AWS Glue API \(p. 288\)](#)

Overview of Workflows

In AWS Glue, you can use workflows to create and visualize complex extract, transform, and load (ETL) activities involving multiple crawlers, jobs, and triggers. Each workflow manages the execution and monitoring of all its components. As a workflow runs each component, it records execution progress and status, providing you with an overview of the larger task and the details of each step. The AWS Glue console provides a visual representation of a workflow as a graph.

To share and manage state throughout a workflow run, you can define default workflow run properties. These properties, which are name/value pairs, are available to all the jobs in the workflow. Using the AWS Glue API, jobs can retrieve the workflow run properties and modify them for jobs that come later in the workflow.

The following image shows the graph of a basic workflow on the AWS Glue console. Your workflow could have dozens of components.



This workflow is started by a schedule trigger, which starts two jobs. Upon successful completion of both jobs, an event trigger starts a crawler.

Static and Dynamic Workflow Views

For each workflow, there is the notion of *static view* and *dynamic view*. The static view indicates the design of the workflow. The dynamic view is a run time view that includes the latest run information for each of the jobs and crawlers. Run information includes success status and error details.

When a workflow is running, the console displays the dynamic view, graphically indicating the jobs that have completed and that are yet to be run. You can also retrieve a dynamic view of a running workflow using the AWS Glue API. For more information, see [Querying Workflows Using the AWS Glue API \(p. 288\)](#).

Workflow Restrictions

Keep the following workflow restrictions in mind:

- A trigger can be associated with only one workflow.
- Only one starting trigger (on-demand or schedule) is permitted.

Creating and Running Workflows

You can use the AWS Glue console to create, visualize, and run workflows. For information about managing workflows using the AWS Glue API, see [Workflows \(p. 570\)](#).

Topics

- [Creating and Building Out a Workflow Using the Console \(p. 285\)](#)
- [Running a Workflow \(p. 287\)](#)

Creating and Building Out a Workflow Using the Console

A workflow contains jobs, crawlers, and triggers. Before creating a workflow, create the jobs and crawlers that the workflow is to include. It is best to specify run-on-demand crawlers for workflows. You can create new triggers while you are building out your workflow, or you can *clone* existing triggers into the workflow. When you clone a trigger, all the catalog objects associated with the trigger—the jobs or crawlers that fire it and the jobs or crawlers that it starts—are added to the workflow.

You build out your workflow by adding triggers to the workflow graph, and defining the watched events and actions for each trigger. You begin with a *start trigger*, which can be either an on-demand or schedule trigger, and complete the graph by adding event (conditional) triggers.

Step 1: Create the workflow

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **ETL**, choose **Workflows**.
3. Choose **Add workflow** and complete the **Add a new ETL workflow** form.

Any optional default run properties that you add are made available as arguments to all jobs in the workflow. For more information, see [Getting and Setting Workflow Run Properties \(p. 287\)](#).

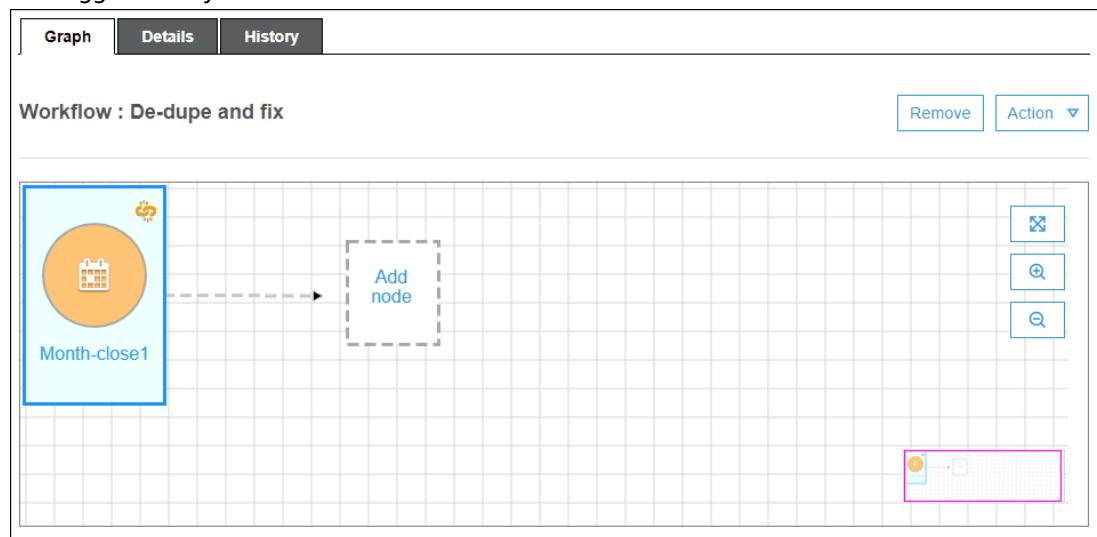
4. Choose **Add workflow**.

The new workflow appears in the list on the **Workflows** page.

Step 2: Add a start trigger

1. On the **Workflows** page, select your new workflow. In the tabs at the bottom, choose **Graph**.
2. Choose **Add trigger**, and in the **Add trigger** dialog box, do one of the following:
 - Choose **Add new**, and complete the **Add trigger** form, selecting **Schedule** or **On demand** for **Trigger Type**. Then choose **Add**.

The trigger appears on the graph, along with a placeholder node (labeled **Add node**). At this point, the trigger is not yet saved.



- Choose **Clone existing**, and choose a trigger to clone. Then choose **Add**.

The trigger appears on the graph, along with the jobs and crawlers that it watches and the jobs and crawlers that it starts.

If you mistakenly selected the wrong trigger, select the trigger on the graph, and then choose **Remove**.

3. If you added a new trigger, complete these steps:

- a. Do one of the following:

- Choose the placeholder node (**Add node**).
- Ensure that the start trigger is selected, and on the **Action** menu above the graph, choose **Add jobs/crawlers to trigger**.

- b. In the **Add jobs(s) and crawler(s) to trigger** dialog box, select one or more jobs or crawlers, and then choose **Add**.

The trigger is saved, and the selected jobs or crawlers appear on the graph with connectors from the trigger.

If you mistakenly added the wrong jobs or crawlers, you can select either the trigger or a connector and choose **Remove**.

Step 3: (Optional) Add more triggers

Continue to build out your workflow by adding more triggers. To zoom in or out or to enlarge the graph canvas, use the icons to the right of the graph. For each trigger to add, complete the following steps:

1. Do one of the following:

- To clone an existing trigger, ensure that no node on the graph is selected, and on the **Action** menu, choose **Add trigger**.
- To add a new trigger that watches a particular job or crawler on the graph, select the job or crawler node, and then choose the **Add trigger** placeholder node.

You can add more jobs or crawlers to watch for this trigger in a later step.

2. In the **Add trigger** dialog box, do one of the following:

- Choose **Add new**, and complete the **Add trigger** form. Then choose **Add**.

The trigger appears on the graph. You will complete the trigger in a later step.

- Choose **Clone existing**, and choose a trigger to clone. Then choose **Add**.

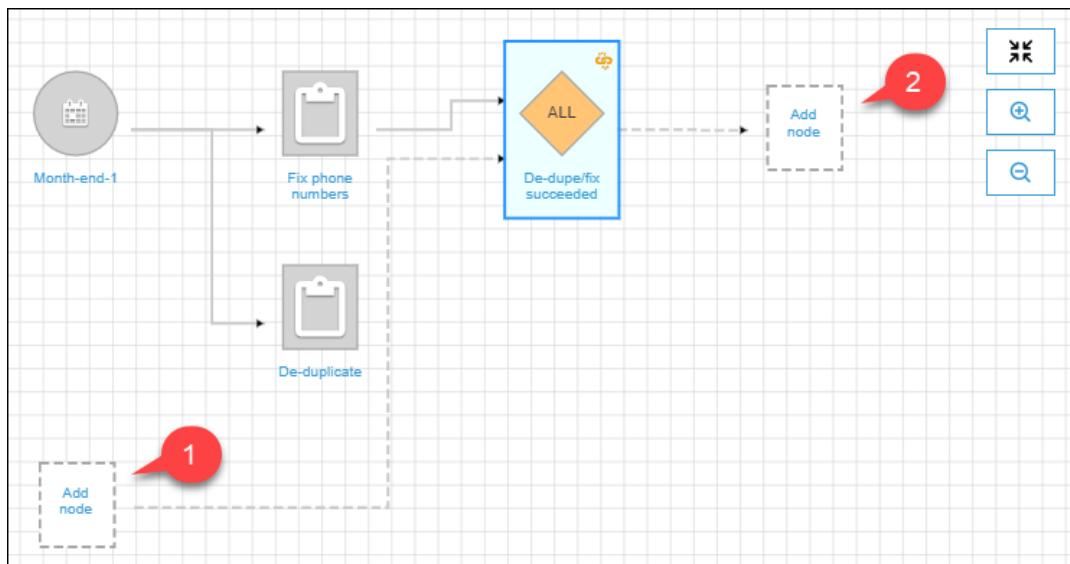
The trigger appears on the graph, along with the jobs and crawlers that it watches and the jobs and crawlers that it starts.

If you mistakenly chose the wrong trigger, select the trigger on the graph, and then choose **Remove**.

3. If you added a new trigger, complete these steps:

- a. Select the new trigger.

As the following graph shows, placeholder nodes appear for (1) events to watch and (2) actions.



- b. (Optional if the trigger already watches an event and you want to add more jobs or crawlers to watch.) Choose the events-to-watch placeholder node, and in the **Add job(s) and crawler(s) to watch** dialog box, select one or more jobs or crawlers. Choose an event to watch (SUCCEEDED, FAILED, etc.), and choose **Add**.
- c. Ensure that the trigger is selected, and choose the actions placeholder node.
- d. In the **Add job(s) and crawler(s) to watch** dialog box, select one or more jobs or crawlers, and choose **Add**.

The selected jobs and crawlers appear on the graph, with connectors from the trigger.

Running a Workflow

If the start trigger for a workflow is an on-demand trigger, you can start the workflow from the AWS Glue console.

To run a workflow

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **ETL**, choose **Workflows**.
3. Select a workflow. On the **Actions** menu, choose **Run**.

Getting and Setting Workflow Run Properties

Use workflow run properties to share and manage state among the jobs in your AWS Glue workflow. You can set default run properties when you create the workflow. Then, as your jobs run, they can retrieve the run property values and optionally modify them for input to jobs that are later in the workflow. When a job modifies a run property, the new value exists only for the workflow run; the default run properties are not affected.

The following sample Python code from an extract, transform, and load (ETL) job demonstrates how to get the workflow run properties.

```
import sys
```

```
import boto3
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from pyspark.context import SparkContext

glue_client = boto3.client("glue")
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'WORKFLOW_NAME', 'WORKFLOW_RUN_ID'])
workflow_name = args['WORKFLOW_NAME']
workflow_run_id = args['WORKFLOW_RUN_ID']
workflow_params = glue_client.get_workflow_run_properties(Name=workflow_name,
                                                          RunId=workflow_run_id)[ "RunProperties" ]

target_database = workflow_params['target_database']
target_s3_location = workflow_params['target_s3_location']
```

The following code continues by setting the `target_format` run property to 'csv'.

```
workflow_params['target_format'] = 'csv'
glue_client.put_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id,
                                         RunProperties=workflow_params)
```

For more information, see the following:

- [GetWorkflowRunProperties Action \(Python: `get_workflow_run_properties`\) \(p. 579\)](#)
- [PutWorkflowRunProperties Action \(Python: `put_workflow_run_properties`\) \(p. 579\)](#)

Querying Workflows Using the AWS Glue API

AWS Glue provides a rich API for managing workflows. You can retrieve a static view of a workflow or a dynamic view of a running workflow using the AWS Glue API. For more information, see [Workflows \(p. 570\)](#).

Topics

- [Querying Static Views \(p. 288\)](#)
- [Querying Dynamic Views \(p. 289\)](#)

Querying Static Views

Use the `GetWorkflow` API operation to get a static view that indicates the design of a workflow. This operation returns a directed graph consisting of nodes and edges, where a node represents a trigger, a job, or a crawler. Edges define the relationships between nodes. They are represented by connectors (arrows) on the graph in the AWS Glue console. You can also use this operation with popular graph-processing libraries such as NetworkX, igraph, JGraphT, and the Java Universal Network/Graph (JUNG) Framework. Because all these libraries represent graphs similarly, minimal transformations are needed.

The static view returned by this API is the most up-to-date view according to the latest definition of triggers associated with the workflow.

Graph Definition

A workflow graph G is an ordered pair (N, E) , where N is a set of nodes and E a set of edges. *Node* is a vertex in the graph identified by a unique number. A node can be of type trigger, job, or crawler; for example: `{name:T1, type:Trigger, uniqueId:1}`, `{name:J1, type:Job, uniqueId:2}`

Edge is a 2-tuple of the form `(src, dest)`, where `src` and `dest` are nodes and there is a directed edge from `src` to `dest`.

Example of Querying a Static View

Consider a conditional trigger `T`, which triggers job `J2` upon completion of job `J1`.

```
J1 ----> T ----> J2
```

Nodes: `J1, T, J2`

Edges: `(J1, T), (T, J2)`

Querying Dynamic Views

Use the `GetWorkflowRun` API operation to get a dynamic view of a running workflow. This operation returns the same static view of the graph along with metadata related to the workflow run.

For instance, nodes representing jobs in the `GetWorkflowRun` call have a list of job runs initiated as part of the latest run of the workflow. You can use this list to display the run status of each job in the graph itself. For downstream dependencies that are not yet executed, this field is set to `null`. The graphed information makes you aware of the current state of any workflow at any point of time.

The dynamic view returned by this API is as per the static view that was present when the workflow run was started.

Run-time nodes example: `{name:T1, type: Trigger, uniqueId:1}, {name:J1, type:Job, uniqueId:2, jobDetails:{jobRuns}}, {name:C1, type:Crawler, uniqueId:3, crawlerDetails:{crawls}}`

Example 1: Dynamic View

The following example illustrates a simple two-trigger workflow.

- Nodes: `t1, j1, t2, j2`
- Edges: `(t1, j1), (j1, t2), (t2, j2)`

The `GetWorkflow` response contains the following.

```
{
  Nodes : [
    {
      "type" : Trigger,
      "name" : "t1",
      "uniqueId" : 1
    },
    {
      "type" : Job,
      "name" : "j1",
      "uniqueId" : 2
    },
    {
      "type" : Trigger,
      "name" : "t2",
      "uniqueId" : 3
    },
    {
      "type" : Job,
```

```

        "name" : "j2",
        "uniqueId" : 4
    }
],
Edges : [
{
    "sourceId" : 1,
    "destinationId" : 2
},
{
    "sourceId" : 2,
    "destinationId" : 3
},
{
    "sourceId" : 3,
    "destinationId" : 4
}
]
}

```

The `GetWorkflowRun` response contains the following.

```

{
    Nodes : [
        {
            "type" : Trigger,
            "name" : "t1",
            "uniqueId" : 1,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j1",
            "uniqueId" : 2,
            "jobDetails" : [
                {
                    "id" : "jr_12334",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        },
        {
            "type" : Trigger,
            "name" : "t2",
            "uniqueId" : 3,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j2",
            "uniqueId" : 4,
            "jobDetails" : [
                {
                    "id" : "jr_1233sdf4",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        }
    ],
    Edges : [

```

```
{  
    "sourceId" : 1,  
    "destinationId" : 2  
,  
{  
    "sourceId" : 2,  
    "destinationId" : 3  
,  
{  
    "sourceId" : 3,  
    "destinationId" : 4  
}  
}
```

Example 2: Multiple Jobs with a Conditional Trigger

The following example shows a workflow with multiple jobs and a conditional trigger (t3).

```
Consider Flow:  
T(t1) ---> J(j1) ---> T(t2) ---> J(j2)  
|           |           |  
|           |           |  
>+-----> T(t3) <-----+  
|           |  
|           |  
J(j3)  
  
Graph generated:  
Nodes: t1, t2, t3, j1, j2, j3  
Edges: (t1, j1), (j1, t2), (t2, j2), (j1, t3), (j2, t3), (t3, j3)
```

Programming ETL Scripts

AWS Glue makes it easy to write or autogenerate extract, transform, and load (ETL) scripts, in addition to testing and running them. This section describes the extensions to Apache Spark that AWS Glue has introduced, and provides examples of how to code and run ETL scripts in Python and Scala.

Topics

- [General Information about Programming AWS Glue ETL Scripts \(p. 292\)](#)
- [Program AWS Glue ETL Scripts in Python \(p. 314\)](#)
- [Programming AWS Glue ETL Scripts in Scala \(p. 387\)](#)

General Information about Programming AWS Glue ETL Scripts

The following sections describe techniques and values that apply generally to AWS Glue ETL (extract, transform, and load) programming in any language.

Topics

- [Special Parameters Used by AWS Glue \(p. 292\)](#)
- [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#)
- [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#)
- [Managing Partitions for ETL Output in AWS Glue \(p. 302\)](#)
- [Reading Input Files in Larger Groups \(p. 303\)](#)
- [Reading from JDBC Tables in Parallel \(p. 304\)](#)
- [Moving Data to and from Amazon Redshift \(p. 305\)](#)
- [AWS Glue Data Catalog Support for Spark SQL Jobs \(p. 306\)](#)
- [Excluding Amazon S3 Storage Classes \(p. 308\)](#)
- [Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library \(p. 310\)](#)

Special Parameters Used by AWS Glue

AWS Glue recognizes and uses several argument names that you can use to set up the script environment for your jobs and job runs:

- `--job-language` — The script programming language. This must be either `scala` or `python`. If this parameter is not present, the default is `python`.
- `--class` — The Scala class that serves as the entry point for your Scala script. This only applies if your `--job-language` is set to `scala`.
- `--scriptLocation` — The Amazon Simple Storage Service (Amazon S3) location where your ETL script is located (in a form like `s3://path/to/my/script.py`). This overrides a script location set in the `JobCommand` object.
- `--extra-py-files` — The Amazon S3 paths to additional Python modules that AWS Glue adds to the Python path before executing your script. Multiple values must be complete paths separated

by a comma (,). Only individual files are supported, not a directory path. Currently, only pure Python modules work. Extension modules written in C or other languages are not supported.

- **--extra-jars** — The Amazon S3 paths to additional Java .jar files that AWS Glue adds to the Java classpath before executing your script. Multiple values must be complete paths separated by a comma (,).
- **--extra-files** — The Amazon S3 paths to additional files such as configuration files that AWS Glue copies to the working directory of your script before executing it. Multiple values must be complete paths separated by a comma (,). Only individual files are supported, not a directory path.
- **--job-bookmark-option** — Controls the behavior of a job bookmark. The following option values can be set.

Description
<p>enable</p> <p>Keep track of previously processed data. When a job runs, process new data since the last bookmark-point.</p>
<p>disable</p> <p>Always process the entire dataset. You are responsible for managing the output from previous job bookmark-</p>
<p>from</p> <p>Process incremental data since the last successful run or the data in the range identified by the following suboptions, without updating the state of last bookmark. You are responsible for managing the output from previous job runs. The following are the two suboptions:</p> <ul style="list-style-type: none"> • job-bookmark-from <from-value> is the run ID that represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input is ignored. • job-bookmark-to <to-value> is the run ID that represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input excluding the input identified by the <from-value> is processed by the job. Any input later than this input is also excluded for processing. <p>The job bookmark state is not updated when this option set is specified.</p> <p>The suboptions are optional. However, when used, both suboptions must be provided.</p>

For example, to enable a job bookmark, pass the following argument.

```
'--job-bookmark-option': 'job-bookmark-enable'
```

- **--TempDir** — Specifies an Amazon S3 path to a bucket that can be used as a temporary directory for the Job.

For example, to set a temporary directory, pass the following argument.

```
'--TempDir': 's3-path-to-directory'
```

- **--enable-metrics** — Enables the collection of metrics for job profiling for this job run. These metrics are available on the AWS Glue console and the Amazon CloudWatch console. To enable metrics, only specify the key; no value is needed.
- **--enable-glue-datacatalog** — Enables you to use the AWS Glue Data Catalog as an Apache Spark Hive metastore.
- **--enable-continuous-cloudwatch-log** — Enables real-time, continuous logging for AWS Glue jobs. You can view real-time Apache Spark job logs in CloudWatch.

- `--enable-continuous-log-filter` — Specifies a standard filter (`true`) or no filter (`false`) when you create or edit a job enabled for continuous logging. Choosing the standard filter prunes out non-useful Apache Spark driver/executor and Apache Hadoop YARN heartbeat log messages. Choosing no filter gives you all the log messages.
- `--continuous-log-logGroup` — Specifies a custom CloudWatch log group name for a job enabled for continuous logging.
- `--continuous-log-logStreamPrefix` — Specifies a custom CloudWatch log stream prefix for a job enabled for continuous logging.
- `--continuous-log-conversionPattern` — Specifies a custom conversion log pattern for a job enabled for continuous logging. The conversion pattern only applies to driver logs and executor logs. It does not affect the AWS Glue progress bar.

For example, the following is the syntax for running a job with a `--argument` and a special parameter.

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://my_glue/libraries/test_lib.py"'
```

The following are several argument names that AWS Glue uses internally that you should never set:

- `--conf` — Internal to AWS Glue. Do not set.
- `--debug` — Internal to AWS Glue. Do not set.
- `--mode` — Internal to AWS Glue. Do not set.
- `--JOB_NAME` — Internal to AWS Glue. Do not set.

Connection Types and Options for ETL in AWS Glue

Various AWS Glue PySpark and Scala methods and transforms specify connection parameters using a `connectionType` parameter and a `connectionOptions` parameter.

The `connectionType` parameter can take the following values, and the associated `connectionOptions` parameter values for each type are documented following.

In general, these are for ETL input and do not apply to ETL sinks.

- "`connectionType": "s3`" ([p. 295](#)) — Designates a connection to Amazon Simple Storage Service (Amazon S3).
- "`connectionType": "parquet`" ([p. 295](#)) — Designates a connection to files stored in Amazon S3 in the Apache Parquet file format.
- "`connectionType": "orc`" ([p. 295](#)) — Designates a connection to files stored in Amazon S3 in the Apache Hive Optimized Row Columnar (ORC) file format.
- "`connectionType": "mysql`" ([p. 296](#)) — Designates a connection to a MySQL database (see [JDBC connectionType Values \(p. 296\)](#)).
- "`connectionType": "redshift`" ([p. 296](#)) — Designates a connection to an Amazon Redshift database (see [JDBC connectionType Values \(p. 296\)](#)).
- "`connectionType": "oracle`" ([p. 296](#)): Designates a connection to an Oracle database (see [JDBC connectionType Values \(p. 296\)](#)).
- "`connectionType": "sqlserver`" ([p. 296](#)) — Designates a connection to a Microsoft SQL Server database (see [JDBC connectionType Values \(p. 296\)](#)).
- "`connectionType": "postgresql`" ([p. 296](#)) — Designates a connection to a PostgreSQL database (see [JDBC connectionType Values \(p. 296\)](#)).

- "connectionType": "dynamodb" ([p. 296](#)) — Designates a connection to Amazon DynamoDB.

"connectionType": "s3"

Designates a connection to Amazon Simple Storage Service (Amazon S3).

Use the following `connectionOptions` with "connectionType": "s3":

- "paths" — (Required) A list of the Amazon S3 paths from which to read.
- "exclusions" — (Optional) A string containing a JSON list of Unix-style glob patterns to exclude. For example "[**.pdf]" would exclude all pdf files. More information about the glob syntax supported by AWS Glue can be found at [Using Include and Exclude Patterns](#).
- "compressionType": or "compression" — (Optional) Specifies how the data is compressed. Use "compressionType" for Amazon S3 sources and "compression" for Amazon S3 targets. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip".
- "groupFiles" — (Optional) Grouping files is enabled by default when the input contains more than 50,000 files. To enable grouping with fewer than 50,000 files, set this parameter to "inPartition". To disable grouping when there are more than 50,000 files, set this parameter to "none".
- "groupSize" — (Optional) The target group size in bytes. The default is computed based on the input data size and the size of your cluster. When there are fewer than 50,000 input files, "groupFiles" must be set to "inPartition" for this to take effect.
- "recurse" — (Optional) If set to true, recursively reads files in all subdirectories under the specified paths.
- "maxBand" — (Optional, Advanced) This option controls the duration in seconds after which s3 listing is likely to be consistent. Files with modification timestamps falling within the last `maxBand` seconds are tracked specially when using JobBookmarks to account for S3 eventual consistency. Most users do not need to set this option. The default is 900 seconds.
- "maxFilesInBand" — (Optional, Advanced) This option specifies the maximum number of files to save from the last `maxBand` seconds. If this number is exceeded, extra files are skipped and only processed in the next job run. Most users do not need to set this option.

"connectionType": "parquet"

Designates a connection to files stored in Amazon Simple Storage Service (Amazon S3) in the [Apache Parquet](#) file format.

Use the following `connectionOptions` with "connectionType": "parquet":

- `paths` — (Required) A list of the Amazon S3 paths from which to read.
- (*Other option name/value pairs*) — Any additional options, including formatting options, are passed directly to the SparkSQL DataSource. For more information, see [Redshift data source for Spark](#).

"connectionType": "orc"

Designates a connection to files stored in Amazon S3 in the [Apache Hive Optimized Row Columnar \(ORC\)](#) file format.

Use the following `connectionOptions` with "connectionType": "orc":

- `paths` — (Required) A list of the Amazon S3 paths from which to read.

- (*Other option name/value pairs*) — Any additional options, including formatting options, are passed directly to the SparkSQL DataSource. For more information, see [Redshift data source for Spark](#).

"connectionType": "dynamodb"

Designates a connection to Amazon DynamoDB.

Note

AWS Glue does not currently support writing to Amazon DynamoDB.

Use the following `connectionOptions` with "connectionType": "dynamodb":

- "dynamodb.input.tableName" — (Required) The DynamoDB table from which to read.
- "dynamodb.throughput.read.percent" — (Optional) The percentage of read capacity units (RCU) to use. The default is set to "0.5". Acceptable values are from "0.1" to "1.5", inclusive.
- "dynamodb.splits" — (Optional) This parameter defines how many splits we partition this DynamoDB table into while reading. The default is set to "1". Acceptable values are from "1" to "1,000,000", inclusive.

JDBC connectionType Values

These include the following:

- "connectionType": "sqlserver" — Designates a connection to a Microsoft SQL Server database.
- "connectionType": "mysql" — Designates a connection to a [MySQL](#) database.
- "connectionType": "oracle" — Designates a connection to an Oracle database.
- "connectionType": "postgresql" — Designates a connection to a [PostgreSQL](#) database.
- "connectionType": "redshift" — Designates a connection to an [Amazon Redshift](#) database.

The following table lists the JDBC driver versions that AWS Glue supports.

Product	JDBC Driver Version
Microsoft SQL Server	6.x
MySQL	5.1
Oracle Database	11.2
PostgreSQL	42.x
Amazon Redshift	4.1

Use these connection options with JDBC connections:

- "url" — (Required) The JDBC URL for the database.
- "dbtable" — The database table to read from. For JDBC data stores that support schemas within a database, specify `schema.table-name`. If a schema is not provided, then the default "public" schema is used.
- "redshiftTmpDir" — (Required for Amazon Redshift, optional for other JDBC types) The Amazon S3 path where temporary data can be staged when copying out of the database.
- "user": (Required) The username to use when connecting.

- "password" — (Required) The password to use when connecting.
- "jobBookmarkKeys" — (Optional) A comma-delimited list of columns to use as the bookmark keys. Takes effect only if bookmarks are enabled for the job.
- "jobBookmarksKeysSortOrder" — (Optional) Sort order for bookmark keys. Permitted values are "asc" and "desc". When "asc", rows with values greater than bookmarked values are identified as new rows. When "desc", rows with values less than bookmarked values are identified as new rows. Takes effect only if bookmarks are enabled for the job.
- (Optional) The following options enable you to supply a custom JDBC driver. Use these options if you must use a driver that AWS Glue does not natively support. ETL jobs can use different JDBC driver versions for the data source and target, even if the source and target are the same database product. This enables you to migrate data between source and target databases with different versions. To use these options, you must first upload the jar file of the JDBC driver to Amazon S3.
 - "customJdbcDriverS3Path" — Amazon S3 path of the custom JDBC driver.
 - "customJdbcDriverClassName" — Class name of JDBC driver.

All other option name/value pairs that are included in connection options for a JDBC connection, including formatting options, are passed directly to the underlying SparkSQL DataSource. For more information, see [Redshift data source for Spark](#).

The following code examples show how to read from and write to JDBC databases with custom JDBC drivers. They demonstrate reading from one version of a database product and writing to a later version of the same product.

Python

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Construct JDBC connection options
connection_mysql5_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_mysql8_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd",
    "customJdbcDriverS3Path": "s3://path/mysql-connector-java-8.0.17.jar",
    "customJdbcDriverClassName": "com.mysql.cj.jdbc.Driver"}

connection_oracle11_options = {
    "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_oracle18_options = {
```

```

"url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
"dbtable": "test",
"user": "admin",
"password": "pwd",
"customJdbcDriverS3Path": "s3://path/ojdbc10.jar",
"customJdbcDriverClassName": "oracle.jdbc.OracleDriver"}
```

```

# Read from JDBC databases with custom driver
df_mysql8 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",
    connection_options=connection_mysql8_options)
```

```

# Read DynamicFrame from MySQL 5 and write to MySQL 8
df_mysql5 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",
    connection_options=connection_mysql5_options)
glueContext.write_from_options(frame_or_dfc=df_mysql5, connection_type="mysql",
    connection_options=connection_mysql8_options)
```

```

# Read DynamicFrame from Oracle 11 and write to Oracle 18
df_oracle11 = glueContext.create_dynamic_frame.from_options(connection_type="oracle",
    connection_options=connection_oracle11_options)
glueContext.write_from_options(frame_or_dfc=df_oracle11, connection_type="oracle",
    connection_options=connection_oracle18_options)
```

Scala

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
    val MYSQL_5_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
    val MYSQL_8_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
    val ORACLE_11_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"
    val ORACLE_18_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"

    // Construct JDBC connection options
    lazy val mysql5JsonOption = jsonOptions(MYSQL_5_URI)
    lazy val mysql8JsonOption = customJDBCJsonOptions(MYSQL_8_URI, "s3://path/
    mysql-connector-java-8.0.17.jar", "com.mysql.cj.jdbc.Driver")
    lazy val oracle11JsonOption = jsonOptions(ORACLE_11_URI)
    lazy val oracle18JsonOption = customJDBCJsonOptions(ORACLE_18_URI, "s3://path/
    ojdbc10.jar", "oracle.jdbc.OracleDriver")

    def main(sysArgs: Array[String]): Unit = {
        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        // Read from JDBC database with custom driver
        val df_mysql8: DynamicFrame = glueContext.getSource("mysql",
            mysql8JsonOption).getDynamicFrame()

        // Read DynamicFrame from MySQL 5 and write to MySQL 8
```

```
    val df_mysql5: DynamicFrame = glueContext.getDataSource("mysql",
mysql5JsonOption).getDynamicFrame()
    glueContext.getSink("mysql", mysql8JsonOption).writeDynamicFrame(df_mysql5)

    // Read DynamicFrame from Oracle 11 and write to Oracle 18
    val df_oracle11: DynamicFrame = glueContext.getDataSource("oracle",
oracle11JsonOption).getDynamicFrame()
    glueContext.getSink("oracle", oracle18JsonOption).writeDynamicFrame(df_oracle11)

    Job.commit()
}

private def jsonOptions(uri: String): JsonOptions = {
    new JsonOptions(
        s"""{"url": "${uri}",
           |"dbtable":"test",
           |"user": "admin",
           |"password": "pwd"}""".stripMargin)
}

private def customJDBCDriverJsonOptions(uri: String, customJdbcDriverS3Path: String,
customJdbcDriverClassName: String): JsonOptions = {
    new JsonOptions(
        s"""{"url": "${uri}",
           |"dbtable":"test",
           |"user": "admin",
           |"password": "pwd",
           |"customJdbcDriverS3Path": "${customJdbcDriverS3Path}",
           |"customJdbcDriverClassName" : "${customJdbcDriverClassName}"}""".stripMargin)
}
```

Format Options for ETL Inputs and Outputs in AWS Glue

Various AWS Glue PySpark and Scala methods and transforms specify their input and/or output format using a `format` parameter and a `format_options` parameter. These parameters can take the following values.

format="avro"

This value designates the [Apache Avro](#) data format.

There are no `format_options` values for `format="avro"`.

format="csv"

This value designates comma-separated-values as the data format (for example, see [RFC 4180](#) and [RFC 7111](#)).

You can use the following `format_options` values with `format="csv"`:

- `separator` — Specifies the delimiter character. The default is a comma: ',', but any other character can be specified.
- `escaper` — Specifies a character to use for escaping. The default value is "none". If enabled, the character which immediately follows is used as-is, except for a small set of well-known escapes (\n, \r, \t, and \o).

- `quoteChar` — Specifies the character to use for quoting. The default is a double quote: `'"`. Set this to `'-1'` to disable quoting entirely.
- `multiline` — A Boolean value that specifies whether a single record can span multiple lines. This can occur when a field contains a quoted new-line character. You must set this option to `"true"` if any record spans multiple lines. The default value is `"false"`, which allows for more aggressive file-splitting during parsing.
- `withHeader` — A Boolean value that specifies whether to treat the first line as a header. The default value is `"false"`. This option can be used in the `DynamicFrameReader` class.
- `writeHeader` — A Boolean value that specifies whether to write the header to output. The default value is `"true"`. This option can be used in the `DynamicFrameWriter` class.
- `skipFirst` — A Boolean value that specifies whether to skip the first data line. The default value is `"false"`.

format="ion"

This value designates [Amazon Ion](#) as the data format. (For more information, see the [Amazon Ion Specification](#).)

Currently, AWS Glue does not support ion for output.

There are no `format_options` values for `format="ion"`.

format="grokLog"

This value designates a log data format specified by one or more Logstash grok patterns (for example, see [Logstash Reference \(6.2\): Grok filter plugin](#)).

Currently, AWS Glue does not support groklog for output.

You can use the following `format_options` values with `format="grokLog"`:

- `logFormat` — Specifies the grok pattern that matches the log's format.
- `customPatterns` — Specifies additional grok patterns used here.
- `MISSING` — Specifies the signal to use in identifying missing values. The default is `'-'`.
- `LineCount` — Specifies the number of lines in each log record. The default is `'1'`, and currently only single-line records are supported.
- `StrictMode` — A Boolean value that specifies whether strict mode is enabled. In strict mode, the reader doesn't do automatic type conversion or recovery. The default value is `"false"`.

format="json"

This value designates a [JSON](#) (JavaScript Object Notation) data format.

Currently, AWS Glue does not support `format_options` for json output.

You can use the following `format_options` values with `format="json"`:

- `jsonPath` — A [JsonPath](#) expression that identifies an object to be read into records. This is particularly useful when a file contains records nested inside an outer array. For example, the following JsonPath expression targets the `id` field of a JSON object.

```
format="json", format_options={"jsonPath": "$.id"}
```

- `multiline` — A Boolean value that specifies whether a single record can span multiple lines. This can occur when a field contains a quoted new-line character. You must set this option to "true" if any record spans multiple lines. The default value is "false", which allows for more aggressive file-splitting during parsing.

format="orc"

This value designates [Apache ORC](#) as the data format. (For more information, see the [LanguageManual ORC](#).)

There are no `format_options` values for `format="orc"`. However, any options that are accepted by the underlying SparkSQL code can be passed to it by way of the `connection_options` map parameter.

format="parquet"

This value designates [Apache Parquet](#) as the data format.

There are no `format_options` values for `format="parquet"`. However, any options that are accepted by the underlying SparkSQL code can be passed to it by way of the `connection_options` map parameter.

format="glueparquet"

This value designates a custom Parquet writer type that is optimized for Dynamic Frames as the data format. A precomputed schema is not required before writing. As data comes in, `glueparquet` computes and modifies the schema dynamically.

You can use the following `format_options` values with `format="glueparquet"`:

- `compression` — Specifies the compression codec used when writing Parquet files. The default value is "snappy".
- `blockSize` — Specifies the size of a row group being buffered in memory. The default value is "128MB".
- `pageSize` — Specifies the size of the smallest unit that must be read fully to access a single record. The default value is "1MB".

Limitations:

- `glueparquet` supports only a schema shrinkage or expansion, but not a type change.
- `glueparquet` is not able to store a schema-only file.

format="xml"

This value designates XML as the data format, parsed through a fork of the [XML Data Source for Apache Spark](#) parser.

Currently, AWS Glue does not support "xml" for output.

You can use the following `format_options` values with `format="xml"`:

- `rowTag` — Specifies the XML tag in the file to treat as a row. Row tags cannot be self-closing.
- `encoding` — Specifies the character encoding. The default value is "UTF-8".
- `excludeAttribute` — A Boolean value that specifies whether you want to exclude attributes in elements or not. The default value is "false".

- `treatEmptyValuesAsNulls` — A Boolean value that specifies whether to treat white space as a null value. The default value is "false".
- `attributePrefix` — A prefix for attributes to differentiate them from elements. This prefix is used for field names. The default value is "_".
- `valueTag` — The tag used for a value when there are attributes in the element that have no child. The default is "_VALUE".
- `ignoreSurroundingSpaces` — A Boolean value that specifies whether the white space that surrounds values should be ignored. The default value is "false".

Managing Partitions for ETL Output in AWS Glue

Partitioning is an important technique for organizing datasets so they can be queried efficiently. It organizes data in a hierarchical directory structure based on the distinct values of one or more columns.

For example, you might decide to partition your application logs in Amazon Simple Storage Service (Amazon S3) by date, broken down by year, month, and day. Files that correspond to a single day's worth of data are then placed under a prefix such as `s3://my_bucket/logs/year=2018/month=01/day=23/`. Systems like Amazon Athena, Amazon Redshift Spectrum, and now AWS Glue can use these partitions to filter data by partition value without having to read all the underlying data from Amazon S3.

Crawlers not only infer file types and schemas, they also automatically identify the partition structure of your dataset when they populate the AWS Glue Data Catalog. The resulting partition columns are available for querying in AWS Glue ETL jobs or query engines like Amazon Athena.

After you crawl a table, you can view the partitions that the crawler created by navigating to the table on the AWS Glue console and choosing **View Partitions**.

For Apache Hive-style partitioned paths in `key=val` style, crawlers automatically populate the column name using the key name. Otherwise, it uses default names like `partition_0`, `partition_1`, and so on. To change the default names on the console, navigate to the table, choose **Edit Schema**, and modify the names of the partition columns there.

In your ETL scripts, you can then filter on the partition columns.

Pre-Filtering Using Pushdown Predicates

In many cases, you can use a pushdown predicate to filter on partitions without having to list and read all the files in your dataset. Instead of reading the entire dataset and then filtering in a DynamicFrame, you can apply the filter directly on the partition metadata in the Data Catalog. Then you only list and read what you actually need into a DynamicFrame.

For example, in Python, you could write the following.

```
glue_context.create_dynamic_frame.from_catalog(
    database = "my_S3_data_set",
    table_name = "catalog_data_table",
    push_down_predicate = my_partition_predicate)
```

This creates a DynamicFrame that loads only the partitions in the Data Catalog that satisfy the predicate expression. Depending on how small a subset of your data you are loading, this can save a great deal of processing time.

The predicate expression can be any Boolean expression supported by Spark SQL. Anything you could put in a `WHERE` clause in a Spark SQL query will work. For example, the predicate expression `pushDownPredicate = "(year=='2017' and month=='04')"` loads only the partitions in the

Data Catalog that have both `year` equal to 2017 and `month` equal to 04. For more information, see the [Apache Spark SQL documentation](#), and in particular, the [Scala SQL functions reference](#).

In addition to Hive-style partitioning for Amazon S3 paths, Apache Parquet and Apache ORC file formats further partition each file into blocks of data that represent column values. Each block also stores statistics for the records that it contains, such as min/max for column values. AWS Glue supports pushdown predicates for both Hive-style partitions and block partitions in these formats. In this way, you can prune unnecessary Amazon S3 partitions in Parquet and ORC formats, and skip blocks that you determine are unnecessary using column statistics.

Writing Partitions

By default, a `DynamicFrame` is not partitioned when it is written. All of the output files are written at the top level of the specified output path. Until recently, the only way to write a `DynamicFrame` into partitions was to convert it to a `Spark SQL DataFrame` before writing.

However, `DynamicFrames` now support native partitioning using a sequence of keys, using the `partitionKeys` option when you create a sink. For example, the following Python code writes out a dataset to Amazon S3 in the Parquet format, into directories partitioned by the `type` field. From there, you can process these partitions using other systems, such as Amazon Athena.

```
glue_context.write_dynamic_frame.from_options(  
    frame = projectedEvents,  
    connection_type = "s3",  
    connection_options = {"path": "$outpath", "partitionKeys": ["type"]},  
    format = "parquet")
```

Reading Input Files in Larger Groups

You can set properties of your tables to enable an AWS Glue ETL job to group files when they are read from an Amazon S3 data store. These properties enable each ETL task to read a group of input files into a single in-memory partition, this is especially useful when there is a large number of small files in your Amazon S3 data store. When you set certain properties, you instruct AWS Glue to group files within an Amazon S3 data partition and set the size of the groups to be read. You can also set these options when reading from an Amazon S3 data store with the `create_dynamic_frame_from_options` method.

To enable grouping files for a table, you set key-value pairs in the `parameters` field of your table structure. Use JSON notation to set a value for the `parameter` field of your table. For more information about editing the properties of a table, see [Viewing and Editing Table Details \(p. 109\)](#).

You can use this method to enable grouping for tables in the Data Catalog with Amazon S3 data stores.

groupFiles

Set `groupFiles` to `inPartition` to enable the grouping of files within an Amazon S3 data partition. AWS Glue automatically enables grouping if there are more than 50,000 input files, as in the following example.

```
'groupFiles': 'inPartition'
```

groupSize

Set `groupSize` to the target size of groups in bytes. The `groupSize` property is optional, if not provided, AWS Glue calculates a size to use all the CPU cores in the cluster while still reducing the overall number of ETL tasks and in-memory partitions.

For example, the following sets the group size to 1 MB.

```
'groupSize': '1048576'
```

Note that the `groupsize` should be set with the result of a calculation. For example $1024 * 1024 = 1048576$.

recurse

Set `recurse` to `True` to recursively read files in all subdirectories when specifying `paths` as an array of paths. You do not need to set `recurse` if `paths` is an array of object keys in Amazon S3, as in the following example.

```
'recurse':True
```

If you are reading from Amazon S3 directly using the `create_dynamic_frame_from_options` method, add these connection options. For example, the following attempts to group files into 1 MB groups.

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://s3path/"],  
'recurse':True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```

Reading from JDBC Tables in Parallel

You can set properties of your JDBC table to enable AWS Glue to read data in parallel. When you set certain properties, you instruct AWS Glue to run parallel SQL queries against logical partitions of your data. You can control partitioning by setting a hash field or a hash expression. You can also control the number of parallel reads that are used to access your data.

To enable parallel reads, you can set key-value pairs in the `parameters` field of your table structure. Use JSON notation to set a value for the `parameter` field of your table. For more information about editing the properties of a table, see [Viewing and Editing Table Details \(p. 109\)](#). You can also enable parallel reads when you call the ETL (extract, transform, and load) methods `create_dynamic_frame_from_options` and `create_dynamic_frame_from_catalog`. For more information about specifying options in these methods, see [from_options \(p. 351\)](#) and [from_catalog \(p. 351\)](#).

You can use this method for JDBC tables, that is, most tables whose base data is a JDBC data store. These properties are ignored when reading Amazon Redshift and Amazon S3 tables.

hashfield

Set `hashfield` to the name of a column in the JDBC table to be used to divide the data into partitions. For best results, this column should have an even distribution of values to spread the data between partitions. This column can be of any data type. AWS Glue generates non-overlapping queries that run in parallel to read the data partitioned by this column. For example, if your data is evenly distributed by month, you can use the `month` column to read each month of data in parallel.

```
'hashfield': 'month'
```

AWS Glue creates a query to hash the field value to a partition number and runs the query for all partitions in parallel. To use your own query to partition a table read, provide a `hashexpression` instead of a `hashfield`.

hashexpression

Set `hashexpression` to an SQL expression (conforming to the JDBC database engine grammar) that returns a whole number. A simple expression is the name of any numeric column in the table. AWS Glue generates SQL queries to read the JDBC data in parallel using the `hashexpression` in the `WHERE` clause to partition data.

For example, use the numeric column `customerID` to read data partitioned by a customer number.

```
'hashexpression': 'customerID'
```

To have AWS Glue control the partitioning, provide a `hashfield` instead of a `hashexpression`.

hashpartitions

Set `hashpartitions` to the number of parallel reads of the JDBC table. If this property is not set, the default value is 7.

For example, set the number of parallel reads to 5 so that AWS Glue reads your data with five queries (or fewer).

```
'hashpartitions': '5'
```

Moving Data to and from Amazon Redshift

When moving data to and from an Amazon Redshift cluster, AWS Glue jobs issue COPY and UNLOAD statements against Amazon Redshift to achieve maximum throughput. These commands require that the Amazon Redshift cluster access Amazon Simple Storage Service (Amazon S3) as a staging directory. By default, AWS Glue passes in temporary credentials that are created using the role that you specified to run the job. For security purposes, these credentials expire after 1 hour, which can cause long running jobs to fail.

To address this issue, you can associate one or more IAM roles with the Amazon Redshift cluster itself. COPY and UNLOAD can use the role, and Amazon Redshift refreshes the credentials as needed. For more information about associating a role with your Amazon Redshift cluster, see [IAM Permissions for COPY, UNLOAD, and CREATE LIBRARY](#) in the *Amazon Redshift Database Developer Guide*. Make sure that the role you associate with your cluster has permissions to read from and write to the Amazon S3 temporary directory that you specified in your job.

After you set up a role for the cluster, you need to specify it in ETL (extract, transform, and load) statements in the AWS Glue script. The syntax depends on how your script reads and writes your dynamic frame. If your script reads from an AWS Glue Data Catalog table, you can specify a role as follows.

```
glueContext.create_dynamic_frame.from_catalog(  
    database = "database-name",  
    table_name = "table-name",  
    redshift_tmp_dir = args["TempDir"],  
    additional_options = {"aws_iam_role": "arn:aws:iam::account-id:role/role-name"})
```

Similarly, if your scripts writes a dynamic frame and reads from an Data Catalog, you can specify the role as follows.

```
glueContext.write_dynamic_frame.from_catalog(
    database = "database-name",
    table_name = "table-name",
    redshift_tmp_dir = args["TempDir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::account-id:role/role-name"})
```

In these examples, *role-name* is the role that you associated with your Amazon Redshift cluster, and *database-name* and *table-name* refer to an Amazon Redshift table in your Data Catalog.

You can also specify a role when you use a dynamic frame and you use `copy_from_options`. The syntax is similar, but you put the additional parameter in the *connection-options* map.

```
connection_options = {
    "url": "jdbc:redshift://host:port/redshift-database",
    "dbtable": "redshift-table",
    "user": "username",
    "password": "password",
    "redshiftTmpDir": args["TempDir"],
    "aws_iam_role": "arn:aws:iam::account-id:role/role-name"
}

df = glueContext.create_dynamic_frame_from_options("redshift", connection_options)
```

The options are similar when writing to Amazon Redshift.

```
connection_options = {
    "dbtable": "redshift-table",
    "database": "redshift-database",
    "aws_iam_role": "arn:aws:iam::account-id:role/role-name"
}

glueContext.write_dynamic_frame.from_jdbc_conf(
    frame = input-dynamic-frame,
    catalog_connection = "connection-name",
    connection_options = connection-options,
    redshift_tmp_dir = args["TempDir"])
```

AWS Glue Data Catalog Support for Spark SQL Jobs

The AWS Glue Data Catalog is an Apache Hive metastore-compatible catalog. You can configure your AWS Glue jobs and development endpoints to use the Data Catalog as an external Apache Hive metastore. You can then directly run Apache Spark SQL queries against the tables stored in the Data Catalog. AWS Glue dynamic frames integrate with the Data Catalog by default. However, with this feature, Spark SQL jobs can start using the Data Catalog as an external Hive metastore.

You can configure AWS Glue jobs and development endpoints by adding the `--enable-glue-datacatalog`: `"` argument to job arguments and development endpoint arguments respectively. Passing this argument sets certain configurations in Spark that enable it to access the Data Catalog as an external Hive metastore. It also enables [Hive support](#) in the `SparkSession` object created in the AWS Glue job or development endpoint.

To enable the Data Catalog access, check the **Use Glue Data Catalog as the Hive metastore** check box in the **Catalog options** group on the **Add job** or **Add endpoint** page on the console. Note that the IAM role used for the job or development endpoint should have `glue:CreateDatabase` permissions. A database called "default" is created in the Data Catalog if it does not exist.

Lets look at an example of how you can use this feature in your Spark SQL jobs. The following example assumes that you have crawled the US legislators dataset available at `s3://awsglue-datasets/examples/us-legislators`.

To serialize/deserialize data from the tables defined in the Glue Data Catalog, Spark SQL needs the [Hive SerDe](#) class for the format defined in the Glue Data Catalog in the classpath of the spark job.

SerDes for certain common formats are distributed by AWS Glue. The following are the Amazon S3 links for these:

- [JSON](#)
- [XML](#)
- [Grok](#)

Add the JSON SerDe as an [extra JAR to the development endpoint](#). For jobs, you can add the SerDe using the `--extra-jars` argument in the arguments field. For more information, see [Special Parameters Used by AWS Glue \(p. 292\)](#).

Here is an example input JSON to create a development endpoint with the Data Catalog enabled for Spark SQL.

```
{
  "EndpointName": "Name",
  "RoleArn": "role_ARN",
  "PublicKey": "public_key_contents",
  "NumberOfNodes": 2,
  "Arguments": {
    "--enable-glue-datacatalog": ""
  },
  "ExtraJarsS3Path": "s3://crawler-public/json/serde/json-serde.jar"
}
```

Now query the tables created from the US legislators dataset using Spark SQL.

```
>>> spark.sql("use legislators")
DataFrame[]
>>> spark.sql("show tables").show()
+-----+-----+
| database |      tableName | isTemporary |
+-----+-----+
| legislators | areas_json |   false |
| legislators | countries_json |   false |
| legislators | events_json |   false |
| legislators | memberships_json |   false |
| legislators | organizations_json |   false |
| legislators | persons_json |   false |
+-----+-----+
>>> spark.sql("describe memberships_json").show()
+-----+-----+
|       col_name | data_type |      comment |
+-----+-----+
|           area_id |   string | from deserializer |
| on_behalf_of_id |   string | from deserializer |
```

```

|   organization_id| string|from deserializer|
|       role| string|from deserializer|
|   person_id| string|from deserializer|
|legislative_perio...| string|from deserializer|
|       start_date| string|from deserializer|
|       end_date| string|from deserializer|
+-----+-----+-----+

```

If the SerDe class for the format is not available in the job's classpath, you will see an error similar to the following.

```

>>> spark.sql("describe memberships_json").show()

Caused by: MetaException(message:java.lang.ClassNotFoundException Class
org.openx.data.jsonserde.JsonSerDe not found)
    at
org.apache.hadoop.hive.metastore.MetaStoreUtils.getDeserializer(MetaStoreUtils.java:399)
    at
org.apache.hadoop.hive.ql.metadata.Table.getDeserializerFromMetaStore(Table.java:276)
... 64 more

```

To view only the distinct `organization_ids` from the `memberships` table, execute the following SQL query.

```

>>> spark.sql("select distinct organization_id from memberships_json").show()
+-----+
|   organization_id|
+-----+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+-----+

```

If you need to do the same with dynamic frames, execute the following.

```

>>> memberships = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="memberships_json")
>>> memberships.toDF().createOrReplaceTempView("memberships")
>>> spark.sql("select distinct organization_id from memberships").show()
+-----+
|   organization_id|
+-----+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+-----+

```

While DynamicFrames are optimized for ETL operations, enabling Spark SQL to access the Data Catalog directly provides a concise way to execute complex SQL statements or port existing applications.

Excluding Amazon S3 Storage Classes

If you're running AWS Glue ETL jobs that read files or partitions from Amazon Simple Storage Service (Amazon S3), you can exclude some Amazon S3 storage class types.

The following storage classes are available in Amazon S3:

- **STANDARD** — For general-purpose storage of frequently accessed data.
- **INTELLIGENT_TIERING** — For data with unknown or changing access patterns.
- **STANDARD_IA** and **ONEZONE_IA** — For long-lived, but less frequently accessed data.

- **GLACIER, DEEP_ARCHIVE, and REDUCED_REDUNDANCY** — For long-term archive and digital preservation.

For more information, see [Amazon S3 Storage Classes](#) in the *Amazon S3 Developer Guide*.

The examples in this section show how to exclude the **GLACIER** and **DEEP_ARCHIVE** storage classes. These classes allow you to list files, but they won't let you read the files unless they are restored. (For more information, see [Restoring Archived Objects](#) in the *Amazon S3 Developer Guide*.)

By using storage class exclusions, you can ensure that your AWS Glue jobs will work on tables that have partitions across these storage class tiers. Without exclusions, jobs that read data from these tiers fail with the following error: `AmazonS3Exception: The operation is not valid for the object's storage class.`

There are different ways that you can filter Amazon S3 storage classes in AWS Glue.

Topics

- [Excluding Amazon S3 Storage Classes When Creating a Dynamic Frame \(p. 309\)](#)
- [Excluding Amazon S3 Storage Classes on a Data Catalog Table \(p. 309\)](#)

Excluding Amazon S3 Storage Classes When Creating a Dynamic Frame

To exclude Amazon S3 storage classes while creating a dynamic frame, use `excludeStorageClasses` in `additionalOptions`. AWS Glue automatically uses its own Amazon S3 Lister implementation to list and exclude files corresponding to the specified storage classes.

The following Python and Scala examples show how to exclude the **GLACIER** and **DEEP_ARCHIVE** storage classes when creating a dynamic frame.

Python example:

```
glueContext.create_dynamic_frame.from_catalog(
    database = "my_database",
    tableName = "my_table_name",
    redshift_tmp_dir = "",
    transformation_ctx = "my_transformation_context",
    additional_options = {
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)
```

Scala example:

```
val* *df = glueContext.getCatalogSource(
    nameSpace, tableName, "", "my_transformation_context",
    additionalOptions = JsonOptions(
        Map("excludeStorageClasses" -> List("GLACIER", "DEEP_ARCHIVE"))
    )
).getDynamicFrame()
```

Excluding Amazon S3 Storage Classes on a Data Catalog Table

You can specify storage class exclusions to be used by an AWS Glue ETL job as a table parameter in the AWS Glue Data Catalog. You can include this parameter in the `CreateTable` operation using the AWS Command Line Interface (AWS CLI) or programmatically using the API. For more information, see [Table Structure](#) and [CreateTable](#).

You can also specify excluded storage classes on the AWS Glue console.

To exclude Amazon S3 storage classes (console)

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane on the left, choose **Tables**.
3. Choose the table name in the list, and then choose **Edit table**.
4. In **Table properties**, add `excludeStorageClasses` as a key and `[\"GLACIER\", \"DEEP_ARCHIVE\"]` as a value.
5. Choose **Apply**.

Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library

The AWS Glue Scala library is available in a public Amazon S3 bucket, and can be consumed by the Apache Maven build system. This enables you to develop and test your Python and Scala extract, transform, and load (ETL) scripts locally, without the need for a network connection.

Local development is available for Glue versions 0.9 and 1.0. For information about the versions of Python and Apache Spark that are available with AWS Glue, see the [Glue version job property \(p. 165\)](#).

The library is released with the Amazon Software license (<https://aws.amazon.com/asl>).

Topics

- [Local Development Restrictions \(p. 310\)](#)
- [Developing Locally with Python \(p. 310\)](#)
- [Developing Locally with Scala \(p. 311\)](#)

Local Development Restrictions

Keep the following restrictions in mind when using the AWS Glue Scala library to develop locally.

- Avoid creating an assembly jar ("fat jar" or "uber jar") with the AWS Glue library because it causes the following features to be disabled:
 - [Job bookmarks](#)
 - AWS Glue Parquet writer ([format="glueparquet" \(p. 301\)](#))
 - [FindMatches transform](#)

These feature are available only within the AWS Glue job system.

Developing Locally with Python

Complete some prerequisite steps and then use AWS Glue utilities to test and submit your Python ETL script.

Prerequisites for Local Python Development

Complete these steps to prepare for local Python development:

1. Download the AWS Glue Python library from github (<https://github.com/awslabs/aws-glue-lbs>).

2. Do one of the following:
 - For Glue version 0.9, stay on the `master` branch.
 - For Glue version 1.0, check out branch `glue-1.0`. This version supports Python 3.
3. Install Apache Maven from the following location: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz>.
4. Install the Apache Spark distribution from one of the following locations:
 - For Glue version 0.9: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>
 - For Glue version 1.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
5. Export the `SPARK_HOME` environment variable, setting it to the root location extracted from the Spark archive. For example:
 - For Glue version 0.9: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
 - For Glue version 1.0: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`

Running Your Python ETL Script

With the AWS Glue jar files available for local development, you can run the AWS Glue Python package locally.

Use the following utilities and frameworks to test and run your Python script. The commands listed in the following table are run from the root directory of the [AWS Glue Python package](#).

Utility	Command	Description
Glue Shell	<code>./bin/gluepyspark</code>	Enter and run Python scripts in a shell that integrates with AWS Glue ETL libraries.
Glue Submit	<code>./bin/gluesparksubmit</code>	Submit a complete Python script for execution.
Pytest	<code>./bin/gluepytest</code>	Write and run unit tests of your Python code. The <code>pytest</code> module must be installed and available in the <code>PATH</code> . For more information, see the pytest documentation .

Developing Locally with Scala

Complete some prerequisite steps and then issue a Maven command to run your Scala ETL script locally.

Prerequisites for Local Scala Development

Complete these steps to prepare for local Scala development.

Step 1: Install Software

In this step, you install software and set the required environment variable.

1. Install Apache Maven from the following location: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz>.
2. Install the Apache Spark distribution from one of the following locations:
 - For Glue version 0.9: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>

- For Glue version 1.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
3. Export the `SPARK_HOME` environment variable, setting it to the root location extracted from the Spark archive. For example:
 - For Glue version 0.9: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
 - For Glue version 1.0: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`

Step 2: Configure Your Maven Project

Use the following `pom.xml` file as a template for your AWS Glue Scala applications. It contains the required dependencies, repositories, and plugins elements. Replace the Glue version string with 1.0.0 for Glue version 1.0 or 0.9.0 for Glue version 0.9.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>AWSGlueApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>${project.artifactId}</name>
  <description>AWS Glue ETL application</description>

  <properties>
    <scala.version>2.11.1</scala.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>${scala.version}</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>AWSGlueETL</artifactId>

      <version>Glue version</version>

    </dependency>
  </dependencies>

  <repositories>
    <repository>
      <id>aws-glue-etl-artifacts</id>
      <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/</url>
    </repository>
  </repositories>
  <build>
    <sourceDirectory>src/main/scala</sourceDirectory>
    <plugins>
      <plugin>
        <!-- see http://davidb.github.com/scala-maven-plugin -->
        <groupId>net.alchim31.maven</groupId>
        <artifactId>scala-maven-plugin</artifactId>
        <version>3.4.0</version>
        <executions>
          <execution>
            <goals>
              <goal>compile</goal>
              <goal>testCompile</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

```

        </goals>
    </execution>
</executions>
</plugin>
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>1.6.0</version>
<executions>
    <execution>
        <goals>
            <goal>java</goal>
        </goals>
    </execution>
</executions>
<configuration>
<systemProperties>
    <systemProperty>
        <key>spark.master</key>
        <value>local[*]</value>
    </systemProperty>
    <systemProperty>
        <key>spark.app.name</key>
        <value>localrun</value>
    </systemProperty>
    <systemProperty>
        <key>org.xerial.snappy.lib.name</key>
        <value>libsnappyjava.jnilib</value>
    </systemProperty>
</systemProperties>
</configuration>
</plugin>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-enforcer-plugin</artifactId>
<version>3.0.0-M2</version>
<executions>
    <execution>
        <id>enforce-maven</id>
        <goals>
            <goal>enforce</goal>
        </goals>
        <configuration>
            <rules>
                <requireMavenVersion>
                    <version>3.5.3</version>
                </requireMavenVersion>
            </rules>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

Running Your Scala ETL Script

Run the following command from the Maven project root directory to execute your Scala ETL script.

```
mvn exec:java -Dexec.mainClass="mainClass" -Dexec.args="--JOB-NAME jobName"
```

Replace **mainClass** with the fully qualified class name of the script's main class. Replace **jobName** with the desired job name.

Program AWS Glue ETL Scripts in Python

You can find Python code examples and utilities for AWS Glue in the [AWS Glue samples repository](#) on the GitHub website.

Using Python with AWS Glue

AWS Glue supports an extension of the PySpark Python dialect for scripting extract, transform, and load (ETL) jobs. This section describes how to use Python in ETL scripts and with the AWS Glue API.

- [Setting Up to Use Python with AWS Glue \(p. 315\)](#)
- [Calling AWS Glue APIs in Python \(p. 315\)](#)
- [Using Python Libraries with AWS Glue \(p. 317\)](#)
- [AWS Glue Python Code Samples \(p. 318\)](#)

AWS Glue PySpark Extensions

AWS Glue has created the following extensions to the PySpark Python dialect.

- [Accessing Parameters Using `getResolvedOptions` \(p. 333\)](#)
- [PySpark Extension Types \(p. 333\)](#)
- [DynamicFrame Class \(p. 337\)](#)
- [DynamicFrameCollection Class \(p. 348\)](#)
- [DynamicFrameWriter Class \(p. 349\)](#)
- [DynamicFrameReader Class \(p. 350\)](#)
- [GlueContext Class \(p. 352\)](#)

AWS Glue PySpark Transforms

AWS Glue has created the following transform Classes to use in PySpark ETL operations.

- [GlueTransform Base Class \(p. 360\)](#)
- [ApplyMapping Class \(p. 362\)](#)
- [DropFields Class \(p. 363\)](#)
- [DropNullFields Class \(p. 364\)](#)
- [ErrorsAsDynamicFrame Class \(p. 366\)](#)
- [Filter Class \(p. 367\)](#)
- [Join Class \(p. 370\)](#)
- [Map Class \(p. 371\)](#)
- [MapToCollection Class \(p. 374\)](#)
- [mergeDynamicFrame \(p. 342\)](#)
- [Relationalize Class \(p. 375\)](#)
- [RenameField Class \(p. 376\)](#)
- [ResolveChoice Class \(p. 377\)](#)
- [SelectFields Class \(p. 379\)](#)
- [SelectFromCollection Class \(p. 380\)](#)
- [Spigot Class \(p. 381\)](#)

- [SplitFields Class \(p. 382\)](#)
- [SplitRows Class \(p. 384\)](#)
- [Unbox Class \(p. 385\)](#)
- [UnnestFrame Class \(p. 386\)](#)

Setting Up to Use Python with AWS Glue

Use Python to develop your ETL scripts for Spark jobs. The supported Python versions for ETL jobs depend on the Glue version of the job. For more information on Glue versions, see the [Glue version job property \(p. 165\)](#).

To set up your system for using Python with AWS Glue

Follow these steps to install Python and to be able to invoke the AWS Glue APIs.

1. If you don't already have Python installed, download and install it from the [Python.org download page](#).
2. Install the AWS Command Line Interface (AWS CLI) as documented in the [AWS CLI documentation](#).

The AWS CLI is not directly necessary for using Python. However, installing and configuring it is a convenient way to set up AWS with your account credentials and verify that they work.

3. Install the AWS SDK for Python (Boto 3), as documented in the [Boto3 Quickstart](#).

Boto 3 resource APIs are not yet available for AWS Glue. Currently, only the Boto 3 client APIs can be used.

For more information about Boto 3, see [AWS SDK for Python \(Boto 3\) Getting Started](#).

You can find Python code examples and utilities for AWS Glue in the [AWS Glue samples repository](#) on the GitHub website.

Calling AWS Glue APIs in Python

Note that Boto 3 resource APIs are not yet available for AWS Glue. Currently, only the Boto 3 client APIs can be used.

AWS Glue API Names in Python

AWS Glue API names in Java and other programming languages are generally CamelCased. However, when called from Python, these generic names are changed to lowercase, with the parts of the name separated by underscore characters to make them more "Pythonic". In the [AWS Glue API \(p. 452\)](#) reference documentation, these Pythonic names are listed in parentheses after the generic CamelCased names.

However, although the AWS Glue API names themselves are transformed to lowercase, their parameter names remain capitalized. It is important to remember this, because parameters should be passed by name when calling AWS Glue APIs, as described in the following section.

Passing and Accessing Python Parameters in AWS Glue

In Python calls to AWS Glue APIs, it's best to pass parameters explicitly by name. For example:

```
job = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
```

```
'ScriptLocation': 's3://my_script_bucket/scripts/  
my_etl_script.py'})
```

It is helpful to understand that Python creates a dictionary of the name/value tuples that you specify as arguments to an ETL script in a [Job Structure \(p. 540\)](#) or [JobRun Structure \(p. 551\)](#). Boto 3 then passes them to AWS Glue in JSON format by way of a REST API call. This means that you cannot rely on the order of the arguments when you access them in your script.

For example, suppose that you're starting a JobRun in a Python Lambda handler function, and you want to specify several parameters. Your code might look something like the following:

```
from datetime import datetime, timedelta

client = boto3.client('glue')

def lambda_handler(event, context):
    last_hour_date_time = datetime.now() - timedelta(hours = 1)
    day_partition_value = last_hour_date_time.strftime("%Y-%m-%d")
    hour_partition_value = last_hour_date_time.strftime("%-H")

    response = client.start_job_run(
        JobName = 'my_test_Job',
        Arguments = {
            '--day_partition_key': 'partition_0',
            '--hour_partition_key': 'partition_1',
            '--day_partition_value': day_partition_value,
            '--hour_partition_value': hour_partition_value } )
```

To access these parameters reliably in your ETL script, specify them by name using AWS Glue's `getResolvedOptions` function and then access them from the resulting dictionary:

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
    ['JOB_NAME',
     'day_partition_key',
     'hour_partition_key',
     'day_partition_value',
     'hour_partition_value'])
print "The day partition key is: ", args['day_partition_key']
print "and the day partition value is: ", args['day_partition_value']
```

Example: Create and Run a Job

The following example shows how to call the AWS Glue APIs using Python, to create and run an ETL job.

To create and run a job

1. Create an instance of the AWS Glue client:

```
import boto3
glue = boto3.client(service_name='glue', region_name='us-east-1',
    endpoint_url='https://glue.us-east-1.amazonaws.com')
```

2. Create a job. You must use `glueetl` as the name for the ETL command, as shown in the following code:

```
myJob = glue.create_job(Name='sample', Role='Glue_DefaultRole',
    Command={'Name': 'glueetl',
```

```
'ScriptLocation': 's3://my_script_bucket/scripts/  
my_etl_script.py'})
```

3. Start a new run of the job that you created in the previous step:

```
myNewJobRun = glue.start_job_run(JobName=myJob['Name'])
```

4. Get the job status:

```
status = glue.get_job_run(JobName=myJob['Name'], RunId=JobRun['JobRunId'])
```

5. Print the current state of the job run:

```
print status['JobRun']['JobRunState']
```

Using Python Libraries with AWS Glue

You can use Python extension modules and libraries with your AWS Glue ETL scripts as long as they are written in pure Python. C libraries such as `pandas` are not supported at the present time, nor are extensions written in other languages.

Zipping Libraries for Inclusion

Unless a library is contained in a single `.py` file, it should be packaged in a `.zip` archive. The package directory should be at the root of the archive, and must contain an `__init__.py` file for the package. Python will then be able to import the package in the normal way.

If your library only consists of a single Python module in one `.py` file, you do not need to place it in a `.zip` file.

Loading Python Libraries in a Development Endpoint

If you are using different library sets for different ETL scripts, you can either set up a separate development endpoint for each set, or you can overwrite the library `.zip` file(s) that your development endpoint loads every time you switch scripts.

You can use the console to specify one or more library `.zip` files for a development endpoint when you create it. After assigning a name and an IAM role, choose **Script Libraries and job parameters (optional)** and enter the full Amazon S3 path to your library `.zip` file in the **Python library path** box. For example:

```
s3://bucket/prefix/site-packages.zip
```

If you want, you can specify multiple full paths to files, separating them with commas but no spaces, like this:

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

If you update these `.zip` files later, you can use the console to re-import them into your development endpoint. Navigate to the developer endpoint in question, check the box beside it, and choose **Update ETL libraries** from the **Action** menu.

In a similar way, you can specify library files using the AWS Glue APIs. When you create a development endpoint by calling [CreateDevEndpoint Action \(Python: create_dev_endpoint\) \(p. 583\)](#), you can specify one or more full paths to libraries in the `ExtraPythonLibsS3Path` parameter, in a call that looks this:

```
dep = glue.create_dev_endpoint(
    EndpointName="testDevEndpoint",
    RoleArn="arn:aws:iam::123456789012",
    SecurityGroupIds="sg-7f5ad1ff",
    SubnetId="subnet-c12fdb4",
    PublicKey="ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQOtp04H/y...",
    NumberOfNodes=3,
    ExtraPythonLibsS3Path="s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/
lib_X.zip")
```

When you update a development endpoint, you can also update the libraries it loads using a [DevEndpointCustomLibraries \(p. 583\)](#) object and setting the `UpdateEtlLibraries` parameter to `True` when calling [UpdateDevEndpoint \(update_dev_endpoint\) \(p. 587\)](#).

If you are using a Zeppelin Notebook with your development endpoint, you will need to call the following PySpark function before importing a package or packages from your .zip file:

```
sc.addPyFile("/home/glue/downloads/python/yourZipFileName.zip")
```

Using Python Libraries in a Job or JobRun

When you are creating a new Job on the console, you can specify one or more library .zip files by choosing **Script Libraries and job parameters (optional)** and entering the full Amazon S3 library path(s) in the same way you would when creating a development endpoint:

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

If you are calling [CreateJob \(create_job\) \(p. 545\)](#), you can specify one or more full paths to default libraries using the `--extra-py-files` default parameter, like this:

```
job = glue.create_job(Name='sampleJob',
                      Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
                                'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'},
                      DefaultArguments={'--extra-py-files': 's3://bucket/prefix/
lib_A.zip,s3://bucket_B/prefix/lib_X.zip'})
```

Then when you are starting a JobRun, you can override the default library setting with a different one:

```
runId = glue.start_job_run(JobName='sampleJob',
                           Arguments={'--extra-py-files': 's3://bucket/prefix/lib_B.zip'})
```

AWS Glue Python Code Samples

- [Code Example: Joining and Relationalizing Data \(p. 318\)](#)
- [Code Example: Data Preparation Using ResolveChoice, Lambda, and ApplyMapping \(p. 327\)](#)

Code Example: Joining and Relationalizing Data

This example uses a dataset that was downloaded from <http://everypolitician.org/> to the sample-dataset bucket in Amazon Simple Storage Service (Amazon S3): `s3://awsglue-datasets/examples/us-legislators/all`. The dataset contains data in JSON format about United States legislators and the seats that they have held in the US House of Representatives and Senate, and has been modified slightly and made available in a public Amazon S3 bucket for purposes of this tutorial.

You can find the source code for this example in the `join_and_relationalize.py` file in the [AWS Glue samples repository](#) on the GitHub website.

Using this data, this tutorial shows you how to do the following:

- Use an AWS Glue crawler to classify objects that are stored in a public Amazon S3 bucket and save their schemas into the AWS Glue Data Catalog.
- Examine the table metadata and schemas that result from the crawl.
- Write a Python extract, transfer, and load (ETL) script that uses the metadata in the Data Catalog to do the following:
 - Join the data in the different source files together into a single data table (that is, denormalize the data).
 - Filter the joined table into separate tables by type of legislator.
 - Write out the resulting data to separate Apache Parquet files for later analysis.

The easiest way to debug Python or PySpark scripts is to create a development endpoint and run your code there. We recommend that you start by setting up a development endpoint to work in. For more information, see [the section called "Viewing Development Endpoint Properties" \(p. 187\)](#).

Step 1: Crawl the Data in the Amazon S3 Bucket

1. Sign in to the AWS Management Console, and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Following the steps in [Working with Crawlers on the AWS Glue Console \(p. 127\)](#), create a new crawler that can crawl the `s3://awsglue-datasets/examples/us-legislators/all` dataset into a database named `legislators` in the AWS Glue Data Catalog. The example data is already in this public Amazon S3 bucket.
3. Run the new crawler, and then check the `legislators` database.

The crawler creates the following metadata tables:

- `persons_json`
- `memberships_json`
- `organizations_json`
- `events_json`
- `areas_json`
- `countries_r_json`

This is a semi-normalized collection of tables containing legislators and their histories.

Step 2: Add Boilerplate Script to the Development Endpoint Notebook

Paste the following boilerplate script into the development endpoint notebook to import the AWS Glue libraries that you need, and set up a single `GlueContext`:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
```

```
glueContext = GlueContext(SparkContext.getOrCreate())
```

Step 3: Examine the Schemas in the Data Catalog

Next, you can easily examine the schemas that the crawler recorded in the AWS Glue Data Catalog. For example, to see the schema of the `persons_json` table, add the following in your notebook:

```
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="persons_json")
print "Count: ", persons.count()
persons.printSchema()
```

Here's the output from the print calls:

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

Each person in the table is a member of some US congressional body.

To view the schema of the `memberships_json` table, type the following:

```
memberships = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="memberships_json")
print "Count: ", memberships.count()
memberships.printSchema()
```

The output is as follows:

```
Count: 10439
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string
```

The organizations are parties and the two chambers of Congress, the Senate and House of Representatives. To view the schema of the `organizations_json` table, type the following:

```
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="organizations_json")
print "Count: ", orgs.count()
orgs.printSchema()
```

The output is as follows:

```
Count: 13
root
|-- classification: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- id: string
|-- name: string
|-- seats: int
|-- type: string
```

Step 4: Filter the Data

Next, keep only the fields that you want, and rename `id` to `org_id`. The dataset is small enough that you can view the whole thing.

The `toDF()` converts a `DynamicFrame` to an Apache Spark `DataFrame`, so you can apply the transforms that already exist in Apache Spark SQL:

```
orgs = orgs.drop_fields(['other_names',
                        'identifiers']).rename_field(
    'id', 'org_id').rename_field(
    'name', 'org_name')
```

```
orgs.toDF().show()
```

The following shows the output:

classification	org_id	org_name	links	seats
type	image			
null	party/al	AL	null	null
null	party/democrat	Democrat	[[website,http://...]	null
null	party/democrat-li...	Democrat-Liberal	[[website,http://...]	null
null	legislature/d56acebe-8fdc-47b...	House of Represen...	null	435 lower
null	party/independent	Independent	null	null
null	party/new_progres...	New Progressive	[[website,http://...]	null
null	party/popular_dem...	Popular Democrat	[[website,http://...]	null
null	party/republican	Republican	[[website,http://...]	null
null	party/republican-...	Republican-Conser...	[[website,http://...]	null
null	party/democrat	Democrat	[[website,http://...]	null
null	party/independent	Independent	null	null
null	party/republican	Republican	[[website,http://...]	null
null	legislature/8fa6c3d2-71dc-478...	Senate	null	100 upper

Type the following to view the organizations that appear in memberships:

```
memberships.select_fields(['organization_id']).toDF().distinct().show()
```

The following shows the output:

organization_id
d56acebe-8fdc-47b...
8fa6c3d2-71dc-478...

Step 5: Put It All Together

Now, use AWS Glue to join these relational tables and create one full history table of legislator memberships and their corresponding organizations.

1. First, join persons and memberships on id and person_id.

2. Next, join the result with `orgs` on `org_id` and `organization_id`.
3. Then, drop the redundant fields, `person_id` and `org_id`.

You can do all these operations in one (extended) line of code:

```
l_history = Join.apply(orgs,
                      Join.apply(persons, memberships, 'id', 'person_id'),
                      'org_id', 'organization_id').drop_fields(['person_id', 'org_id'])
print "Count: ", l_history.count()
l_history.printSchema()
```

The output is as follows:

```
Count: 10439
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- death_date: string
|-- legislative_period_id: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- image: string
|-- given_name: string
|-- family_name: string
|-- id: string
|-- start_date: string
|-- end_date: string
```

You now have the final table that you can use for analysis. You can write it out in a compact, efficient format for analytics—namely Parquet—that you can run SQL over in AWS Glue, Amazon Athena, or Amazon Redshift Spectrum.

The following call writes the table across multiple files to support fast parallel reads when doing analysis later:

```
glueContext.write_dynamic_frame.from_options(frame = l_history,
    connection_type = "s3",
    connection_options = {"path": "s3://glue-sample-target/output-dir/
legislator_history"},
    format = "parquet")
```

To put all the history data into a single file, you must convert it to a data frame, repartition it, and write it out:

```
s_history = l_history.toDF().repartition(1)
s_history.write.parquet('s3://glue-sample-target/output-dir/legislator_single')
```

Or, if you want to separate it by the Senate and the House:

```
l_history.toDF().write.parquet('s3://glue-sample-target/output-dir/legislator_part',
    partitionBy=['org_name'])
```

Step 6: Write the Data to Relational Databases

AWS Glue makes it easy to write the data to relational databases like Amazon Redshift, even with semi-structured data. It offers a transform `relationalize`, which flattens `DynamicFrames` no matter how complex the objects in the frame might be.

Using the `l_history` `DynamicFrame` in this example, pass in the name of a root table (`hist_root`) and a temporary working path to `relationalize`. This returns a `DynamicFrameCollection`. You can then list the names of the `DynamicFrames` in that collection:

```
dfc = l_history.relationalize("hist_root", "s3://glue-sample-target/temp-dir/")
dfc.keys()
```

The following is the output of the `keys` call:

```
[u'hist_root', u'hist_root_contact_details', u'hist_root_links',
u'hist_root_other_names', u'hist_root_images', u'hist_root_identifiers']
```

`Relationalize` broke the history table out into six new tables: a root table that contains a record for each object in the `DynamicFrame`, and auxiliary tables for the arrays. Array handling in relational databases is often suboptimal, especially as those arrays become large. Separating the arrays into different tables makes the queries go much faster.

Next, look at the separation by examining `contact_details`:

```
l_history.select_fields('contact_details').printSchema()
dfc.select('hist_root_contact_details').toDF().where("id = 10 or id =
75").orderBy(['id','index']).show()
```

The following is the output of the `show` call:

```

root
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
+---+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+
| 10|    0|           fax|          202-225-1314|
| 10|    1|           |          |
| 10|    2|       phone|          202-225-3772|
| 10|    3|           |          |
| 10|    4|      twitter|        MikeRossUpdates|
| 10|    5|           fax|          202-225-7856|
| 75|    0|           |          |
| 75|    1|           |          |
| 75|    2|       phone|          202-225-2711|
| 75|    3|           |          |
| 75|    4|      twitter|        SenCapito|
| 75|    5|           |          |
+---+-----+-----+

```

The `contact_details` field was an array of structs in the original `DynamicFrame`. Each element of those arrays is a separate row in the auxiliary table, indexed by `index`. The `id` here is a foreign key into the `hist_root` table with the key `contact_details`:

```

dfc.select('hist_root').toDF().where(
    "contact_details = 10 or contact_details = 75").select(
        ['id', 'given_name', 'family_name', 'contact_details']).show()

```

The following is the output:

```

+-----+-----+-----+
|           id|given_name|family_name|contact_details|
+-----+-----+-----+
|f4fc30ee-7b42-432...|     Mike|      Ross|          10|
|e3c60f34-7d1b-4c0...| Shelley|      Capito|          75|
+-----+-----+-----+

```

Notice in these commands that `toDF()` and then a `where` expression are used to filter for the rows that you want to see.

So, joining the `hist_root` table with the auxiliary tables lets you do the following:

- Load data into databases without array support.
- Query each individual item in an array using SQL.

You already have a connection set up named `redshift3`. For information about how to create your own connection, see [the section called “Adding a Connection to Your Data Store” \(p. 110\)](#).

Next, write this collection into Amazon Redshift by cycling through the `DynamicFrames` one at a time:

```

for df_name in dfc.keys():
    m_df = dfc.select(df_name)
    print "Writing to Redshift table: ", df_name
    glueContext.write_dynamic_frame.from_jdbc_conf(frame = m_df,

```

```
catalog_connection = "redshift3",
connection_options = {"dbtable":  
df_name, "database": "testdb"},  
redshift_tmp_dir = "s3://glue-sample-  
target/temp-dir/")
```

The dbtable property is the name of the JDBC table. For JDBC data stores that support schemas within a database, specify schema.table-name. If a schema is not provided, then the default "public" schema is used.

For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

Here's what the tables look like in Amazon Redshift. (You connected to Amazon Redshift through psql.)

```
testdb=# \d
      List of relations
schema |       name        | type | owner
-----+-----+-----+-----+
public | hist_root          | table | test_user
public | hist_root_contact_details | table | test_user
public | hist_root_identifiers | table | test_user
public | hist_root_images    | table | test_user
public | hist_root_links     | table | test_user
public | hist_root_other_names | table | test_user
(6 rows)

testdb=# \d hist_root_contact_details
      Table "public.hist_root_contact_details"
 Column |           Type           | Modifiers
-----+-----+-----+
 id    | bigint
 index | integer
 contact_details.val.type | character varying(65535)
 contact_details.val.value | character varying(65535)

testdb=# \d hist_root
      Table "public.hist_root"
 Column |           Type           | Modifiers
-----+-----+-----+
 role   | character varying(65535)
 seats  | integer
 org_name | character varying(65535)
 links  | bigint
 type   | character varying(65535)
 sort_name | character varying(65535)
 area_id | character varying(65535)
 images  | bigint
 on_behalf_of_id | character varying(65535)
 other_names | bigint
 birth_date | character varying(65535)
 name    | character varying(65535)
 organization_id | character varying(65535)
 gender   | character varying(65535)
 classification | character varying(65535)
 legislative_period_id | character varying(65535)
 identifiers | bigint
 given_name | character varying(65535)
 image    | character varying(65535)
 family_name | character varying(65535)
 id      | character varying(65535)
 death_date | character varying(65535)
 start_date | character varying(65535)
 contact_details | bigint
```

```
| end_date | character varying(65535) |
```

Now you can query these tables using SQL in Amazon Redshift:

```
testdb=# select * from hist_root_contact_details where id = 10 or id = 75 order by id, index;
```

The following shows the result:

id	index	contact_details.val.type	contact_details.val.value
10	0	fax	202-224-6020
10	1	phone	202-224-3744
10	2	twitter	ChuckGrassley
75	0	fax	202-224-4680
75	1	phone	202-224-4642
75	2	twitter	SenJackReed

(6 rows)

Conclusion

Overall, AWS Glue is very flexible. It lets you accomplish, in a few lines of code, what normally would take days to write. You can find the entire source-to-target ETL scripts in the Python file `join_and_relatealize.py` in the [AWS Glue samples](#) on GitHub.

Code Example: Data Preparation Using ResolveChoice, Lambda, and ApplyMapping

The dataset that is used in this example consists of Medicare Provider payment data downloaded from two Data.CMS.gov sites: [Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011](#), and [Inpatient Charge Data FY 2011](#). After downloading it, we modified the data to introduce a couple of erroneous records at the end of the file. This modified file is located in a public Amazon S3 bucket at `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`.

You can find the source code for this example in the `data_cleaning_and_lambda.py` file in the [AWS Glue examples](#) GitHub repository.

The easiest way to debug Python or PySpark scripts is to create a development endpoint and run your code there. We recommend that you start by setting up a development endpoint to work in. For more information, see [the section called "Viewing Development Endpoint Properties" \(p. 187\)](#).

Step 1: Crawl the Data in the Amazon S3 Bucket

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Following the process described in [Working with Crawlers on the AWS Glue Console \(p. 127\)](#), create a new crawler that can crawl the `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv` file, and can place the resulting metadata into a database named `payments` in the AWS Glue Data Catalog.
3. Run the new crawler, and then check the `payments` database. You should find that the crawler has created a metadata table named `medicare` in the database after reading the beginning of the file to determine its format and delimiter.

The schema of the new `medicare` table is as follows:

Column name	Data type
drg definition	string
provider id	bigint
provider name	string
provider street address	string
provider city	string
provider state	string
provider zip code	bigint
hospital referral region description	string
total discharges	bigint
average covered charges	string
average total payments	string
average medicare payments	string

Step 2: Add Boilerplate Script to the Development Endpoint Notebook

Paste the following boilerplate script into the development endpoint notebook to import the AWS Glue libraries that you need, and set up a single `GlueContext`:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

Step 3: Compare Different Schema Parses

Next, you can see if the schema that was recognized by an Apache Spark `DataFrame` is the same as the one that your AWS Glue crawler recorded. Run this code:

```
medicare = spark.read.format(
    "com.databricks.spark.csv").option(
    "header", "true").option(
    "inferSchema", "true").load(
    's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

Here's the output from the `printSchema` call:

```
root
 |-- DRG Definition: string (nullable = true)
 |-- Provider Id: string (nullable = true)
 |-- Provider Name: string (nullable = true)
 |-- Provider Street Address: string (nullable = true)
 |-- Provider City: string (nullable = true)
 |-- Provider State: string (nullable = true)
 |-- Provider Zip Code: integer (nullable = true)
 |-- Hospital Referral Region Description: string (nullable = true)
 |-- Total Discharges : integer (nullable = true)
 |-- Average Covered Charges : string (nullable = true)
 |-- Average Total Payments : string (nullable = true)
 |-- Average Medicare Payments: string (nullable = true)
```

Next, look at the schema that an AWS Glue DynamicFrame generates:

```
medicare_dynamicframe = glueContext.create_dynamic_frame.from_catalog(
    database = "payments",
    table_name = "medicare")
medicare_dynamicframe.printSchema()
```

The output from printSchema is as follows:

```
root
|-- drg definition: string
|-- provider id: choice
|   |-- long
|   |-- string
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

The DynamicFrame generates a schema in which provider id could be either a long or a string type. The DataFrame schema lists Provider Id as being a string type, and the Data Catalog lists provider id as being a bigint type.

Which one is correct? There are two records at the end of the file (out of 160,000 records) with string values in that column. These are the erroneous records that were introduced to illustrate a problem.

To address this kind of problem, the AWS Glue DynamicFrame introduces the concept of a *choice* type. In this case, the DynamicFrame shows that both long and string values can appear in that column. The AWS Glue crawler missed the string values because it considered only a 2 MB prefix of the data. The Apache Spark DataFrame considered the whole dataset, but it was forced to assign the most general type to the column, namely string. In fact, Spark often resorts to the most general case when there are complex types or variations with which it is unfamiliar.

To query the provider id column, resolve the choice type first. You can use the resolveChoice transform method in your DynamicFrame to convert those string values to long values with a cast:long option:

```
medicare_res = medicare_dynamicframe.resolveChoice(specs = [('provider id','cast:long')])
medicare_res.printSchema()
```

The printSchema output is now:

```
root
|-- drg definition: string
|-- provider id: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
```

```
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

Where the value was a string that could not be cast, AWS Glue inserted a null.

Another option is to convert the choice type to a struct, which keeps values of both types.

Next, look at the rows that were anomalous:

```
medicare_res.toDF().where("'provider id' is NULL").show()
```

You see the following:

drg definition provider id provider name provider street address provider city provider state provider zip code hospital referral region description total discharges average covered charges average total payments average medicare payments
948 - SIGNS & SYM... null INC 1050 DIVISION ST MAUSTON WI 53948 \$11961.41 \$4619.00 \$3775.33
948 - SIGNS & SYM... null INC- ST JOSEPH 5000 W CHAMBERS ST MILWAUKEE WI 53210 \$10514.28 \$5562.50 \$4522.78

Now remove the two malformed records, as follows:

```
medicare_dataframe = medicare_res.toDF()
medicare_dataframe = medicare_dataframe.where("'provider id' is NOT NULL")
```

Step 4: Map the Data and Use Apache Spark Lambda Functions

AWS Glue does not yet directly support Lambda functions, also known as user-defined functions. But you can always convert a DynamicFrame to and from an Apache Spark DataFrame to take advantage of Spark functionality in addition to the special features of DynamicFrames.

Next, turn the payment information into numbers, so analytic engines like Amazon Redshift or Amazon Athena can do their number crunching faster:

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

chop_f = udf(lambda x: x[1:], StringType())
medicare_dataframe = medicare_dataframe.withColumn(
    "ACC", chop_f(
        medicare_dataframe["average covered charges"])).withColumn(
    "ATP", chop_f(
        medicare_dataframe["average total payments"])).withColumn(
    "AMP", chop_f(
        medicare_dataframe["average medicare payments"]))
```

```
medicare_dataframe.select(['ACC', 'ATP', 'AMP']).show()
```

The output from the show call is as follows:

```
+-----+-----+-----+
|     ACC |      ATP |      AMP |
+-----+-----+-----+
|32963.07|5777.24|4763.73|
|15131.85|5787.57|4976.71|
|37560.37|5434.95|4453.79|
|13998.28|5417.56|4129.16|
|31633.27|5658.33|4851.44|
|16920.79|6653.80|5374.14|
|11977.13|5834.74|4761.41|
|35841.09|8031.12|5858.50|
|28523.39|6113.38|5228.40|
|75233.38|5541.05|4386.94|
|67327.92|5461.57|4493.57|
|39607.28|5356.28|4408.20|
|122862.23|5374.65|4186.02|
|31110.85|5366.23|4376.23|
|25411.33|5282.93|4383.73|
| 9234.51|5676.55|4509.11|
|15895.85|5930.11|3972.85|
|19721.16|6192.54|5179.38|
|10710.88|4968.00|3898.88|
|51343.75|5996.00|4962.45|
+-----+-----+-----+
only showing top 20 rows
```

These are all still strings in the data. We can use the powerful `apply_mapping` transform method to drop, rename, cast, and nest the data so that other data programming languages and systems can easily access it:

```
medicare_tmp_dyf = DynamicFrame.fromDF(medicare_dataframe, glueContext, "nested")
medicare_nest_dyf = medicare_tmp_dyf.apply_mapping([
    ('drg definition', 'string', 'drg', 'string'),
    ('provider id', 'long', 'provider.id', 'long'),
    ('provider name', 'string', 'provider.name', 'string'),
    ('provider city', 'string', 'provider.city', 'string'),
    ('provider state', 'string', 'provider.state', 'string'),
    ('provider zip code', 'long', 'provider.zip', 'long'),
    ('hospital referral region description', 'string', 'rr', 'string'),
    ('ACC', 'string', 'charges.covered', 'double'),
    ('ATP', 'string', 'charges.total_pay', 'double'),
    ('AMP', 'string', 'charges.medicare_pay', 'double')])
medicare_nest_dyf.printSchema()
```

The `printSchema` output is as follows:

```
root
 |-- drg: string
 |-- provider: struct
 |   |-- id: long
 |   |-- name: string
 |   |-- city: string
 |   |-- state: string
 |   |-- zip: long
 |-- rr: string
 |-- charges: struct
 |   |-- covered: double
 |   |-- total_pay: double
```

```
| -- medicare_pay: double
```

Turning the data back into a Spark DataFrame, you can show what it looks like now:

```
medicare_nest_dyf.toDF().show()
```

The output is as follows:

drg	provider	rr	charges
039 - EXTRACRANIA... [10001,SOUTHEAST ...	AL - Dothan	[32963.07,5777.24...]	
039 - EXTRACRANIA... [10005,MARSHALL M...	AL - Birmingham	[15131.85,5787.57...]	
039 - EXTRACRANIA... [10006,ELIZA COFF...	AL - Birmingham	[37560.37,5434.95...]	
039 - EXTRACRANIA... [10011,ST VINCENT...	AL - Birmingham	[13998.28,5417.56...]	
039 - EXTRACRANIA... [10016,SHELBY BAP...	AL - Birmingham	[31633.27,5658.33...]	
039 - EXTRACRANIA... [10023,BAPTIST ME...	AL - Montgomery	[16920.79,6653.8,...]	
039 - EXTRACRANIA... [10029,EAST ALABA...	AL - Birmingham	[11977.13,5834.74...]	
039 - EXTRACRANIA... [10033,UNIVERSITY...	AL - Birmingham	[35841.09,8031.12...]	
039 - EXTRACRANIA... [10039,HUNTSVILLE...	AL - Huntsville	[28523.39,6113.38...]	
039 - EXTRACRANIA... [10040,GADSDEN RE...	AL - Birmingham	[75233.38,5541.05...]	
039 - EXTRACRANIA... [10046,RIVERVIEW ...	AL - Birmingham	[67327.92,5461.57...]	
039 - EXTRACRANIA... [10055,FLOWERS HO...	AL - Dothan	[39607.28,5356.28...]	
039 - EXTRACRANIA... [10056,ST VINCENT...	AL - Birmingham	[22862.23,5374.65...]	
039 - EXTRACRANIA... [10078,NORTHEAST ...	AL - Birmingham	[31110.85,5366.23...]	
039 - EXTRACRANIA... [10083,SOUTH BALD...	AL - Mobile	[25411.33,5282.93...]	
039 - EXTRACRANIA... [10085,DECATUR GE...	AL - Huntsville	[9234.51,5676.55,...]	
039 - EXTRACRANIA... [10090,PROVIDENCE...	AL - Mobile	[15895.85,5930.11...]	
039 - EXTRACRANIA... [10092,D C H REGI...	AL - Tuscaloosa	[19721.16,6192.54...]	
039 - EXTRACRANIA... [10100,THOMAS HOS...	AL - Mobile	[10710.88,4968.0,...]	
039 - EXTRACRANIA... [10103,BAPTIST ME...	AL - Birmingham	[51343.75,5996.0,...]	

only showing top 20 rows

Step 5: Write the Data to Apache Parquet

AWS Glue makes it easy to write the data in a format such as Apache Parquet that relational databases can effectively consume:

```
glueContext.write_dynamic_frame.from_options(
    frame = medicare_nest_dyf,
    connection_type = "s3",
    connection_options = {"path": "s3://glue-sample-target/output-dir/
    medicare_parquet"},
    format = "parquet")
```

AWS Glue PySpark Extensions Reference

AWS Glue has created the following extensions to the PySpark Python dialect.

- [Accessing Parameters Using getResolvedOptions \(p. 333\)](#)
- [PySpark Extension Types \(p. 333\)](#)
- [DynamicFrame Class \(p. 337\)](#)
- [DynamicFrameCollection Class \(p. 348\)](#)
- [DynamicFrameWriter Class \(p. 349\)](#)
- [DynamicFrameReader Class \(p. 350\)](#)
- [GlueContext Class \(p. 352\)](#)

Accessing Parameters Using `getResolvedOptions`

The AWS Glue `getResolvedOptions(args, options)` utility function gives you access to the arguments that are passed to your script when you run a job. To use this function, start by importing it from the AWS Glue `utils` module, along with the `sys` module:

```
import sys
from awsglue.utils import getResolvedOptions
```

`getResolvedOptions(args, options)`

- `args` – The list of arguments contained in `sys.argv`.
- `options` – A Python array of the argument names that you want to retrieve.

Example Retrieving arguments passed to a JobRun

Suppose that you created a JobRun in a script, perhaps within a Lambda function:

```
response = client.start_job_run(
    JobName = 'my_test_Job',
    Arguments = {
        '--day_partition_key': 'partition_0',
        '--hour_partition_key': 'partition_1',
        '--day_partition_value': day_partition_value,
        '--hour_partition_value': hour_partition_value } )
```

To retrieve the arguments that are passed, you can use the `getResolvedOptions` function as follows:

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
    ['JOB_NAME',
     'day_partition_key',
     'hour_partition_key',
     'day_partition_value',
     'hour_partition_value'])
print "The day-partition key is: ", args['day_partition_key']
print "and the day-partition value is: ", args['day_partition_value']
```

Note that each of the arguments are defined as beginning with two hyphens, then referenced in the script without the hyphens. Your arguments need to follow this convention to be resolved.

PySpark Extension Types

The types that are used by the AWS Glue PySpark extensions.

`DataType`

The base class for the other AWS Glue types.

`__init__(properties={})`

- `properties` – Properties of the data type (optional).

typeName(`cls`)

Returns the type of the AWS Glue type class (that is, the class name with "Type" removed from the end).

- `cls` – An AWS Glue class instance derived from `DataType`.

jsonValue()

Returns a JSON object that contains the data type and properties of the class:

```
{  
    "dataType": typeName,  
    "properties": properties  
}
```

[AtomicType and Simple Derivatives](#)

Inherits from and extends the [DataType \(p. 333\)](#) class, and serves as the base class for all the AWS Glue atomic data types.

fromJsonValue(`cls`, `json_value`)

Initializes a class instance with values from a JSON object.

- `cls` – An AWS Glue type class instance to initialize.
- `json_value` – The JSON object to load key-value pairs from.

The following types are simple derivatives of the [AtomicType \(p. 334\)](#) class:

- `BinaryType` – Binary data.
- `BooleanType` – Boolean values.
- `ByteType` – A byte value.
- `DateType` – A datetime value.
- `DoubleType` – A floating-point double value.
- `IntegerType` – An integer value.
- `LongType` – A long integer value.
- `NullType` – A null value.
- `ShortType` – A short integer value.
- `StringType` – A text string.
- `TimestampType` – A timestamp value (typically in seconds from 1/1/1970).
- `UnknownType` – A value of unidentified type.

[DecimalType\(AtomicType\)](#)

Inherits from and extends the [AtomicType \(p. 334\)](#) class to represent a decimal number (a number expressed in decimal digits, as opposed to binary base-2 numbers).

__init__(precision=10, scale=2, properties={})

- `precision` – The number of digits in the decimal number (optional; the default is 10).
- `scale` – The number of digits to the right of the decimal point (optional; the default is 2).
- `properties` – The properties of the decimal number (optional).

EnumType(AtomicType)

Inherits from and extends the [AtomicType \(p. 334\)](#) class to represent an enumeration of valid options.

`__init__(options)`

- `options` – A list of the options being enumerated.

Collection Types

- [ArrayType\(DataType\) \(p. 335\)](#)
- [ChoiceType\(DataType\) \(p. 335\)](#)
- [MapType\(DataType\) \(p. 335\)](#)
- [Field\(Object\) \(p. 336\)](#)
- [StructType\(DataType\) \(p. 336\)](#)
- [EntityType\(DataType\) \(p. 336\)](#)

ArrayType(DataType)

`__init__(elementType=UnknownType(), properties={})`

- `elementType` – The type of elements in the array (optional; the default is `UnknownType`).
- `properties` – Properties of the array (optional).

ChoiceType(DataType)

`__init__(choices=[], properties={})`

- `choices` – A list of possible choices (optional).
- `properties` – Properties of these choices (optional).

`add(new_choice)`

Adds a new choice to the list of possible choices.

- `new_choice` – The choice to add to the list of possible choices.

`merge(new_choices)`

Merges a list of new choices with the existing list of choices.

- `new_choices` – A list of new choices to merge with existing choices.

MapType(DataType)

`__init__(valueType=UnknownType, properties={})`

- `valueType` – The type of values in the map (optional; the default is `UnknownType`).
- `properties` – Properties of the map (optional).

Field(Object)

Creates a field object out of an object that derives from [DataType \(p. 333\)](#).

`__init__(name, dataType, properties={})`

- `name` – The name to be assigned to the field.
- `dataType` – The object to create a field from.
- `properties` – Properties of the field (optional).

StructType(DataType)

Defines a data structure (`struct`).

`__init__(fields=[], properties={})`

- `fields` – A list of the fields (of type `Field`) to include in the structure (optional).
- `properties` – Properties of the structure (optional).

`add(field)`

- `field` – An object of type `Field` to add to the structure.

`hasField(field)`

Returns `True` if this structure has a field of the same name, or `False` if not.

- `field` – A field name, or an object of type `Field` whose name is used.

`getField(field)`

- `field` – A field name or an object of type `Field` whose name is used. If the structure has a field of the same name, it is returned.

EntityType(DataType)

`__init__(entity, base_type, properties)`

This class is not yet implemented.

Other Types

- [DataSource\(object\) \(p. 336\)](#)
- [DataSink\(object\) \(p. 337\)](#)

DataSource(object)

`__init__(j_source, sql_ctx, name)`

- `j_source` – The data source.
- `sql_ctx` – The SQL context.

- name – The data-source name.

setFormat(format, **options)

- format – The format to set for the data source.
- options – A collection of options to set for the data source.

getFrame()

Returns a DynamicFrame for the data source.

DataSink(object)

__init__(j_sink, sql_ctx)

- j_sink – The sink to create.
- sql_ctx – The SQL context for the data sink.

setFormat(format, **options)

- format – The format to set for the data sink.
- options – A collection of options to set for the data sink.

setAccumulableSize(size)

- size – The accumulable size to set, in bytes.

writeFrame(dynamic_frame, info="")

- dynamic_frame – The DynamicFrame to write.
- info – Information about the DynamicFrame (optional).

write(dynamic_frame_or_dfc, info="")

Writes a DynamicFrame or a DynamicFrameCollection.

- dynamic_frame_or_dfc – Either a DynamicFrame object or a DynamicFrameCollection object to be written.
- info – Information about the DynamicFrame or DynamicFrames to be written (optional).

DynamicFrame Class

One of the major abstractions in Apache Spark is the SparkSQL DataFrame, which is similar to the DataFrame construct found in R and Pandas. A DataFrame is similar to a table and supports functional-style (map/reduce/filter/etc.) operations and SQL operations (select, project, aggregate).

DataFrames are powerful and widely used, but they have limitations with respect to extract, transform, and load (ETL) operations. Most significantly, they require a schema to be specified before any data is

loaded. SparkSQL addresses this by making two passes over the data—the first to infer the schema, and the second to load the data. However, this inference is limited and doesn't address the realities of messy data. For example, the same field might be of a different type in different records. Apache Spark often gives up and reports the type as `string` using the original field text. This might not be correct, and you might want finer control over how schema discrepancies are resolved. And for large datasets, an additional pass over the source data might be prohibitively expensive.

To address these limitations, AWS Glue introduces the `DynamicFrame`. A `DynamicFrame` is similar to a `DataFrame`, except that each record is self-describing, so no schema is required initially. Instead, AWS Glue computes a schema on-the-fly when required, and explicitly encodes schema inconsistencies using a choice (or union) type. You can resolve these inconsistencies to make your datasets compatible with data stores that require a fixed schema.

Similarly, a `DynamicRecord` represents a logical record within a `DynamicFrame`. It is like a row in a `Spark DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

You can convert `DynamicFrames` to and from `DataFrames` after you resolve any schema inconsistencies.

— Construction —

- [__init__ \(p. 338\)](#)
- [fromDF \(p. 338\)](#)
- [toDF \(p. 338\)](#)

__init__

`__init__(jdf, glue_ctx, name)`

- `jdf` – A reference to the data frame in the Java Virtual Machine (JVM).
- `glue_ctx` – A [GlueContext Class \(p. 352\)](#) object.
- `name` – An optional name string, empty by default.

[fromDF](#)

`fromDF(dataframe, glue_ctx, name)`

Converts a `DataFrame` to a `DynamicFrame` by converting `DataFrame` fields to `DynamicRecord` fields. Returns the new `DynamicFrame`.

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in a `Spark DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `dataframe` – The Apache Spark SQL `DataFrame` to convert (required).
- `glue_ctx` – The [GlueContext Class \(p. 352\)](#) object that specifies the context for this transform (required).
- `name` – The name of the resulting `DynamicFrame` (required).

[toDF](#)

`toDF(options)`

Converts a `DynamicFrame` to an Apache Spark `DataFrame` by converting `DynamicRecords` into `DataFrame` fields. Returns the new `DataFrame`.

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in a Spark `Dataframe`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `options` – A list of options. Specify the target type if you choose the `Project` and `Cast` action type. Examples include the following.

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])  
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

— Information —

- [count \(p. 339\)](#)
- [schema \(p. 339\)](#)
- [printSchema \(p. 339\)](#)
- [show \(p. 339\)](#)
- [repartition \(p. 339\)](#)
- [coalesce \(p. 339\)](#)

count

`count()` – Returns the number of rows in the underlying `Dataframe`.

schema

`schema()` – Returns the schema of this `DynamicFrame`, or if that is not available, the schema of the underlying `Dataframe`.

printSchema

`printSchema()` – Prints the schema of the underlying `Dataframe`.

show

`show(num_rows)` – Prints a specified number of rows from the underlying `Dataframe`.

repartition

`repartition(numPartitions)` – Returns a new `DynamicFrame` with `numPartitions` partitions.

coalesce

`coalesce(numPartitions)` – Returns a new `DynamicFrame` with `numPartitions` partitions.

— Transforms —

- [apply_mapping \(p. 340\)](#)
- [drop_fields \(p. 340\)](#)
- [filter \(p. 340\)](#)
- [join \(p. 341\)](#)
- [map \(p. 341\)](#)
- [mergeDynamicFrame \(p. 342\)](#)
- [relationalize \(p. 342\)](#)
- [rename_field \(p. 343\)](#)

- [resolveChoice \(p. 343\)](#)
- [select_fields \(p. 344\)](#)
- [spigot \(p. 345\)](#)
- [split_fields \(p. 345\)](#)
- [split_rows \(p. 345\)](#)
- [unbox \(p. 346\)](#)
- [unnest \(p. 346\)](#)
- [write \(p. 347\)](#)

apply_mapping

```
apply_mapping(mappings, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)
```

Applies a declarative mapping to this `DynamicFrame` and returns a new `DynamicFrame` with those mappings applied.

- `mappings` – A list of mapping tuples, each consisting of: (source column, source type, target column, target type). Required.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

drop_fields

```
drop_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)
```

Calls the [FlatMap Class \(p. 369\)](#) transform to remove fields from a `DynamicFrame`. Returns a new `DynamicFrame` with the specified fields dropped.

- `paths` – A list of strings, each containing the full path to a field node you want to drop.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

filter

```
filter(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)
```

Returns a new `DynamicFrame` built by selecting all `DynamicRecords` within the input `DynamicFrame` that satisfy the specified predicate function `f`.

- **f** – The predicate function to apply to the `DynamicFrame`. The function must take a `DynamicRecord` as an argument and return True if the `DynamicRecord` meets the filter requirements, or False if not (required).

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in a Spark `DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

For an example of how to use the `filter` transform, see [Filter Class \(p. 367\)](#).

join

```
join(paths1, paths2, frame2, transformation_ctx="", info="",
      stageThreshold=0, totalThreshold=0)
```

Performs an equality join with another `DynamicFrame` and returns the resulting `DynamicFrame`.

- `paths1` – A list of the keys in this frame to join.
- `paths2` – A list of the keys in the other frame to join.
- `frame2` – The other `DynamicFrame` to join.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

map

```
map(f, transformation_ctx="", info="", stageThreshold=0,
     totalThreshold=0)
```

Returns a new `DynamicFrame` that results from applying the specified mapping function to all records in the original `DynamicFrame`.

- `f` – The mapping function to apply to all records in the `DynamicFrame`. The function must take a `DynamicRecord` as an argument and return a new `DynamicRecord` (required).

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in an Apache Spark `DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).

- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

For an example of how to use the `map` transform, see [Map Class \(p. 371\)](#).

mergeDynamicFrame

```
mergeDynamicFrame(stage_dynamic_frame, primary_keys, transformation_ctx = "", options = {}, info = "", stageThreshold = 0, totalThreshold = 0)
```

Merges this `DynamicFrame` with a staging `DynamicFrame` based on the specified primary keys to identify records. Duplicate records (records with the same primary keys) are not de-duplicated. If there is no matching record in the staging frame, all records (including duplicates) are retained from the source. If the staging frame has matching records, the records from the staging frame overwrite the records in the source in AWS Glue.

- `stage_dynamic_frame` – The staging `DynamicFrame` to merge.
- `primary_keys` – The list of primary key fields to match records from the source and staging dynamic frames.
- `transformation_ctx` – A unique string that is used to retrieve metadata about the current transformation (optional).
- `options` – A string of JSON name-value pairs that provide additional information for this transformation.
- `info` – A String. Any string to be associated with errors in this transformation.
- `stageThreshold` – A Long. The number of errors in the given transformation for which the processing needs to error out.
- `totalThreshold` – A Long. The total number of errors up to and including in this transformation for which the processing needs to error out.

Returns a new `DynamicFrame` obtained by merging this `DynamicFrame` with the staging `DynamicFrame`.

The returned `DynamicFrame` contains record A in these cases:

1. If A exists in both the source frame and the staging frame, then A in the staging frame is returned.
2. If A is in the source table and A.primaryKeys is not in the `stagingDynamicFrame` (that means A is not updated in the staging table).

The source frame and staging frame do not need to have the same schema.

Example

```
merged_frame = source_frame.mergeDynamicFrame(stage_frame, ["id"])
```

relationalize

```
relationalize(root_table_name, staging_path, options, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)
```

Relationalizes a `DynamicFrame` by producing a list of frames that are generated by unnesting nested columns and pivoting array columns. The pivoted array column can be joined to the root table using the `joinkey` generated during the unnest phase.

- `root_table_name` – The name for the root table.
- `staging_path` – The path at which to store partitions of pivoted tables in CSV format (optional). Pivoted tables are read back from this path.
- `options` – A dictionary of optional parameters.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

rename_field

```
rename_field(oldName, newName, transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

Renames a field in this `DynamicFrame` and returns a new `DynamicFrame` with the field renamed.

- `oldName` – The full path to the node you want to rename.

If the old name has dots in it, `RenameField` doesn't work unless you place back-ticks around it (`). For example, to replace `this.old.name` with `thisNewName`, you would call `rename_field` as follows.

```
newDyF = oldDyF.rename_field(`this.old.name`, "thisNewName")
```

- `newName` – The new name, as a full path.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

resolveChoice

```
resolveChoice(specs = None, option="", transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

Resolves a choice type within this `DynamicFrame` and returns the new `DynamicFrame`.

- `specs` – A list of specific ambiguities to resolve, each in the form of a tuple: `(path, action)`. The `path` value identifies a specific ambiguous element, and the `action` value identifies the corresponding resolution. Only one of the `specs` and `option` parameters can be used. If the `spec` parameter is not `None`, then the `option` parameter must be an empty string. Conversely if the `option` is not an empty string, then the `spec` parameter must be `None`. If neither parameter is provided, AWS Glue tries to parse the schema and use it to resolve ambiguities.

The `action` portion of a `specs` tuple can specify one of four resolution strategies:

- `cast`: Allows you to specify a type to cast to (for example, `cast:int`).

- `make_cols`: Resolves a potential ambiguity by flattening the data. For example, if `columnA` could be an `int` or a `string`, the resolution would be to produce two columns named `columnA_int` and `columnA_string` in the resulting `DynamicFrame`.
- `make_struct`: Resolves a potential ambiguity by using a struct to represent the data. For example, if data in a column could be an `int` or a `string`, using the `make_struct` action produces a column of structures in the resulting `DynamicFrame` that each contains both an `int` and a `string`.
- `project`: Resolves a potential ambiguity by projecting all the data to one of the possible data types. For example, if data in a column could be an `int` or a `string`, using a `project:string` action produces a column in the resulting `DynamicFrame` where all the `int` values have been converted to strings.

If the path identifies an array, place empty square brackets after the name of the array to avoid ambiguity. For example, suppose you are working with data structured as follows:

```
"myList": [
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

You can select the numeric rather than the string version of the price by setting the path to `"myList[].price"`, and the action to `"cast:double"`.

- `option` – The default resolution action if the `specs` parameter is `None`. If the `specs` parameter is not `None`, then this must not be set to anything but an empty string.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

Example

```
df1 = df.resolveChoice(option = "make_cols")
df2 = df.resolveChoice(specs = [("a.b", "make_struct"), ("c.d", "cast:double")])
```

select_fields

```
select_fields(paths, transformation_ctx="", info="", stageThreshold=0,
totalThreshold=0)
```

Returns a new `DynamicFrame` containing the selected fields.

- `paths` – A list of strings, each of which is a path to a top-level node that you want to select.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

spigot

```
spigot(path, options={})
```

Writes sample records to a specified destination during a transformation, and returns the input DynamicFrame with an additional write step.

- path – The path to the destination to which to write (required).
- options – Key-value pairs specifying options (optional). The "topk" option specifies that the first k records should be written. The "prob" option specifies the probability (as a decimal) of picking any given record, to be used in selecting records to write.
- transformation_ctx – A unique string that is used to identify state information (optional).

split_fields

```
split_fields(paths, name1, name2, transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

Returns a new DynamicFrameCollection that contains two DynamicFrames: the first containing all the nodes that have been split off, and the second containing the nodes that remain.

- paths – A list of strings, each of which is a full path to a node that you want to split into a new DynamicFrame.
- name1 – A name string for the DynamicFrame that is split off.
- name2 – A name string for the DynamicFrame that remains after the specified nodes have been split off.
- transformation_ctx – A unique string that is used to identify state information (optional).
- info – A string to be associated with error reporting for this transformation (optional).
- stageThreshold – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- totalThreshold – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

split_rows

Splits one or more rows in a DynamicFrame off into a new DynamicFrame.

```
split_rows(comparison_dict, name1, name2, transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

Returns a new DynamicFrameCollection containing two DynamicFrames: the first containing all the rows that have been split off and the second containing the rows that remain.

- comparison_dict – A dictionary in which the key is a path to a column and the value is another dictionary for mapping comparators to values to which the column value are compared. For example, {"age": {">": 10, "<": 20}} splits off all rows whose value in the age column is greater than 10 and less than 20.
- name1 – A name string for the DynamicFrame that is split off.
- name2 – A name string for the DynamicFrame that remains after the specified nodes have been split off.
- transformation_ctx – A unique string that is used to identify state information (optional).
- info – A string to be associated with error reporting for this transformation (optional).

- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

unbox

```
unbox(path, format, transformation_ctx="", info="", stageThreshold=0,  
totalThreshold=0, **options)
```

Unboxes a string field in a `DynamicFrame` and returns a new `DynamicFrame` containing the unboxed `DynamicRecords`.

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in an Apache Spark `DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `path` – A full path to the string node you want to unbox.
- `format` – A format specification (optional). This is used for an Amazon Simple Storage Service (Amazon S3) or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).
- `options` – One or more of the following:
 - `separator` – A string containing the separator character.
 - `escaper` – A string containing the escape character.
 - `skipFirst` – A Boolean value indicating whether to skip the first instance.
 - `withSchema` – A string containing the schema; must be called using `StructType.json()`.
 - `withHeader` – A Boolean value indicating whether a header is included.

For example: `unbox("a.b.c", "csv", separator="|")`

unnest

Unnests nested objects in a `DynamicFrame`, making them top-level objects, and returns a new unnested `DynamicFrame`.

```
unnest(transformation_ctx="", info="", stageThreshold=0,  
totalThreshold=0)
```

Unnests nested objects in a `DynamicFrame`, making them top-level objects, and returns a new unnested `DynamicFrame`.

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string to be associated with error reporting for this transformation (optional).
- `stageThreshold` – The number of errors encountered during this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

- `totalThreshold` – The number of errors encountered up to and including this transformation at which the process should error out (optional: zero by default, indicating that the process should not error out).

For example: `unnest()`

write

```
write(connection_type, connection_options, format, format_options,  
      accumulator_size)
```

Gets a [DataSink\(object\) \(p. 337\)](#) of the specified connection type from the [GlueContext Class \(p. 352\)](#) of this `DynamicFrame`, and uses it to format and write the contents of this `DynamicFrame`. Returns the new `DynamicFrame` formatted and written as specified.

- `connection_type` – The connection type to use. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, and `oracle`.
- `connection_options` – The connection option to use (optional). For a `connection_type` of `s3`, an Amazon S3 path is defined.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password":  
                      "password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

- `format` – A format specification (optional). This is used for an Amazon Simple Storage Service (Amazon S3) or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `format_options` – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `accumulator_size` – The accumulable size to use (optional).

— Errors —

- [assertErrorThreshold \(p. 347\)](#)
- [errorsAsDynamicFrame \(p. 347\)](#)
- [errorsCount \(p. 347\)](#)
- [stageErrorsCount \(p. 348\)](#)

assertErrorThreshold

`assertErrorThreshold()` – An assert for errors in the transformations that created this `DynamicFrame`. Returns an `Exception` from the underlying `DataFrame`.

errorsAsDynamicFrame

`errorsAsDynamicFrame()` – Returns a `DynamicFrame` that has error records nested inside.

errorsCount

`errorsCount()` – Returns the total number of errors in a `DynamicFrame`.

stageErrorsCount

`stageErrorsCount` – Returns the number of errors that occurred in the process of generating this `DynamicFrame`.

DynamicFrameCollection Class

A `DynamicFrameCollection` is a dictionary of [DynamicFrame Class \(p. 337\)](#) objects, in which the keys are the names of the `DynamicFrames` and the values are the `DynamicFrame` objects.

`__init__`

`__init__(dynamic_frames, glue_ctx)`

- `dynamic_frames` – A dictionary of [DynamicFrame Class \(p. 337\)](#) objects.
- `glue_ctx` – A [GlueContext Class \(p. 352\)](#) object.

keys

`keys()` – Returns a list of the keys in this collection, which generally consists of the names of the corresponding `DynamicFrame` values.

values

`values(key)` – Returns a list of the `DynamicFrame` values in this collection.

select

`select(key)`

Returns the `DynamicFrame` that corresponds to the specified key (which is generally the name of the `DynamicFrame`).

- `key` – A key in the `DynamicFrameCollection`, which usually represents the name of a `DynamicFrame`.

map

`map(callable, transformation_ctx="")`

Uses a passed-in function to create and return a new `DynamicFrameCollection` based on the `DynamicFrames` in this collection.

- `callable` – A function that takes a `DynamicFrame` and the specified transformation context as parameters and returns a `DynamicFrame`.
- `transformation_ctx` – A transformation context to be used by the callable (optional).

flatmap

`flatmap(f, transformation_ctx="")`

Uses a passed-in function to create and return a new `DynamicFrameCollection` based on the `DynamicFrames` in this collection.

- `f` – A function that takes a `DynamicFrame` as a parameter and returns a `DynamicFrame` or `DynamicFrameCollection`.
- `transformation_ctx` – A transformation context to be used by the function (optional).

DynamicFrameWriter Class

Methods

- [__init__ \(p. 349\)](#)
- [from_options \(p. 349\)](#)
- [from_catalog \(p. 350\)](#)
- [from_jdbc_conf \(p. 350\)](#)

[__init__](#)

[__init__\(glue_context\)](#)

- `glue_context` – The [GlueContext Class \(p. 352\)](#) to use.

[from_options](#)

```
from_options(frame, connection_type, connection_options={}, format=None,  
format_options={}, transformation_ctx="")
```

Writes a `DynamicFrame` using the specified connection and format.

- `frame` – The `DynamicFrame` to write.
- `connection_type` – The connection type. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, and `oracle`.
- `connection_options` – Connection options, such as path and database table (optional). For a `connection_type` of `s3`, an Amazon S3 path is defined.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password":  
"password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

The `dbtable` property is the name of the JDBC table. For JDBC data stores that support schemas within a database, specify `schema.table-name`. If a schema is not provided, then the default "public" schema is used.

For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

- `format` – A format specification (optional). This is used for an Amazon Simple Storage Service (Amazon S3) or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `format_options` – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.

- `transformation_ctx` – A transformation context to use (optional).

[from_catalog](#)

```
from_catalog(frame, name_space, table_name, redshift_tmp_dir="",
transformation_ctx="")
```

Writes a `DynamicFrame` using the specified catalog database and table name.

- `frame` – The `DynamicFrame` to write.
- `name_space` – The database to use.
- `table_name` – The `table_name` to use.
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – A transformation context to use (optional).

[from_jdbc_conf](#)

```
from_jdbc_conf(frame, catalog_connection, connection_options={},
redshift_tmp_dir = "", transformation_ctx "")
```

Writes a `DynamicFrame` using the specified JDBC connection information.

- `frame` – The `DynamicFrame` to write.
- `catalog_connection` – A catalog connection to use.
- `connection_options` – Connection options, such as path and database table (optional).
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – A transformation context to use (optional).

DynamicFrameReader Class

— Methods —

- [__init__ \(p. 350\)](#)
- [from_rdd \(p. 350\)](#)
- [from_options \(p. 351\)](#)
- [from_catalog \(p. 351\)](#)

__init__

__init__(glue_context)

- `glue_context` – The [GlueContext Class \(p. 352\)](#) to use.

[from_rdd](#)

```
from_rdd(data, name, schema=None, sampleRatio=None)
```

Reads a `DynamicFrame` from a Resilient Distributed Dataset (RDD).

- `data` – The dataset to read from.
- `name` – The name to read from.
- `schema` – The schema to read (optional).
- `sampleRatio` – The sample ratio (optional).

from_options

```
from_options(connection_type, connection_options={}, format=None,
format_options={}, transformation_ctx="")
```

Reads a `DynamicFrame` using the specified connection and format.

- `connection_type` – The connection type. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle`, and `dynamodb`.
- `connection_options` – Connection options, such as path and database table (optional). For a `connection_type` of `s3`, Amazon S3 paths are defined in an array.

```
connection_options = {"paths": [ "s3://mybucket/object_a", "s3://mybucket/object_b"]}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password": "password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

For a JDBC connection that performs parallel reads, you can set the `hashfield` option. For example:

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password": "password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path" , "hashfield": "month"}
```

For more information, see [Reading from JDBC Tables in Parallel \(p. 304\)](#).

- `format` – A format specification (optional). This is used for an Amazon Simple Storage Service (Amazon S3) or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `format_options` – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- `transformation_ctx` – The transformation context to use (optional).

from_catalog

```
from_catalog(name_space, table_name, redshift_tmp_dir="", transformation_ctx="", push_down_predicate="", additional_options={})
```

Reads a `DynamicFrame` using the specified catalog namespace and table name.

- `name_space` – The database to read from.
- `table_name` – The name of the table to read from.
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – The transformation context to use (optional).

- `push_down_predicate` – Filters partitions without having to list and read all the files in your dataset. For more information, see [Pre-Filtering Using Pushdown Predicates \(p. 302\)](#).
- `additional_options` – Additional options provided to AWS Glue. To use a JDBC connection that performs parallel reads, you can set the `hashfield`, `hashexpression`, or `hashpartitions` options. For example:

```
additional_options = {"hashfield": "month"}
```

For more information, see [Reading from JDBC Tables in Parallel \(p. 304\)](#).

GlueContext Class

Wraps the Apache SparkSQL [SQLContext](#) object, and thereby provides mechanisms for interacting with the Apache Spark platform.

Working with Datasets in Amazon S3

- [purge_table \(p. 352\)](#)
- [purge_s3_path \(p. 353\)](#)
- [transition_table \(p. 353\)](#)
- [transition_s3_path \(p. 354\)](#)

`purge_table`

```
purge_table(database, table_name, options={}, transformation_ctx="",
catalog_id=None)
```

Deletes files from Amazon S3 for the specified catalog's database and table. If all files in a partition are deleted, that partition is also deleted from the catalog.

If you want to be able to recover deleted objects, you can enable [object versioning](#) on the Amazon S3 bucket. When an object is deleted from a bucket that doesn't have object versioning enabled, the object can't be recovered. For more information about how to recover deleted objects in a version-enabled bucket, see [How can I retrieve an Amazon S3 object that was deleted?](#) in the AWS Support Knowledge Center.

- `database` – The database to use.
- `table_name` – The name of the table to use.
- `options` – Options to filter files to be deleted and for manifest file generation.
 - `retentionPeriod` – Specifies a period in number of hours to retain files. Files newer than the retention period are retained. Set to 168 hours (7 days) by default.
 - `partitionPredicate` – Partitions satisfying this predicate are deleted. Files within the retention period in these partitions are not deleted. Set to "" – empty by default.
 - `excludeStorageClasses` – Files with storage class in the `excludeStorageClasses` set are not deleted. The default is `Set()` – an empty set.
 - `manifestFilePath` – An optional path for manifest file generation. All files that were successfully purged are recorded in `Success.csv`, and those that failed in `Failed.csv`
- `transformation_ctx` – The transformation context to use (optional). Used in the manifest file path.
- `catalog_id` – The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog). Set to None by default. None defaults to the catalog ID of the calling account in the service.

Example

```
glueContext.purge_table("database", "table", {"partitionPredicate": "(month=='march')",
    "retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://
bucketmanifest/"})
```

[purge_s3_path](#)

purge_s3_path(s3_path, options={}, transformation_ctx="")

Deletes files from the specified Amazon S3 path recursively.

If you want to be able to recover deleted objects, you can enable [object versioning](#) on the Amazon S3 bucket. When an object is deleted from a bucket that doesn't have object versioning enabled, the object can't be recovered. For more information about how to recover deleted objects in a version-enabled bucket, see [How can I retrieve an Amazon S3 object that was deleted?](#) in the AWS Support Knowledge Center.

- **s3_path** – The path in Amazon S3 of the files to be deleted in the format `s3://<bucket>/<prefix>/`
- **options** – Options to filter files to be deleted and for manifest file generation.
 - **retentionPeriod** – Specifies a period in number of hours to retain files. Files newer than the retention period are retained. Set to 168 hours (7 days) by default.
 - **partitionPredicate** – Partitions satisfying this predicate are deleted. Files within the retention period in these partitions are not deleted. Set to "" – empty by default.
 - **excludeStorageClasses** – Files with storage class in the excludeStorageClasses set are not deleted. The default is `Set()` – an empty set.
 - **manifestFilePath** – An optional path for manifest file generation. All files that were successfully purged are recorded in `Success.csv`, and those that failed in `Failed.csv`
 - **transformation_ctx** – The transformation context to use (optional). Used in the manifest file path.
 - **catalog_id** – The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog). Set to None by default. None defaults to the catalog ID of the calling account in the service.

Example

```
glueContext.purge_s3_path("s3://bucket/path/", {"retentionPeriod": 1,
    "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/"})
```

[transition_table](#)

transition_table(database, table_name, transition_to, options={}, transformation_ctx="", catalog_id=None)

Transitions the storage class of the files stored on Amazon S3 for the specified catalog's database and table.

You can transition between any two storage classes. For the `GLACIER` and `DEEP_ARCHIVE` storage classes, you can transition to these classes. However, you would use an `S3 RESTORE` to transition from `GLACIER` and `DEEP_ARCHIVE` storage classes.

If you're running AWS Glue ETL jobs that read files or partitions from Amazon S3, you can exclude some Amazon S3 storage class types. For more information, see [Excluding Amazon S3 Storage Classes](#).

- **database** – The database to use.

- `table_name` – The name of the table to use.
- `transition_to` – The [Amazon S3 storage class](#) to transition to.
- `options` – Options to filter files to be deleted and for manifest file generation.
 - `retentionPeriod` – Specifies a period in number of hours to retain files. Files newer than the retention period are retained. Set to 168 hours (7 days) by default.
 - `partitionPredicate` – Partitions satisfying this predicate are transitioned. Files within the retention period in these partitions are not transitioned. Set to "" – empty by default.
 - `excludeStorageClasses` – Files with storage class in the `excludeStorageClasses` set are not transitioned. The default is `Set()` – an empty set.
 - `manifestFilePath` – An optional path for manifest file generation. All files that were successfully transitioned are recorded in `Success.csv`, and those that failed in `Failed.csv`
 - `accountId` – The AWS account ID to run the transition transform. Mandatory for this transform.
 - `roleArn` – The AWS role to run the transition transform. Mandatory for this transform.
- `transformation_ctx` – The transformation context to use (optional). Used in the manifest file path.
- `catalog_id` – The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog). Set to `None` by default. None defaults to the catalog ID of the calling account in the service.

Example

```
glueContext.transition_table("database", "table", "STANDARD_IA", {"retentionPeriod": 1,
"excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/",
"accountId": "12345678901", "roleArn": "arn:aws:iam::123456789012:user/example-username"})
```

[transition_s3_path](#)

```
transition_s3_path(s3_path, transition_to, options={}, transformation_ctx="")
```

Transitions the storage class of the files in the specified Amazon S3 path recursively.

You can transition between any two storage classes. For the `GLACIER` and `DEEP_ARCHIVE` storage classes, you can transition to these classes. However, you would use an `S3 RESTORE` to transition from `GLACIER` and `DEEP_ARCHIVE` storage classes.

If you're running AWS Glue ETL jobs that read files or partitions from Amazon S3, you can exclude some Amazon S3 storage class types. For more information, see [Excluding Amazon S3 Storage Classes](#).

- `s3_path` – The path in Amazon S3 of the files to be transitioned in the format `s3://<bucket>/<prefix>/`
- `transition_to` – The [Amazon S3 storage class](#) to transition to.
- `options` – Options to filter files to be deleted and for manifest file generation.
 - `retentionPeriod` – Specifies a period in number of hours to retain files. Files newer than the retention period are retained. Set to 168 hours (7 days) by default.
 - `partitionPredicate` – Partitions satisfying this predicate are transitioned. Files within the retention period in these partitions are not transitioned. Set to "" – empty by default.
 - `excludeStorageClasses` – Files with storage class in the `excludeStorageClasses` set are not transitioned. The default is `Set()` – an empty set.
 - `manifestFilePath` – An optional path for manifest file generation. All files that were successfully transitioned are recorded in `Success.csv`, and those that failed in `Failed.csv`
 - `accountId` – The AWS account ID to run the transition transform. Mandatory for this transform.
 - `roleArn` – The AWS role to run the transition transform. Mandatory for this transform.
- `transformation_ctx` – The transformation context to use (optional). Used in the manifest file path.

Example

```
glueContext.transition_s3_path("s3://bucket/prefix/", "STANDARD_IA", {"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/", "accountId": "12345678901", "roleArn": "arn:aws:iam::123456789012:user/example-username"})
```

Creating

- [__init__ \(p. 355\)](#)
- [getSource \(p. 355\)](#)
- [create_dynamic_frame_from_rdd \(p. 355\)](#)
- [create_dynamic_frame_from_catalog \(p. 356\)](#)
- [create_dynamic_frame_from_options \(p. 356\)](#)

[__init__](#)

[__init__\(sparkContext\)](#)

- `sparkContext` – The Apache Spark context to use.

[getSource](#)

```
getSource(connection_type, transformation_ctx = "", **options)
```

Creates a `DataSource` object that can be used to read `DynamicFrames` from external sources.

- `connection_type` – The connection type to use, such as Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and JDBC. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle`, and `dynamodb`.
- `transformation_ctx` – The transformation context to use (optional).
- `options` – A collection of optional name-value pairs. For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

The following is an example of using `getSource`.

```
>>> data_source = context.getSource("file", paths=["/in/path"])
>>> data_source.setFormat("json")
>>> myFrame = data_source.getFrame()
```

[create_dynamic_frame_from_rdd](#)

```
create_dynamic_frame_from_rdd(data, name, schema=None, sample_ratio=None, transformation_ctx="")
```

Returns a `DynamicFrame` that is created from an Apache Spark Resilient Distributed Dataset (RDD).

- `data` – The data source to use.
- `name` – The name of the data to use.
- `schema` – The schema to use (optional).
- `sample_ratio` – The sample ratio to use (optional).
- `transformation_ctx` – The transformation context to use (optional).

[create_dynamic_frame_from_catalog](#)

```
create_dynamic_frame_from_catalog(database, table_name, redshift_tmp_dir,
transformation_ctx = "", push_down_predicate= "", additional_options =
{}, catalog_id = None)
```

Returns a DynamicFrame that is created using a catalog database and table name.

- Database – The database to read from.
- table_name – The name of the table to read from.
- redshift_tmp_dir – An Amazon Redshift temporary directory to use (optional).
- transformation_ctx – The transformation context to use (optional).
- push_down_predicate – Filters partitions without having to list and read all the files in your dataset. For more information, see [Pre-Filtering Using Pushdown Predicates \(p. 302\)](#).
- additional_options – Additional options provided to AWS Glue.
- catalog_id — The catalog ID (account ID) of the Data Catalog being accessed. When None, the default account ID of the caller is used.

[create_dynamic_frame_from_options](#)

```
create_dynamic_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")
```

Returns a DynamicFrame created with the specified connection and format.

- connection_type – The connection type, such as Amazon S3, Amazon Redshift, and JDBC. Valid values include s3, mysql, postgresql, redshift, sqlserver, oracle, and dynamodb.
- connection_options – Connection options, such as paths and database table (optional). For a connection_type of s3, a list of Amazon S3 paths is defined.

```
connection_options = {"paths": ["s3://aws-glue-target/temp"]}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password": "password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

The dbtable property is the name of the JDBC table. For JDBC data stores that support schemas within a database, specify schema.table-name. If a schema is not provided, then the default "public" schema is used.

For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

- format – A format specification (optional). This is used for an Amazon S3 or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- format_options – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- transformation_ctx – The transformation context to use (optional).

Writing

- [getSink \(p. 357\)](#)

- [write_dynamic_frame_from_options \(p. 357\)](#)
- [write_from_options \(p. 358\)](#)
- [write_dynamic_frame_from_catalog \(p. 358\)](#)
- [write_dynamic_frame_from_jdbc_conf \(p. 359\)](#)
- [write_from_jdbc_conf \(p. 359\)](#)

getSink

```
getSink(connection_type, format = None, transformation_ctx = "",
**options)
```

Gets a DataSink object that can be used to write DynamicFrames to external sources. Check the SparkSQL format first to be sure to get the expected sink.

- **connection_type** – The connection type to use, such as Amazon S3, Amazon Redshift, and JDBC. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, and `oracle`.
- **format** – The SparkSQL format to use (optional).
- **transformation_ctx** – The transformation context to use (optional).
- **options** – A collection of option name-value pairs.

For example:

```
>>> data_sink = context.getSink("s3")
>>> data_sink.setFormat("json"),
>>> data_sink.writeFrame(myFrame)
```

write_dynamic_frame_from_options

```
write_dynamic_frame_from_options(frame, connection_type,
connection_options={}, format=None, format_options={}, transformation_ctx
= "")
```

Writes and returns a DynamicFrame using the specified connection and format.

- **frame** – The DynamicFrame to write.
- **connection_type** – The connection type, such as Amazon S3, Amazon Redshift, and JDBC. Valid values include `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, and `oracle`.
- **connection_options** – Connection options, such as path and database table (optional). For a `connection_type` of `s3`, an Amazon S3 path is defined.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password": "password",
"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

The `dbtable` property is the name of the JDBC table. For JDBC data stores that support schemas within a database, specify `schema.table-name`. If a schema is not provided, then the default "public" schema is used.

For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

- **format** – A format specification (optional). This is used for an Amazon S3 or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- **format_options** – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- **transformation_ctx** – A transformation context to use (optional).

[write_from_options](#)

```
write_from_options(frame_or_dfc, connection_type, connection_options={},
format={}, format_options={}, transformation_ctx = "")
```

Writes and returns a DynamicFrame or DynamicFrameCollection that is created with the specified connection and format information.

- **frame_or_dfc** – The DynamicFrame or DynamicFrameCollection to write.
- **connection_type** – The connection type, such as Amazon S3, Amazon Redshift, and JDBC. Valid values include s3, mysql, postgresql, redshift, sqlserver, and oracle.
- **connection_options** – Connection options, such as path and database table (optional). For a connection_type of s3, an Amazon S3 path is defined.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

For JDBC connections, several properties must be defined. Note that the database name must be part of the URL. It can optionally be included in the connection options.

```
connection_options = {"url": "jdbc-url/database", "user": "username", "password":  
"password", "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

The dbtable property is the name of the JDBC table. For JDBC data stores that support schemas within a database, specify schema.table-name. If a schema is not provided, then the default "public" schema is used.

For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).

- **format** – A format specification (optional). This is used for an Amazon S3 or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- **format_options** – Format options for the specified format. See [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 299\)](#) for the formats that are supported.
- **transformation_ctx** – A transformation context to use (optional).

[write_dynamic_frame_from_catalog](#)

```
write_dynamic_frame_from_catalog(frame, database, table_name,
redshift_tmp_dir, transformation_ctx = "", additional_options = {},
catalog_id = None)
```

Writes and returns a DynamicFrame using a catalog database and a table name.

- **frame** – The DynamicFrame to write.
- **Database** – The database to read from.
- **table_name** – The name of the table to read from.

- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – The transformation context to use (optional).
- `catalog_id` — The catalog ID (account ID) of the Data Catalog being accessed. When None, the default account ID of the caller is used.

[write_dynamic_frame_from_jdbc_conf](#)

```
write_dynamic_frame_from_jdbc_conf(frame, catalog_connection,
connection_options={}, redshift_tmp_dir = "", transformation_ctx = "",
catalog_id = None)
```

Writes and returns a `DynamicFrame` using the specified JDBC connection information.

- `frame` – The `DynamicFrame` to write.
- `catalog_connection` – A catalog connection to use.
- `connection_options` – Connection options, such as path and database table (optional). For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – A transformation context to use (optional).
- `catalog_id` — The catalog ID (account ID) of the Data Catalog being accessed. When None, the default account ID of the caller is used.

[write_from_jdbc_conf](#)

```
write_from_jdbc_conf(frame_or_dfc, catalog_connection,
connection_options={}, redshift_tmp_dir = "", transformation_ctx = "",
catalog_id = None)
```

Writes and returns a `DynamicFrame` or `DynamicFrameCollection` using the specified JDBC connection information.

- `frame_or_dfc` – The `DynamicFrame` or `DynamicFrameCollection` to write.
- `catalog_connection` – A catalog connection to use.
- `connection_options` – Connection options, such as path and database table (optional). For more information, see [Connection Types and Options for ETL in AWS Glue \(p. 294\)](#).
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – A transformation context to use (optional).
- `catalog_id` — The catalog ID (account ID) of the Data Catalog being accessed. When None, the default account ID of the caller is used.

[Extracting](#)

- [extract_jdbc_conf \(p. 359\)](#)

[extract_jdbc_conf](#)

```
extract_jdbc_conf(connection_name, catalog_id = None)
```

Returns a dict with keys `user`, `password`, `vendor`, and `url` from the connection object in the Data Catalog.

- `connection_name` – The name of the connection in the Data Catalog

- `catalog_id` — The catalog ID (account ID) of the Data Catalog being accessed. When None, the default account ID of the caller is used.

AWS Glue PySpark Transforms Reference

AWS Glue has created the following transform Classes to use in PySpark ETL operations.

- [GlueTransform Base Class \(p. 360\)](#)
- [ApplyMapping Class \(p. 362\)](#)
- [DropFields Class \(p. 363\)](#)
- [DropNullFields Class \(p. 364\)](#)
- [ErrorsAsDynamicFrame Class \(p. 366\)](#)
- [Filter Class \(p. 367\)](#)
- [Join Class \(p. 370\)](#)
- [Map Class \(p. 371\)](#)
- [MapToCollection Class \(p. 374\)](#)
- [mergeDynamicFrame \(p. 342\)](#)
- [Relationalize Class \(p. 375\)](#)
- [RenameField Class \(p. 376\)](#)
- [ResolveChoice Class \(p. 377\)](#)
- [SelectFields Class \(p. 379\)](#)
- [SelectFromCollection Class \(p. 380\)](#)
- [Spigot Class \(p. 381\)](#)
- [SplitFields Class \(p. 382\)](#)
- [SplitRows Class \(p. 384\)](#)
- [Unbox Class \(p. 385\)](#)
- [UnnestFrame Class \(p. 386\)](#)

GlueTransform Base Class

The base class that all the `awsglue.transforms` classes inherit from.

The classes all define a `__call__` method. They either override the `GlueTransform` class methods listed in the following sections, or they are called using the class name by default.

Methods

- [apply\(cls, *args, **kwargs\) \(p. 360\)](#)
- [name\(cls\) \(p. 361\)](#)
- [describeArgs\(cls\) \(p. 361\)](#)
- [describeReturn\(cls\) \(p. 361\)](#)
- [describeTransform\(cls\) \(p. 361\)](#)
- [describeErrors\(cls\) \(p. 361\)](#)
- [describe\(cls\) \(p. 362\)](#)

`apply(cls, *args, **kwargs)`

Applies the transform by calling the transform class, and returns the result.

- `cls` – The `self` class object.

`name(cls)`

Returns the name of the derived transform class.

- `cls` – The `self` class object.

`describeArgs(cls)`

- `cls` – The `self` class object.

Returns a list of dictionaries, each corresponding to a named argument, in the following format:

```
[  
  {  
    "name": "(name of argument)",  
    "type": "(type of argument)",  
    "description": "(description of argument)",  
    "optional": "(Boolean, True if the argument is optional)",  
    "defaultValue": "(Default value string, or None)(String; the default value, or None)"  
  },  
  ...  
]
```

Raises a `NotImplementedError` exception when called in a derived transform where it is not implemented.

`describeReturn(cls)`

- `cls` – The `self` class object.

Returns a dictionary with information about the return type, in the following format:

```
{  
  "type": "(return type)",  
  "description": "(description of output)"  
}
```

Raises a `NotImplementedError` exception when called in a derived transform where it is not implemented.

`describeTransform(cls)`

Returns a string describing the transform.

- `cls` – The `self` class object.

Raises a `NotImplementedError` exception when called in a derived transform where it is not implemented.

`describeErrors(cls)`

- `cls` – The `self` class object.

Returns a list of dictionaries, each describing a possible exception thrown by this transform, in the following format:

```
[  
  {  
    "type": "(type of error)",  
    "description": "(description of error)"  
  },  
  ...  
]
```

describe(cls)

- `cls` – The `self` class object.

Returns an object with the following format:

```
{  
  "transform" : {  
    "name" : cls.name( ),  
    "args" : cls.describeArgs( ),  
    "returns" : cls.describeReturn( ),  
    "raises" : cls.describeErrors( ),  
    "location" : "internal"  
  }  
}
```

ApplyMapping Class

Applies a mapping in a `DynamicFrame`.

Methods

- [__call__ \(p. 362\)](#)
- [apply \(p. 363\)](#)
- [name \(p. 363\)](#)
- [describeArgs \(p. 363\)](#)
- [describeReturn \(p. 363\)](#)
- [describeTransform \(p. 363\)](#)
- [describeErrors \(p. 363\)](#)
- [describe \(p. 363\)](#)

`__call__(frame, mappings, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

Applies a declarative mapping to a specified `DynamicFrame`.

- `frame` – The `DynamicFrame` in which to apply the mapping (required).
- `mappings` – A list of mapping tuples, each consisting of: (source column, source type, target column, target type). Required.

If the source column has dots in it, the mapping will not work unless you place back-ticks around it (` `). For example, to map `this.old.name` (string) to `thisNewName` (string), you would use the following tuple:

```
("`this.old.name`", "string", "thisNewName", "string")
```

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns only the fields of the `DynamicFrame` specified in the "mapping" tuples.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[DropFields Class](#)

Drops fields within a `DynamicFrame`.

Methods

- [__call__ \(p. 364\)](#)
- [apply \(p. 364\)](#)
- [name \(p. 364\)](#)
- [describeArgs \(p. 364\)](#)
- [describeReturn \(p. 364\)](#)
- [describeTransform \(p. 364\)](#)
- [describeErrors \(p. 364\)](#)
- [describe \(p. 364\)](#)

[__call__\(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0\)](#)

Drops nodes within a `DynamicFrame`.

- `frame` – The `DynamicFrame` in which to drop the nodes (required).
- `paths` – A list of full paths to the nodes to drop (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a new `DynamicFrame` without the specified fields.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[DropNullFields Class](#)

Drops all null fields in a `DynamicFrame` whose type is `NullType`. These are fields with missing or null values in every record in the `DynamicFrame` data set.

Methods

- [__call__ \(p. 365\)](#)
- [apply \(p. 365\)](#)
- [name \(p. 365\)](#)

- [describeArgs \(p. 365\)](#)
- [describeReturn \(p. 365\)](#)
- [describeTransform \(p. 365\)](#)
- [describeErrors \(p. 365\)](#)
- [describe \(p. 365\)](#)

[`__call__\(frame, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0\)`](#)

Drops all null fields in a `DynamicFrame` whose type is `NullType`. These are fields with missing or null values in every record in the `DynamicFrame` data set.

- `frame` – The `DynamicFrame` in which to drop null fields (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a new `DynamicFrame` with no null fields.

[`apply\(cls, *args, **kwargs\)`](#)

- `cls` – `cls`

[`name\(cls\)`](#)

- `cls` – `cls`

[`describeArgs\(cls\)`](#)

- `cls` – `cls`

[`describeReturn\(cls\)`](#)

- `cls` – `cls`

[`describeTransform\(cls\)`](#)

- `cls` – `cls`

[`describeErrors\(cls\)`](#)

- `cls` – `cls`

[`describe\(cls\)`](#)

- `cls` – `cls`

ErrorsAsDynamicFrame Class

Returns a `DynamicFrame` that contains nested error records leading up to the creation of the source `DynamicFrame`.

Methods

- `__call__(frame)`
- `apply` (p. 366)
- `name` (p. 366)
- `describeArgs` (p. 366)
- `describeReturn` (p. 366)
- `describeTransform` (p. 366)
- `describeErrors` (p. 366)
- `describe` (p. 366)

`__call__(frame)`

Returns a `DynamicFrame` that contains nested error records relating to the source `DynamicFrame`.

- `frame` – The source `DynamicFrame` (required).

`apply(cls, *args, **kwargs)`

- `cls` – `cls`

`name(cls)`

- `cls` – `cls`

`describeArgs(cls)`

- `cls` – `cls`

`describeReturn(cls)`

- `cls` – `cls`

`describeTransform(cls)`

- `cls` – `cls`

`describeErrors(cls)`

- `cls` – `cls`

`describe(cls)`

- `cls` – `cls`

Filter Class

Builds a new `DynamicFrame` by selecting records from the input `DynamicFrame` that satisfy a specified predicate function.

Methods

- [__call__ \(p. 367\)](#)
- [apply \(p. 367\)](#)
- [name \(p. 367\)](#)
- [describeArgs \(p. 367\)](#)
- [describeReturn \(p. 367\)](#)
- [describeTransform \(p. 368\)](#)
- [describeErrors \(p. 368\)](#)
- [describe \(p. 368\)](#)
- [Example Code \(p. 368\)](#)

`__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

Returns a new `DynamicFrame` built by selecting records from the input `DynamicFrame` that satisfy a specified predicate function.

- `frame` – The source `DynamicFrame` to apply the specified filter function to (required).
- `f` – The predicate function to apply to each `DynamicRecord` in the `DynamicFrame`. The function must take a `DynamicRecord` as its argument and return `True` if the `DynamicRecord` meets the filter requirements, or `False` if it does not (required).

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in a Spark `DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

`apply(cls, *args, **kwargs)`

Inherited from `GlueTransform` [apply \(p. 360\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 361\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

describeTransform(cls)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

describeErrors(cls)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

describe(cls)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

AWS Glue Python Example

This example filters sample data using the `Filter` transform and a simple Lambda function. The dataset used here consists of Medicare Provider payment data downloaded from two `Data.CMS.gov` sites: [Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011](#), and [Inpatient Charge Data FY 2011](#).

After downloading the sample data, we modified it to introduce a couple of erroneous records at the end of the file. This modified file is located in a public Amazon S3 bucket at `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`. For another example that uses this dataset, see [Code Example: Data Preparation Using ResolveChoice, Lambda, and ApplyMapping \(p. 327\)](#).

Begin by creating a `DynamicFrame` for the data:

```
%pyspark
from awsglue.context import GlueContext
from awsglue.transforms import *
from pyspark.context import SparkContext

glueContext = GlueContext(SparkContext.getOrCreate())

dyF = glueContext.create_dynamic_frame.from_options(
    's3',
    {'paths': ['s3://awsglue-datasets/examples/medicare/
Medicare_Hospital_Provider.csv']},
    'csv',
    {'withHeader': True})

print "Full record count: ", dyF.count()
dyF.printSchema()
```

The output should be as follows:

```
Full record count:  163065L
root
|--- DRG Definition: string
|--- Provider Id: string
|--- Provider Name: string
|--- Provider Street Address: string
|--- Provider City: string
|--- Provider State: string
|--- Provider Zip Code: string
|--- Hospital Referral Region Description: string
|--- Total Discharges: string
|--- Average Covered Charges: string
|--- Average Total Payments: string
|--- Average Medicare Payments: string
```

Next, use the `Filter` transform to condense the dataset, retaining only those entries that are from Sacramento, California, or from Montgomery, Alabama. The filter transform works with any filter function that takes a `DynamicRecord` as input and returns True if the `DynamicRecord` meets the filter requirements, or False if not.

Note

You can use Python's dot notation to access many fields in a `DynamicRecord`. For example, you can access the `column_A` field in `dynamic_record_X` as: `dynamic_record_X.column_A`. However, this technique doesn't work with field names that contain anything besides alphanumeric characters and underscores. For fields that contain other characters, such as spaces or periods, you must fall back to Python's dictionary notation. For example, to access a field named `col-B`, use: `dynamic_record_X["col-B"]`.

You can use a simple Lambda function with the `Filter` transform to remove all `DynamicRecords` that don't originate in Sacramento or Montgomery. To confirm that this worked, print out the number of records that remain:

```
sac_or_mon_dyF = Filter.apply(frame = dyF,
                               f = lambda x: x["Provider State"] in ["CA", "AL"] and
                               x["Provider City"] in ["SACRAMENTO", "MONTGOMERY"])
print "Filtered record count: ", sac_or_mon_dyF.count()
```

The output that you get looks like the following:

```
Filtered record count: 564L
```

FlatMap Class

Applies a transform to each `DynamicFrame` in a collection and flattens the results.

Methods

- [__call__ \(p. 369\)](#)
- [apply \(p. 370\)](#)
- [name \(p. 370\)](#)
- [describeArgs \(p. 370\)](#)
- [describeReturn \(p. 370\)](#)
- [describeTransform \(p. 370\)](#)
- [describeErrors \(p. 370\)](#)
- [describe \(p. 370\)](#)

`__call__(dfc, BaseTransform, frame_name, transformation_ctx = "",
**base_kwargs)`

Applies a transform to each `DynamicFrame` in a collection and flattens the results.

- `dfc` – The `DynamicFrameCollection` over which to flatmap (required).
- `BaseTransform` – A transform derived from `GlueTransform` to apply to each member of the collection (required).
- `frame_name` – The argument name to pass the elements of the collection to (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `base_kwargs` – Arguments to pass to the base transform (required).

Returns a new `DynamicFrameCollection` created by applying the transform to each `DynamicFrame` in the source `DynamicFrameCollection`.

[apply\(`cls`, *`args`, **`kwargs`\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(`cls`\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(`cls`\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(`cls`\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(`cls`\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(`cls`\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(`cls`\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

Join Class

Performs an equality join on two `DynamicFrames`.

Methods

- [__call__ \(p. 370\)](#)
- [apply \(p. 371\)](#)
- [name \(p. 371\)](#)
- [describeArgs \(p. 371\)](#)
- [describeReturn \(p. 371\)](#)
- [describeTransform \(p. 371\)](#)
- [describeErrors \(p. 371\)](#)
- [describe \(p. 371\)](#)

[__call__\(`frame1`, `frame2`, `keys1`, `keys2`, `transformation_ctx = ""`\)](#)

Performs an equality join on two `DynamicFrames`.

- `frame1` – The first `DynamicFrame` to join (required).
- `frame2` – The second `DynamicFrame` to join (required).
- `keys1` – The keys to join on for the first frame (required).
- `keys2` – The keys to join on for the second frame (required).

- `transformation_ctx` – A unique string that is used to identify state information (optional).

Returns a new `DynamicFrame` obtained by joining the two `DynamicFrames`.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#)

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#)

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#)

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#)

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#)

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#)

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#)

Map Class

Builds a new `DynamicFrame` by applying a function to all records in the input `DynamicFrame`.

Methods

- [__call__ \(p. 371\)](#)
- [apply \(p. 372\)](#)
- [name \(p. 372\)](#)
- [describeArgs \(p. 372\)](#)
- [describeReturn \(p. 372\)](#)
- [describeTransform \(p. 372\)](#)
- [describeErrors \(p. 372\)](#)
- [describe \(p. 372\)](#)
- [Example Code \(p. 372\)](#)

[__call__\(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0\)](#)

Returns a new `DynamicFrame` that results from applying the specified function to all `DynamicRecords` in the original `DynamicFrame`.

- `frame` – The original `DynamicFrame` to which to apply the mapping function (required).
- `f` – The function to apply to all `DynamicRecords` in the `DynamicFrame`. The function must take a `DynamicRecord` as an argument and return a new `DynamicRecord` produced by the mapping (required).

A `DynamicRecord` represents a logical record in a `DynamicFrame`. It is similar to a row in an Apache Spark `DataFrame`, except that it is self-describing and can be used for data that does not conform to a fixed schema.

- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a new `DynamicFrame` that results from applying the specified function to all `DynamicRecords` in the original `DynamicFrame`.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[AWS Glue Python Example](#)

This example uses the `Map` transform to merge several fields into one `struct` type. The dataset that is used here consists of Medicare Provider payment data downloaded from two `Data.CMS.gov` sites: [Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011](#), and [Inpatient Charge Data FY 2011](#).

After downloading the sample data, we modified it to introduce a couple of erroneous records at the end of the file. This modified file is located in a public Amazon S3 bucket at `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`. For another example

that uses this dataset, see [Code Example: Data Preparation Using ResolveChoice, Lambda, and ApplyMapping \(p. 327\)](#).

Begin by creating a `DynamicFrame` for the data:

```
from awsglue.context import GlueContext
from awsglue.transforms import *
from pyspark.context import SparkContext

glueContext = GlueContext(SparkContext.getOrCreate())

dyF = glueContext.create_dynamic_frame.from_options(
    's3',
    {'paths': ['s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv']},
    'csv',
    {'withHeader': True})

print "Full record count: ", dyF.count()
dyF.printSchema()
```

The output of this code should be as follows:

```
Full record count: 163065L
root
|-- DRG Definition: string
|-- Provider Id: string
|-- Provider Name: string
|-- Provider Street Address: string
|-- Provider City: string
|-- Provider State: string
|-- Provider Zip Code: string
|-- Hospital Referral Region Description: string
|-- Total Discharges: string
|-- Average Covered Charges: string
|-- Average Total Payments: string
|-- Average Medicare Payments: string
```

Next, create a mapping function to merge provider-address fields in a `DynamicRecord` into a `struct`, and then delete the individual address fields:

```
def MergeAddress(rec):
    rec["Address"] = {}
    rec["Address"]["Street"] = rec["Provider Street Address"]
    rec["Address"]["City"] = rec["Provider City"]
    rec["Address"]["State"] = rec["Provider State"]
    rec["Address"]["Zip.Code"] = rec["Provider Zip Code"]
    rec["Address"]["Array"] = [rec["Provider Street Address"], rec["Provider City"],
    rec["Provider State"], rec["Provider Zip Code"]]
    del rec["Provider Street Address"]
    del rec["Provider City"]
    del rec["Provider State"]
    del rec["Provider Zip Code"]
    return rec
```

In this mapping function, the line `rec["Address"] = {}` creates a dictionary in the input `DynamicRecord` that contains the new structure.

Note

Python map fields are *not* supported here. For example, you can't have a line like the following:
`rec["Addresses"] = [] # ILLEGAL!`

The lines that are like `rec["Address"]["Street"] = rec["Provider Street Address"]` add fields to the new structure using Python dictionary syntax.

After the address lines are added to the new structure, the lines that are like `del rec["Provider Street Address"]` remove the individual fields from the `DynamicRecord`.

Now you can use the `Map` transform to apply your mapping function to all `DynamicRecords` in the `DynamicFrame`.

```
mapped_dyF = Map.apply(frame = dyF, f = MergeAddress)
mapped_dyF.printSchema()
```

The output is as follows:

```
root
|--- Average Total Payments: string
|--- Average Covered Charges: string
|--- DRG Definition: string
|--- Average Medicare Payments: string
|--- Hospital Referral Region Description: string
|--- Address: struct
|   |--- Zip.Code: string
|   |--- City: string
|   |--- Array: array
|   |   |--- element: string
|   |--- State: string
|   |--- Street: string
|--- Provider Id: string
|--- Total Discharges: string
|--- Provider Name: string
```

MapToCollection Class

Applies a transform to each `DynamicFrame` in the specified `DynamicFrameCollection`.

Methods

- [__call__ \(p. 374\)](#)
- [apply \(p. 375\)](#)
- [name \(p. 375\)](#)
- [describeArgs \(p. 375\)](#)
- [describeReturn \(p. 375\)](#)
- [describeTransform \(p. 375\)](#)
- [describeErrors \(p. 375\)](#)
- [describe \(p. 375\)](#)

`__call__(dfc, BaseTransform, frame_name, transformation_ctx = "",
**base_kwargs)`

Applies a transform function to each `DynamicFrame` in the specified `DynamicFrameCollection`.

- `dfc` – The `DynamicFrameCollection` over which to apply the transform function (required).
- `callable` – A callable transform function to apply to each member of the collection (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).

Returns a new `DynamicFrameCollection` created by applying the transform to each `DynamicFrame` in the source `DynamicFrameCollection`.

[apply\(`cls`, *`args`, **`kwargs`\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#)

[name\(`cls`\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(`cls`\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(`cls`\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(`cls`\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(`cls`\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(`cls`\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[Relationalize Class](#)

Flattens nested schema in a `DynamicFrame` and pivots out array columns from the flattened frame.

[Methods](#)

- [__call__ \(p. 375\)](#)
- [apply \(p. 376\)](#)
- [name \(p. 376\)](#)
- [describeArgs \(p. 376\)](#)
- [describeReturn \(p. 376\)](#)
- [describeTransform \(p. 376\)](#)
- [describeErrors \(p. 376\)](#)
- [describe \(p. 376\)](#)

[__call__\(`frame`, `staging_path=None`, `name='roottable'`, `options=None`, `transformation_ctx = ""`, `info = ""`, `stageThreshold = 0`, `totalThreshold = 0`\)](#)

Relationalizes a `DynamicFrame` and produces a list of frames that are generated by unnesting nested columns and pivoting array columns. The pivoted array column can be joined to the root table using the `joinkey` generated in the unnest phase.

- `frame` – The `DynamicFrame` to relationalize (required).
- `staging_path` – The path at which to store partitions of pivoted tables in CSV format (optional). Pivoted tables are read back from this path.

- `name` – The name of the root table (optional).
- `options` – A dictionary of optional parameters.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Return a `DynamicFrameCollection` containing the `DynamicFrames` produced by from the `relationalize` operation.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

RenameField Class

Renames a node within a `DynamicFrame`.

Methods

- [__call__ \(p. 377\)](#)
- [apply \(p. 377\)](#)
- [name \(p. 377\)](#)
- [describeArgs \(p. 377\)](#)
- [describeReturn \(p. 377\)](#)
- [describeTransform \(p. 377\)](#)
- [describeErrors \(p. 377\)](#)
- [describe \(p. 377\)](#)

`__call__(frame, old_name, new_name, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

Renames a node within a `DynamicFrame`.

- `frame` – The `DynamicFrame` in which to rename a node (required).
- `old_name` – Full path to the node to rename (required).

If the old name has dots in it, `RenameField` will not work unless you place back-ticks around it (` `). For example, to replace `this.old.name` with `thisNewName`, you would call `RenameField` as follows:

```
newDyF = RenameField(oldDyF, "`this.old.name`", "thisNewName")
```

- `new_name` – New name, including full path (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a `DynamicFrame` with the specified field renamed.

`apply(cls, *args, **kwargs)`

Inherited from `GlueTransform` [apply \(p. 360\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 361\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 362\)](#).

ResolveChoice Class

Resolves a choice type within a `DynamicFrame`.

Methods

- [__call__ \(p. 378\)](#)
- [apply \(p. 379\)](#)
- [name \(p. 379\)](#)
- [describeArgs \(p. 379\)](#)
- [describeReturn \(p. 379\)](#)
- [describeTransform \(p. 379\)](#)
- [describeErrors \(p. 379\)](#)
- [describe \(p. 379\)](#)

`__call__(frame, specs = None, choice = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

Provides information for resolving ambiguous types within a `DynamicFrame`. Returns the resulting `DynamicFrame`.

- `frame` – The `DynamicFrame` in which to resolve the choice type (required).
- `specs` – A list of specific ambiguities to resolve, each in the form of a tuple: `(path, action)`. The path value identifies a specific ambiguous element, and the action value identifies the corresponding resolution. Only one of the spec and choice parameters can be used. If the spec parameter is not `None`, then the choice parameter must be an empty string. Conversely if the choice is not an empty string, then the spec parameter must be `None`. If neither parameter is provided, AWS Glue tries to parse the schema and use it to resolve ambiguities.

The action portion of a specs tuple can specify one of four resolution strategies:

- `cast`: Allows you to specify a type to cast to (for example, `cast:int`).
- `make_cols`: Resolves a potential ambiguity by flattening the data. For example, if `columnA` could be an `int` or a `string`, the resolution is to produce two columns named `columnA_int` and `columnA_string` in the resulting `DynamicFrame`.
- `make_struct`: Resolves a potential ambiguity by using a struct to represent the data. For example, if data in a column could be an `int` or a `string`, using the `make_struct` action produces a column of structures in the resulting `DynamicFrame` with each containing both an `int` and a `string`.
- `project`: Resolves a potential ambiguity by retaining only values of a specified type in the resulting `DynamicFrame`. For example, if data in a `ChoiceType` column could be an `int` or a `string`, specifying a `project:string` action drops columns from the resulting `DynamicFrame` which are not type `string`.

If the path identifies an array, place empty square brackets after the name of the array to avoid ambiguity. For example, suppose you are working with data structured as follows:

```
"myList": [
    { "price": 100.00 },
    { "price": "$100.00" }
]
```

You can select the numeric rather than the string version of the price by setting the path to `"myList[].price"`, and the action to `"cast:double"`.

- `choice` – The default resolution action if the `specs` parameter is `None`. If the `specs` parameter is not `None`, then this must not be set to anything but an empty string.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).

- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a `DynamicFrame` with the resolved choice.

Example

```
df1 = ResolveChoice.apply(df, choice = "make_cols")
df2 = ResolveChoice.apply(df, specs = [("a.b", "make_struct"), ("c.d", "cast:double")])
```

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

SelectFields Class

Gets fields in a `DynamicFrame`.

Methods

- [__call__ \(p. 380\)](#)
- [apply \(p. 380\)](#)
- [name \(p. 380\)](#)
- [describeArgs \(p. 380\)](#)
- [describeReturn \(p. 380\)](#)
- [describeTransform \(p. 380\)](#)
- [describeErrors \(p. 380\)](#)

- [describe \(p. 380\)](#)

`__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

Gets fields (nodes) in a `DynamicFrame`.

- `frame` – The `DynamicFrame` in which to select fields (required).
- `paths` – A list of full paths to the fields to select (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a new `DynamicFrame` containing only the specified fields.

[`apply\(cls, *args, **kwargs\)`](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[`name\(cls\)`](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[`describeArgs\(cls\)`](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[`describeReturn\(cls\)`](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[`describeTransform\(cls\)`](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[`describeErrors\(cls\)`](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[`describe\(cls\)`](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

SelectFromCollection Class

Selects one `DynamicFrame` in a `DynamicFrameCollection`.

Methods

- [__call__ \(p. 381\)](#)

- [apply \(p. 381\)](#)
- [name \(p. 381\)](#)
- [describeArgs \(p. 381\)](#)
- [describeReturn \(p. 381\)](#)
- [describeTransform \(p. 381\)](#)
- [describeErrors \(p. 381\)](#)
- [describe \(p. 381\)](#)

[`__call__\(dfc, key, transformation_ctx = ""\)`](#)

Gets one `DynamicFrame` from a `DynamicFrameCollection`.

- `dfc` – The `DynamicFrameCollection` from which the `DynamicFrame` should be selected (required).
- `key` – The key of the `DynamicFrame` to select (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).

Returns the specified `DynamicFrame`.

[`apply\(cls, *args, **kwargs\)`](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[`name\(cls\)`](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[`describeArgs\(cls\)`](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[`describeReturn\(cls\)`](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[`describeTransform\(cls\)`](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[`describeErrors\(cls\)`](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[`describe\(cls\)`](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[`Spigot Class`](#)

Writes sample records to a specified destination during a transformation.

Methods

- [__call__ \(p. 382\)](#)

- [apply \(p. 382\)](#)
- [name \(p. 382\)](#)
- [describeArgs \(p. 382\)](#)
- [describeReturn \(p. 382\)](#)
- [describeTransform \(p. 382\)](#)
- [describeErrors \(p. 382\)](#)
- [describe \(p. 382\)](#)

`__call__(frame, path, options, transformation_ctx = "")`

Writes sample records to a specified destination during a transformation.

- `frame` – The `DynamicFrame` to spigot (required).
- `path` – The path to the destination to write to (required).
- `options` – JSON key-value pairs specifying options (optional). The "topk" option specifies that the first k records should be written. The "prob" option specifies the probability (as a decimal) of picking any given record, to be used in selecting records to write.
- `transformation_ctx` – A unique string that is used to identify state information (optional).

Returns the input `DynamicFrame` with an additional write step.

`apply(cls, *args, **kwargs)`

Inherited from `GlueTransform` [apply \(p. 360\)](#)

`name(cls)`

Inherited from `GlueTransform` [name \(p. 361\)](#)

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#)

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#)

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#)

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#)

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 362\)](#)

`SplitFields Class`

Splits a `DynamicFrame` into two new ones, by specified fields.

Methods

- [__call__ \(p. 383\)](#)
- [apply \(p. 383\)](#)
- [name \(p. 383\)](#)
- [describeArgs \(p. 383\)](#)
- [describeReturn \(p. 383\)](#)
- [describeTransform \(p. 383\)](#)
- [describeErrors \(p. 384\)](#)
- [describe \(p. 384\)](#)

[`__call__\(frame, paths, name1 = None, name2 = None, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0\)`](#)

Splits one or more fields in a `DynamicFrame` off into a new `DynamicFrame` and creates another new `DynamicFrame` containing the fields that remain.

- `frame` – The source `DynamicFrame` to split into two new ones (required).
- `paths` – A list of full paths to the fields to be split (required).
- `name1` – The name to assign to the `DynamicFrame` that will contain the fields to be split off (optional). If no name is supplied, the name of the source frame is used with "1" appended.
- `name2` – The name to assign to the `DynamicFrame` that will contain the fields that remain after the specified fields are split off (optional). If no name is provided, the name of the source frame is used with "2" appended.
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a `DynamicFrameCollection` containing two `DynamicFrames`: one contains only the specified fields to split off, and the other contains the remaining fields.

[`apply\(cls, *args, **kwargs\)`](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[`name\(cls\)`](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[`describeArgs\(cls\)`](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[`describeReturn\(cls\)`](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[`describeTransform\(cls\)`](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform](#) [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from [GlueTransform](#) [describe \(p. 362\)](#).

SplitRows Class

Splits a [DynamicFrame](#) in two by specified rows.

Methods

- [__call__ \(p. 384\)](#)
- [apply \(p. 384\)](#)
- [name \(p. 384\)](#)
- [describeArgs \(p. 385\)](#)
- [describeReturn \(p. 385\)](#)
- [describeTransform \(p. 385\)](#)
- [describeErrors \(p. 385\)](#)
- [describe \(p. 385\)](#)

`__call__\(frame, comparison_dict, name1="frame1", name2="frame2", transformation_ctx = "", info = None, stageThreshold = 0, totalThreshold = 0\)`

Splits one or more rows in a [DynamicFrame](#) off into a new [DynamicFrame](#).

- `frame` – The source [DynamicFrame](#) to split into two new ones (required).
- `comparison_dict` – A dictionary where the key is the full path to a column, and the value is another dictionary mapping comparators to the value to which the column values are compared. For example, `{"age": {">": 10, "<": 20}}` splits rows where the value of "age" is between 10 and 20, exclusive, from rows where "age" is outside that range (required).
- `name1` – The name to assign to the [DynamicFrame](#) that will contain the rows to be split off (optional).
- `name2` – The name to assign to the [DynamicFrame](#) that will contain the rows that remain after the specified rows are split off (optional).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns a [DynamicFrameCollection](#) that contains two [DynamicFrames](#): one contains only the specified rows to be split, and the other contains all remaining rows.

[apply\(cls, *args, **kwargs\)](#)

Inherited from [GlueTransform](#) [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from [GlueTransform](#) [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from [GlueTransform](#) [describeArgs](#) (p. 361).

[describeReturn\(cls\)](#)

Inherited from [GlueTransform](#) [describeReturn](#) (p. 361).

[describeTransform\(cls\)](#)

Inherited from [GlueTransform](#) [describeTransform](#) (p. 361).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform](#) [describeErrors](#) (p. 361).

[describe\(cls\)](#)

Inherited from [GlueTransform](#) [describe](#) (p. 362).

Unbox Class

Unboxes a string field in a [DynamicFrame](#).

Methods

- [__call__](#) (p. 385)
- [apply](#) (p. 386)
- [name](#) (p. 386)
- [describeArgs](#) (p. 386)
- [describeReturn](#) (p. 386)
- [describeTransform](#) (p. 386)
- [describeErrors](#) (p. 386)
- [describe](#) (p. 386)

[__call__\(frame, path, format, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0, **options\)](#)

Unboxes a string field in a [DynamicFrame](#).

- **frame** – The [DynamicFrame](#) in which to unbox a field. (required).
- **path** – The full path to the [StringNode](#) to unbox (required).
- **format** – A format specification (optional). This is used for an Amazon Simple Storage Service (Amazon S3) or an AWS Glue connection that supports multiple formats. See [Format Options for ETL Inputs and Outputs in AWS Glue](#) (p. 299) for the formats that are supported.
- **transformation_ctx** – A unique string that is used to identify state information (optional).
- **info** – A string associated with errors in the transformation (optional).
- **stageThreshold** – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- **totalThreshold** – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

- `separator` – A separator token (optional).
- `escaper` – An escape token (optional).
- `skipFirst` – `True` if the first line of data should be skipped, or `False` if it should not be skipped (optional).
- `withSchema` – A string containing schema for the data to be unboxed (optional). This should always be created using `StructType.json`.
- `withHeader` – `True` if the data being unpacked includes a header, or `False` if not (optional).

Returns a new `DynamicFrame` with unboxed `DynamicRecords`.

[apply\(cls, *args, **kwargs\)](#)

Inherited from `GlueTransform` [apply \(p. 360\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 361\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 362\)](#).

[UnnestFrame Class](#)

Unnests a `DynamicFrame`, flattens nested objects to top-level elements, and generates joinkeys for array objects.

Methods

- [__call__ \(p. 387\)](#)
- [apply \(p. 387\)](#)
- [name \(p. 387\)](#)
- [describeArgs \(p. 387\)](#)
- [describeReturn \(p. 387\)](#)
- [describeTransform \(p. 387\)](#)
- [describeErrors \(p. 387\)](#)
- [describe \(p. 387\)](#)

`__call__(frame, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0)`

Unnests a `DynamicFrame`. Flattens nested objects to top-level elements, and generates joinkeys for array objects.

- `frame` – The `DynamicFrame` to unnest (required).
- `transformation_ctx` – A unique string that is used to identify state information (optional).
- `info` – A string associated with errors in the transformation (optional).
- `stageThreshold` – The maximum number of errors that can occur in the transformation before it errors out (optional; the default is zero).
- `totalThreshold` – The maximum number of errors that can occur overall before processing errors out (optional; the default is zero).

Returns the unnested `DynamicFrame`.

`apply(cls, *args, **kwargs)`

Inherited from `GlueTransform` [apply \(p. 360\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 361\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 361\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 361\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 361\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 361\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 362\)](#).

Programming AWS Glue ETL Scripts in Scala

You can find Scala code examples and utilities for AWS Glue in the [AWS Glue samples repository](#) on the GitHub website.

AWS Glue supports an extension of the PySpark Scala dialect for scripting extract, transform, and load (ETL) jobs. The following sections describe how to use the AWS Glue Scala library and the AWS Glue API in ETL scripts, and provide reference documentation for the library.

Contents

- [Using Scala to Program AWS Glue ETL Scripts \(p. 392\)](#)
 - [Testing a Scala ETL Program in a Zeppelin Notebook on a Development Endpoint \(p. 392\)](#)
 - [Testing a Scala ETL Program in a Scala REPL \(p. 393\)](#)
- [APIs in the AWS Glue Scala Library \(p. 393\)](#)
 - [com.amazonaws.services.glue \(p. 393\)](#)
 - [com.amazonaws.services.glue.types \(p. 393\)](#)
 - [com.amazonaws.services.glue.util \(p. 394\)](#)
 - [AWS Glue Scala ChoiceOption APIs \(p. 394\)](#)
 - [ChoiceOption Trait \(p. 394\)](#)
 - [ChoiceOption Object \(p. 394\)](#)
 - [def apply \(p. 394\)](#)
 - [case class ChoiceOptionWithResolver \(p. 394\)](#)
 - [case class MatchCatalogSchemaChoiceOption \(p. 395\)](#)
 - [Abstract DataSink Class \(p. 395\)](#)
 - [def writeDynamicFrame \(p. 395\)](#)
 - [def pyWriteDynamicFrame \(p. 395\)](#)
 - [def setCatalogInfo \(p. 395\)](#)
 - [def supportsFormat \(p. 395\)](#)
 - [def setFormat \(p. 395\)](#)
 - [def withFormat \(p. 396\)](#)
 - [def setAccumulableSize \(p. 396\)](#)
 - [def getOutputErrorRecordsAccumulable \(p. 396\)](#)
 - [def errorsAsDynamicFrame \(p. 396\)](#)
 - [DataSink Object \(p. 396\)](#)
 - [def recordMetrics \(p. 396\)](#)
 - [AWS Glue Scala DataSource Trait \(p. 396\)](#)
 - [AWS Glue Scala DynamicFrame APIs \(p. 397\)](#)
 - [AWS Glue Scala DynamicFrame Class \(p. 398\)](#)
 - [val errorsCount \(p. 398\)](#)
 - [def applyMapping \(p. 399\)](#)
 - [def assertErrorThreshold \(p. 400\)](#)
 - [def count \(p. 400\)](#)
 - [def dropField \(p. 400\)](#)
 - [def dropFields \(p. 400\)](#)
 - [def dropNulls \(p. 400\)](#)
 - [def errorsAsDynamicFrame \(p. 401\)](#)
 - [def filter \(p. 401\)](#)
 - [def getName \(p. 401\)](#)
 - [def getNumPartitions \(p. 401\)](#)
 - [def getSchemaIfComputed \(p. 401\)](#)
 - [def isSchemaComputed \(p. 401\)](#)
 - [def javaToPython \(p. 402\)](#)
 - [def join \(p. 402\)](#)
 - [def map \(p. 403\)³⁸⁸](#)
 - [def mergeDynamicFrames \(p. 402\)](#)
 - [def printSchema \(p. 403\)](#)

- [def recomputeSchema \(p. 403\)](#)
- [def relationalize \(p. 403\)](#)
- [def renameField \(p. 404\)](#)
- [def repartition \(p. 405\)](#)
- [def resolveChoice \(p. 405\)](#)
- [def schema \(p. 406\)](#)
- [def selectField \(p. 406\)](#)
- [def selectFields \(p. 406\)](#)
- [def show \(p. 407\)](#)
- [def spigot \(p. 407\)](#)
- [def splitFields \(p. 407\)](#)
- [def splitRows \(p. 408\)](#)
- [def stageErrorsCount \(p. 408\)](#)
- [def toDF \(p. 408\)](#)
- [def unbox \(p. 409\)](#)
- [def unnest \(p. 409\)](#)
- [def withFrameSchema \(p. 410\)](#)
- [def withName \(p. 410\)](#)
- [def withTransformationContext \(p. 410\)](#)
- [The DynamicFrame Object \(p. 411\)](#)
 - [def apply \(p. 411\)](#)
 - [def emptyDynamicFrame \(p. 411\)](#)
 - [def fromPythonRDD \(p. 411\)](#)
 - [def ignoreErrors \(p. 411\)](#)
 - [def inlineErrors \(p. 411\)](#)
 - [def newFrameWithErrors \(p. 411\)](#)
- [AWS Glue Scala DynamicRecord Class \(p. 411\)](#)
 - [def addField \(p. 412\)](#)
 - [def dropField \(p. 412\)](#)
 - [def setError \(p. 412\)](#)
 - [def isError \(p. 413\)](#)
 - [def getError \(p. 413\)](#)
 - [def clearError \(p. 413\)](#)
 - [def write \(p. 413\)](#)
 - [def readFields \(p. 413\)](#)
 - [def clone \(p. 413\)](#)
 - [def schema \(p. 413\)](#)
 - [def getRoot \(p. 413\)](#)
 - [def toJson \(p. 413\)](#)
 - [def getFieldNode \(p. 414\)](#)
 - [def getField \(p. 414\)](#)
 - [def hashCode \(p. 414\)](#)
 - [def equals \(p. 414\) 389](#)
- [DynamicRecord Object \(p. 414\)](#)
 - [def apply \(p. 414\)](#)

- [RecordTraverser Trait \(p. 414\)](#)
 - [AWS Glue Scala GlueContext APIs \(p. 415\)](#)
 - [def getCatalogSink \(p. 415\)](#)
 - [def getCatalogSource \(p. 416\)](#)
 - [def getJDBCSink \(p. 416\)](#)
 - [def getSink \(p. 417\)](#)
 - [def getSinkWithFormat \(p. 417\)](#)
 - [def getSource \(p. 418\)](#)
 - [def getSourceWithFormat \(p. 419\)](#)
 - [def getSparkSession \(p. 419\)](#)
 - [def purgeTable \(p. 419\)](#)
 - [def purgeS3Path \(p. 420\)](#)
 - [def transitionTable \(p. 421\)](#)
 - [def transitionS3Path \(p. 421\)](#)
 - [def this \(p. 422\)](#)
 - [def this \(p. 422\)](#)
 - [def this \(p. 423\)](#)
 - [MappingSpec \(p. 423\)](#)
 - [MappingSpec Case Class \(p. 423\)](#)
 - [MappingSpec Object \(p. 423\)](#)
 - [val orderingByTarget \(p. 423\)](#)
 - [def apply \(p. 423\)](#)
 - [def apply \(p. 424\)](#)
 - [def apply \(p. 424\)](#)
 - [AWS Glue Scala ResolveSpec APIs \(p. 424\)](#)
 - [ResolveSpec Object \(p. 424\)](#)
 - [def \(p. 425\)](#)
 - [def \(p. 425\)](#)
 - [ResolveSpec Case Class \(p. 425\)](#)
 - [ResolveSpec def Methods \(p. 425\)](#)
 - [AWS Glue Scala ArrayNode APIs \(p. 425\)](#)
 - [ArrayNode Case Class \(p. 425\)](#)
 - [ArrayNode def Methods \(p. 426\)](#)
 - [AWS Glue Scala BinaryNode APIs \(p. 426\)](#)
 - [BinaryNode Case Class \(p. 426\)](#)
 - [BinaryNode val Fields \(p. 426\)](#)
 - [BinaryNode def Methods \(p. 427\)](#)
 - [AWS Glue Scala BooleanNode APIs \(p. 427\)](#)
 - [BooleanNode Case Class \(p. 427\)](#)
 - [BooleanNode val Fields \(p. 427\)](#)
 - [BooleanNode def Methods \(p. 427\)](#)
 - [AWS Glue Scala ByteNode APIs \(p. 427\)](#)
 - [ByteNode Case Class \(p. 427\)](#)
 - [ByteNode val Fields \(p. 427\)](#)
 - [ByteNode def Methods \(p. 427\)](#)
-

- [AWS Glue Scala DateNode APIs \(p. 427\)](#)
 - [DateNode Case Class \(p. 428\)](#)
 - [DateNode val Fields \(p. 428\)](#)
 - [DateNode def Methods \(p. 428\)](#)
 - [AWS Glue Scala DecimalNode APIs \(p. 428\)](#)
 - [DecimalNode Case Class \(p. 428\)](#)
 - [DecimalNode val Fields \(p. 428\)](#)
 - [DecimalNode def Methods \(p. 428\)](#)
 - [AWS Glue Scala DoubleNode APIs \(p. 428\)](#)
 - [DoubleNode Case Class \(p. 428\)](#)
 - [DoubleNode val Fields \(p. 428\)](#)
 - [DoubleNode def Methods \(p. 429\)](#)
 - [AWS Glue Scala DynamicNode APIs \(p. 429\)](#)
 - [DynamicNode Class \(p. 429\)](#)
 - [DynamicNode def Methods \(p. 429\)](#)
 - [DynamicNode Object \(p. 429\)](#)
 - [DynamicNode def Methods \(p. 429\)](#)
 - [AWS Glue Scala FloatNode APIs \(p. 430\)](#)
 - [FloatNode Case Class \(p. 430\)](#)
 - [FloatNode val Fields \(p. 430\)](#)
 - [FloatNode def Methods \(p. 430\)](#)
 - [AWS Glue Scala IntegerNode APIs \(p. 430\)](#)
 - [IntegerNode Case Class \(p. 430\)](#)
 - [IntegerNode val Fields \(p. 430\)](#)
 - [IntegerNode def Methods \(p. 430\)](#)
 - [AWS Glue Scala LongNode APIs \(p. 430\)](#)
 - [LongNode Case Class \(p. 430\)](#)
 - [LongNode val Fields \(p. 431\)](#)
 - [LongNode def Methods \(p. 431\)](#)
 - [AWS Glue Scala MapLikeNode APIs \(p. 431\)](#)
 - [MapLikeNode Class \(p. 431\)](#)
 - [MapLikeNode def Methods \(p. 431\)](#)
 - [AWS Glue Scala MapNode APIs \(p. 432\)](#)
 - [MapNode Case Class \(p. 432\)](#)
 - [MapNode def Methods \(p. 432\)](#)
 - [AWS Glue Scala NullNode APIs \(p. 432\)](#)
 - [NullNode Class \(p. 432\)](#)
 - [NullNode Case Object \(p. 432\)](#)
 - [AWS Glue Scala ObjectNode APIs \(p. 432\)](#)
 - [ObjectNode Object \(p. 433\)](#)
 - [ObjectNode def Methods \(p. 433\)](#)
 - [ObjectNode Case Class \(p. 433\)](#)
 - [ObjectNode def Methods \(p. 433\)](#)
 - [AWS Glue Scala ScalarNode APIs \(p. 433\)](#)
 - [ScalarNode Class \(p. 433\)](#)
-

- [ScalarNode def Methods \(p. 434\)](#)
- [ScalarNode Object \(p. 434\)](#)
 - [ScalarNode def Methods \(p. 434\)](#)
- [AWS Glue Scala ShortNode APIs \(p. 434\)](#)
 - [ShortNode Case Class \(p. 434\)](#)
 - [ShortNode val Fields \(p. 435\)](#)
 - [ShortNode def Methods \(p. 435\)](#)
- [AWS Glue Scala StringNode APIs \(p. 435\)](#)
 - [StringNode Case Class \(p. 435\)](#)
 - [StringNode val Fields \(p. 435\)](#)
 - [StringNode def Methods \(p. 435\)](#)
- [AWS Glue Scala TimestampNode APIs \(p. 435\)](#)
 - [TimestampNode Case Class \(p. 435\)](#)
 - [TimestampNode val Fields \(p. 435\)](#)
 - [TimestampNode def Methods \(p. 435\)](#)
- [AWS Glue Scala GlueArgParser APIs \(p. 436\)](#)
 - [GlueArgParser Object \(p. 436\)](#)
 - [GlueArgParser def Methods \(p. 436\)](#)
- [AWS Glue Scala Job APIs \(p. 436\)](#)
 - [Job Object \(p. 436\)](#)
 - [Job def Methods \(p. 436\)](#)

Using Scala to Program AWS Glue ETL Scripts

You can automatically generate a Scala extract, transform, and load (ETL) program using the AWS Glue console, and modify it as needed before assigning it to a job. Or, you can write your own program from scratch. For more information, see [Adding Jobs in AWS Glue \(p. 164\)](#). AWS Glue then compiles your Scala program on the server before running the associated job.

To ensure that your program compiles without errors and runs as expected, it's important that you load it on a development endpoint in a REPL (Read-Eval-Print Loop) or an Apache Zeppelin Notebook and test it there before running it in a job. Because the compile process occurs on the server, you will not have good visibility into any problems that happen there.

Testing a Scala ETL Program in a Zeppelin Notebook on a Development Endpoint

To test a Scala program on an AWS Glue development endpoint, set up the development endpoint as described in [Managing Notebooks \(p. 184\)](#).

Next, connect it to an Apache Zeppelin Notebook that is either running locally on your machine or remotely on an Amazon EC2 notebook server. To install a local version of a Zeppelin Notebook, follow the instructions in [Tutorial: Local Zeppelin Notebook \(p. 195\)](#).

The only difference between running Scala code and running PySpark code on your Notebook is that you should start each paragraph on the Notebook with the the following:

```
%spark
```

This prevents the Notebook server from defaulting to the PySpark flavor of the Spark interpreter.

Testing a Scala ETL Program in a Scala REPL

You can test a Scala program on a development endpoint using the AWS Glue Scala REPL. Follow the instructions in [Tutorial: Use a SageMaker Notebook \(p. 202\)](#), except at the end of the SSH-to-REPL command, replace `-t gluepyspark` with `-t glue-spark-shell`. This invokes the AWS Glue Scala REPL.

To close the REPL when you are finished, type `sys.exit`.

APIs in the AWS Glue Library

AWS Glue supports an extension of the PySpark Scala dialect for scripting extract, transform, and load (ETL) jobs. The following sections describe the APIs in the AWS Glue Scala library.

[com.amazonaws.services.glue](#)

The **com.amazonaws.services.glue** package in the AWS Glue Scala library contains the following APIs:

- [ChoiceOption \(p. 394\)](#)
- [DataSink \(p. 395\)](#)
- [DataSource trait \(p. 396\)](#)
- [DynamicFrame \(p. 397\)](#)
- [DynamicRecord \(p. 411\)](#)
- [GlueContext \(p. 415\)](#)
- [MappingSpec \(p. 423\)](#)
- [ResolveSpec \(p. 424\)](#)

[com.amazonaws.services.glue.types](#)

The **com.amazonaws.services.glue.types** package in the AWS Glue Scala library contains the following APIs:

- [ArrayNode \(p. 425\)](#)
- [BinaryNode \(p. 426\)](#)
- [BooleanNode \(p. 427\)](#)
- [ByteNode \(p. 427\)](#)
- [DateNode \(p. 427\)](#)
- [DecimalNode \(p. 428\)](#)
- [DoubleNode \(p. 428\)](#)
- [DynamicNode \(p. 429\)](#)
- [FloatNode \(p. 430\)](#)
- [IntegerNode \(p. 430\)](#)
- [LongNode \(p. 430\)](#)
- [MapLikeNode \(p. 431\)](#)
- [MapNode \(p. 432\)](#)
- [NullNode \(p. 432\)](#)
- [ObjectNode \(p. 432\)](#)
- [ScalarNode \(p. 433\)](#)

- [ShortNode \(p. 434\)](#)
- [StringNode \(p. 435\)](#)
- [TimestampNode \(p. 435\)](#)

com.amazonaws.services.glue.util

The **com.amazonaws.services.glue.util** package in the AWS Glue Scala library contains the following APIs:

- [GlueArgParser \(p. 436\)](#)
- [Job \(p. 436\)](#)

AWS Glue Scala ChoiceOption APIs

Topics

- [ChoiceOption Trait \(p. 394\)](#)
- [ChoiceOption Object \(p. 394\)](#)
- [case class ChoiceOptionWithResolver \(p. 394\)](#)
- [case class MatchCatalogSchemaChoiceOption \(p. 395\)](#)

Package: com.amazonaws.services.glue

ChoiceOption Trait

```
trait ChoiceOption extends Serializable
```

ChoiceOption Object

ChoiceOption

```
object ChoiceOption
```

A general strategy to resolve choice applicable to all `ChoiceType` nodes in a `DynamicFrame`.

- `val CAST`
- `val MAKE_COLS`
- `val MAKE_STRUCT`
- `val MATCH_CATALOG`
- `val PROJECT`

def apply

```
def apply(choice: String): ChoiceOption
```

case class ChoiceOptionWithResolver

```
case class ChoiceOptionWithResolver(name: String, choiceResolver: ChoiceResolver) extends ChoiceOption {}
```

case class MatchCatalogSchemaChoiceOption

```
case class MatchCatalogSchemaChoiceOption() extends ChoiceOption {}
```

Abstract DataSink Class

Topics

- [def writeDynamicFrame \(p. 395\)](#)
- [def pyWriteDynamicFrame \(p. 395\)](#)
- [def setCatalogInfo \(p. 395\)](#)
- [def supportsFormat \(p. 395\)](#)
- [def setFormat \(p. 395\)](#)
- [def withFormat \(p. 396\)](#)
- [def setAccumulableSize \(p. 396\)](#)
- [def getOutputErrorRecordsAccumulable \(p. 396\)](#)
- [def errorsAsDynamicFrame \(p. 396\)](#)
- [DataSink Object \(p. 396\)](#)

Package: com.amazonaws.services.glue

```
abstract class DataSink
```

The writer analog to a DataSource. DataSink encapsulates a destination and a format that a DynamicFrame can be written to.

def writeDynamicFrame

```
def writeDynamicFrame( frame : DynamicFrame,  
                      callSite : CallSite = CallSite("Not provided", "")  
                    ) : DynamicFrame
```

def pyWriteDynamicFrame

```
def pyWriteDynamicFrame( frame : DynamicFrame,  
                        site : String = "Not provided",  
                        info : String = "" )
```

def setCatalogInfo

```
def setCatalogInfo(catalogDatabase: String,  
                   catalogTableName : String,  
                   catalogId : String = "")
```

def supportsFormat

```
def supportsFormat( format : String ) : Boolean
```

def setFormat

```
def setFormat( format : String,
```

```
    options : JsonOptions
) : Unit
```

def withFormat

```
def withFormat( format : String,
                options : JsonOptions = JsonOptions.empty
              ) : DataSink
```

def setAccumulableSize

```
def setAccumulableSize( size : Int ) : Unit
```

def getOutputErrorRecordsAccumulable

```
def getOutputErrorRecordsAccumulable : Accumulable[List[OutputError], OutputError]
```

def errorsAsDynamicFrame

```
def errorsAsDynamicFrame : DynamicFrame
```

DataSink Object

```
object DataSink
```

def recordMetrics

```
def recordMetrics( frame : DynamicFrame,
                   ctxt : String
                 ) : DynamicFrame
```

AWS Glue Scala DataSource Trait

Package: com.amazonaws.services.glue

A high-level interface for producing a DynamicFrame.

```
trait DataSource {

  def getDynamicFrame : DynamicFrame

  def getDynamicFrame( minPartitions : Int,
                      targetPartitions : Int
                    ) : DynamicFrame

  def glueContext : GlueContext

  def setFormat( format : String,
                options : String
              ) : Unit

  def setFormat( format : String,
                options : JsonOptions
```

```
) : Unit

def supportsFormat( format : String ) : Boolean

def withFormat( format : String,
                 options : JsonOptions = JsonOptions.empty
                 ) : DataSource
}
```

AWS Glue Scala DynamicFrame APIs

Package: com.amazonaws.services.glue

Contents

- [AWS Glue Scala DynamicFrame Class \(p. 398\)](#)
 - [val errorsCount \(p. 398\)](#)
 - [def applyMapping \(p. 399\)](#)
 - [def assertErrorThreshold \(p. 400\)](#)
 - [def count \(p. 400\)](#)
 - [def dropField \(p. 400\)](#)
 - [def dropFields \(p. 400\)](#)
 - [def dropNulls \(p. 400\)](#)
 - [def errorsAsDynamicFrame \(p. 401\)](#)
 - [def filter \(p. 401\)](#)
 - [def getName \(p. 401\)](#)
 - [def getNumPartitions \(p. 401\)](#)
 - [def getSchemaComputed \(p. 401\)](#)
 - [def isSchemaComputed \(p. 401\)](#)
 - [def javaToPython \(p. 402\)](#)
 - [def join \(p. 402\)](#)
 - [def map \(p. 402\)](#)
 - [def mergeDynamicFrames \(p. 402\)](#)
 - [def printSchema \(p. 403\)](#)
 - [def recomputeSchema \(p. 403\)](#)
 - [def relationalize \(p. 403\)](#)
 - [def renameField \(p. 404\)](#)
 - [def repartition \(p. 405\)](#)
 - [def resolveChoice \(p. 405\)](#)
 - [def schema \(p. 406\)](#)
 - [def selectField \(p. 406\)](#)
 - [def selectFields \(p. 406\)](#)
 - [def show \(p. 407\)](#)
 - [def spigot \(p. 407\)](#)
 - [def splitFields \(p. 407\)](#)
 - [def splitRows \(p. 408\)](#)
 - [def stageErrorsCount \(p. 408\)](#)
 - [def toDF \(p. 408\)](#)
 - [def unbox \(p. 409\)](#)
 - [def unnest \(p. 409\)](#)

- [def withFrameSchema \(p. 410\)](#)
- [def withName \(p. 410\)](#)
- [def withTransformationContext \(p. 410\)](#)
- [The DynamicFrame Object \(p. 411\)](#)
 - [def apply \(p. 411\)](#)
 - [def emptyDynamicFrame \(p. 411\)](#)
 - [def fromPythonRDD \(p. 411\)](#)
 - [def ignoreErrors \(p. 411\)](#)
 - [def inlineErrors \(p. 411\)](#)
 - [def newFrameWithErrors \(p. 411\)](#)

AWS Glue Scala DynamicFrame Class

Package: `com.amazonaws.services.glue`

```
class DynamicFrame extends Serializable with Logging {
    val glueContext : GlueContext,
    _records : RDD[DynamicRecord],
    val name : String = s"",
    val transformationContext : String = DynamicFrame.UNDEFINED,
    callSite : CallSite = CallSite("Not provided", ""),
    stageThreshold : Long = 0,
    totalThreshold : Long = 0,
    prevErrors : => Long = 0,
    errorExpr : => Unit = {} }
```

A `DynamicFrame` is a distributed collection of self-describing `DynamicRecord` (p. 411) objects.

`DynamicFrames` are designed to provide a flexible data model for ETL (extract, transform, and load) operations. They don't require a schema to create, and you can use them to read and transform data that contains messy or inconsistent values and types. A schema can be computed on demand for those operations that need one.

`DynamicFrames` provide a range of transformations for data cleaning and ETL. They also support conversion to and from SparkSQL `DataFrames` to integrate with existing code and the many analytics operations that `DataFrames` provide.

The following parameters are shared across many of the AWS Glue transformations that construct `DynamicFrames`:

- `transformationContext` — The identifier for this `DynamicFrame`. The `transformationContext` is used as a key for job bookmark state that is persisted across runs.
- `callSite` — Provides context information for error reporting. These values are automatically set when calling from Python.
- `stageThreshold` — The maximum number of error records that are allowed from the computation of this `DynamicFrame` before throwing an exception, excluding records that are present in the previous `DynamicFrame`.
- `totalThreshold` — The maximum number of total error records before an exception is thrown, including those from previous frames.

`val errorsCount`

```
val errorsCount
```

The number of error records in this `DynamicFrame`. This includes errors from previous operations.

def applyMapping

```
def applyMapping( mappings : Seq[Product4[String, String, String, String]],  
                 caseSensitive : Boolean = true,  
                 transformationContext : String = "",  
                 callSite : CallSite = CallSite("Not provided", ""),  
                 stageThreshold : Long = 0,  
                 totalThreshold : Long = 0  
               ) : DynamicFrame
```

- `mappings` — A sequence of mappings to construct a new `DynamicFrame`.
- `caseSensitive` — Whether to treat source columns as case sensitive. Setting this to false might help when integrating with case-insensitive stores like the AWS Glue Data Catalog.

Selects, projects, and casts columns based on a sequence of mappings.

Each mapping is made up of a source column and type and a target column and type. Mappings can be specified as either a four-tuple (`source_path`, `source_type`, `target_path`, `target_type`) or a [MappingSpec \(p. 423\)](#) object containing the same information.

In addition to using mappings for simple projections and casting, you can use them to nest or unnest fields by separating components of the path with '.' (period).

For example, suppose that you have a `DynamicFrame` with the following schema.

```
{{{  
  root  
  |-- name: string  
  |-- age: int  
  |-- address: struct  
  |   |-- state: string  
  |   |-- zip: int  
}}}
```

You can make the following call to unnest the `state` and `zip` fields.

```
{{{  
  df.applyMapping(  
    Seq(("name", "string", "name", "string"),  
         ("age", "int", "age", "int"),  
         ("address.state", "string", "state", "string"),  
         ("address.zip", "int", "zip", "int"))  
  )  
}}}
```

The resulting schema is as follows.

```
{{{  
  root  
  |-- name: string  
  |-- age: int  
  |-- state: string  
  |-- zip: int  
}}}
```

You can also use `applyMapping` to re-nest columns. For example, the following inverts the previous transformation and creates a struct named `address` in the target.

```
{{{  
    df.applyMapping(  
        Seq(("name", "string", "name", "string"),  
            ("age", "int", "age", "int"),  
            ("state", "string", "address.state", "string"),  
            ("zip", "int", "address.zip", "int")))  
}}}
```

Field names that contain '.' (period) characters can be quoted by using backticks (` `).

Note

Currently, you can't use the `applyMapping` method to map columns that are nested under arrays.

[def assertErrorThreshold](#)

```
def assertErrorThreshold : Unit
```

An action that forces computation and verifies that the number of error records falls below `stageThreshold` and `totalThreshold`. Throws an exception if either condition fails.

[def count](#)

```
lazy  
def count
```

Returns the number of elements in this `DynamicFrame`.

[def dropField](#)

```
def dropField( path : String,  
              transformationContext : String = "",  
              callSite : CallSite = CallSite("Not provided", ""),  
              stageThreshold : Long = 0,  
              totalThreshold : Long = 0  
            ) : DynamicFrame
```

Returns a new `DynamicFrame` with the specified column removed.

[def dropFields](#)

```
def dropFields( fieldNames : Seq[String], // The column names to drop.  
               transformationContext : String = "",  
               callSite : CallSite = CallSite("Not provided", ""),  
               stageThreshold : Long = 0,  
               totalThreshold : Long = 0  
             ) : DynamicFrame
```

Returns a new `DynamicFrame` with the specified columns removed.

You can use this method to delete nested columns, including those inside of arrays, but not to drop specific array elements.

[def dropNulls](#)

```
def dropNulls( transformationContext : String = "",  
              callSite : CallSite = CallSite("Not provided", ""),
```

```
stageThreshold : Long = 0,  
totalThreshold : Long = 0 )
```

Returns a new `DynamicFrame` with all null columns removed.

Note

This only removes columns of type `NullType`. Individual null values in other columns are not removed or modified.

`def errorsAsDynamicFrame`

```
def errorsAsDynamicFrame
```

Returns a new `DynamicFrame` containing the error records from this `DynamicFrame`.

`def filter`

```
def filter( f : DynamicRecord => Boolean,  
           errorMsg : String = "",  
           transformationContext : String = "",  
           callSite : CallSite = CallSite("Not provided"),  
           stageThreshold : Long = 0,  
           totalThreshold : Long = 0  
         ) : DynamicFrame
```

Constructs a new `DynamicFrame` containing only those records for which the function 'f' returns `true`. The filter function 'f' should not mutate the input record.

`def getName`

```
def getName : String
```

Returns the name of this `DynamicFrame`.

`def getNumPartitions`

```
def getNumPartitions
```

Returns the number of partitions in this `DynamicFrame`.

`def getSchemaIfComputed`

```
def getSchemaIfComputed : Option[Schema]
```

Returns the schema if it has already been computed. Does not scan the data if the schema has not already been computed.

`def isSchemaComputed`

```
def isSchemaComputed : Boolean
```

Returns `true` if the schema has been computed for this `DynamicFrame`, or `false` if not. If this method returns `false`, then calling the `schema` method requires another pass over the records in this `DynamicFrame`.

def javaToPython

```
def javaToPython : JavaRDD[Array[Byte]]
```

def join

```
def join( keys1 : Seq[String],
          keys2 : Seq[String],
          frame2 : DynamicFrame,
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided", ""),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

- **keys1** — The columns in this `DynamicFrame` to use for the join.
- **keys2** — The columns in `frame2` to use for the join. Must be the same length as `keys1`.
- **frame2** — The `DynamicFrame` to join against.

Returns the result of performing an equijoin with `frame2` using the specified keys.

def map

```
def map( f : DynamicRecord => DynamicRecord,
          errorMsg : String = "",
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided", ""),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

Returns a new `DynamicFrame` constructed by applying the specified function '`f`' to each record in this `DynamicFrame`.

This method copies each record before applying the specified function, so it is safe to mutate the records. If the mapping function throws an exception on a given record, that record is marked as an error, and the stack trace is saved as a column in the error record.

def mergeDynamicFrames

```
def mergeDynamicFrames( stageDynamicFrame: DynamicFrame, primaryKeys: Seq[String],
                        transformationContext: String = "",
                        options: JsonOptions = JsonOptions.empty, callSite: CallSite =
                        CallSite("Not provided"),
                        stageThreshold: Long = 0, totalThreshold: Long = 0): DynamicFrame
```

- **stageDynamicFrame** — The staging `DynamicFrame` to merge.
- **primaryKeys** — The list of primary key fields to match records from the source and staging `DynamicFrames`.
- **transformationContext** — A unique string that is used to retrieve metadata about the current transformation (optional).
- **options** — A string of JSON name-value pairs that provide additional information for this transformation.
- **callSite** — Used to provide context information for error reporting.

- `stageThreshold` — A `Long`. The number of errors in the given transformation for which the processing needs to error out.
- `totalThreshold` — A `Long`. The total number of errors up to and including in this transformation for which the processing needs to error out.

Merges this `DynamicFrame` with a staging `DynamicFrame` based on the specified primary keys to identify records. Duplicate records (records with the same primary keys) are not de-duplicated. If there is no matching record in the staging frame, all records (including duplicates) are retained from the source. If the staging frame has matching records, the records from the staging frame overwrite the records in the source in AWS Glue.

The returned `DynamicFrame` contains record A in the following cases:

1. If A exists in both the source frame and the staging frame, then A in the staging frame is returned.
2. If A is in the source table and A.`primaryKeys` is not in the `stagingDynamicFrame` (that means A is not updated in the staging table).

The source frame and staging frame do not need to have the same schema.

Example

```
val mergedFrame: DynamicFrame = srcFrame.mergeDynamicFrames(stageFrame, Seq("id1", "id2"))
```

`def printSchema`

```
def printSchema : Unit
```

Prints the schema of this `DynamicFrame` to `stdout` in a human-readable format.

`def recomputeSchema`

```
def recomputeSchema : Schema
```

Forces a schema recomputation. This requires a scan over the data, but it might "tighten" the schema if there are some fields in the current schema that are not present in the data.

Returns the recomputed schema.

`def relationalize`

```
def relationalize( rootTableName : String,
                  stagingPath : String,
                  options : JsonOptions = JsonOptions.empty,
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided"),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : Seq[DynamicFrame]
```

- `rootTableName` — The name to use for the base `DynamicFrame` in the output. `DynamicFrames` that are created by pivoting arrays start with this as a prefix.
- `stagingPath` — The Amazon Simple Storage Service (Amazon S3) path for writing intermediate data.
- `options` — Relationalize options and configuration. Currently unused.

Flattens all nested structures and pivots arrays into separate tables.

You can use this operation to prepare deeply nested data for ingestion into a relational database. Nested structs are flattened in the same manner as the [unnest \(p. 409\)](#) transform. Additionally, arrays are pivoted into separate tables with each array element becoming a row. For example, suppose that you have a `DynamicFrame` with the following data.

```
{"name": "Nancy", "age": 47, "friends": ["Fred", "Lakshmi"]}  
{"name": "Stephanie", "age": 28, "friends": ["Yao", "Phil", "Alvin"]}  
{"name": "Nathan", "age": 54, "friends": ["Nicolai", "Karen"]}
```

Execute the following code.

```
{{{  
    df.relationelize("people", "s3:/my_bucket/my_path", JsonOptions.empty)  
}}}
```

This produces two tables. The first table is named "people" and contains the following.

```
{{{  
    {"name": "Nancy", "age": 47, "friends": 1}  
    {"name": "Stephanie", "age": 28, "friends": 2}  
    {"name": "Nathan", "age": 54, "friends": 3}  
}}}
```

Here, the friends array has been replaced with an auto-generated join key. A separate table named `people.friends` is created with the following content.

```
{{{  
    {"id": 1, "index": 0, "val": "Fred"}  
    {"id": 1, "index": 1, "val": "Lakshmi"}  
    {"id": 2, "index": 0, "val": "Yao"}  
    {"id": 2, "index": 1, "val": "Phil"}  
    {"id": 2, "index": 2, "val": "Alvin"}  
    {"id": 3, "index": 0, "val": "Nicolai"}  
    {"id": 3, "index": 1, "val": "Karen"}  
}}}
```

In this table, 'id' is a join key that identifies which record the array element came from, 'index' refers to the position in the original array, and 'val' is the actual array entry.

The `relationelize` method returns the sequence of `DynamicFrames` created by applying this process recursively to all arrays.

Note

The AWS Glue library automatically generates join keys for new tables. To ensure that join keys are unique across job runs, you must enable job bookmarks.

[def renameField](#)

```
def renameField( oldName : String,  
                newName : String,  
                transformationContext : String = "",  
                callSite : CallSite = CallSite("Not provided", ""),  
                stageThreshold : Long = 0,  
                totalThreshold : Long = 0  
              ) : DynamicFrame
```

- `oldName` — The original name of the column.
- `newName` — The new name of the column.

Returns a new `DynamicFrame` with the specified field renamed.

You can use this method to rename nested fields. For example, the following code would rename `state` to `state_code` inside the `address` struct.

```
{{{  
    df.renameField("address.state", "address.state_code")  
}}}
```

[def repartition](#)

```
def repartition( numPartitions : Int,  
                 transformationContext : String = "",  
                 callSite : CallSite = CallSite("Not provided", ""),  
                 stageThreshold : Long = 0,  
                 totalThreshold : Long = 0  
               ) : DynamicFrame
```

Returns a new `DynamicFrame` with `numPartitions` partitions.

[def resolveChoice](#)

```
def resolveChoice( specs : Seq[Product2[String, String]] = Seq.empty[ResolveSpec],  
                  choiceOption : Option[ChoiceOption] = None,  
                  database : Option[String] = None,  
                  tableName : Option[String] = None,  
                  transformationContext : String = "",  
                  callSite : CallSite = CallSite("Not provided", ""),  
                  stageThreshold : Long = 0,  
                  totalThreshold : Long = 0  
                ) : DynamicFrame
```

- `choiceOption` — An action to apply to all `ChoiceType` columns not listed in the `specs` sequence.
- `database` — The Data Catalog database to use with the `match_catalog` action.
- `tableName` — The Data Catalog table to use with the `match_catalog` action.

Returns a new `DynamicFrame` by replacing one or more `ChoiceTypes` with a more specific type.

There are two ways to use `resolveChoice`. The first is to specify a sequence of specific columns and how to resolve them. These are specified as tuples made up of (`column, action`) pairs.

The following are the possible actions:

- `cast:type` — Attempts to cast all values to the specified type.
- `make_cols` — Converts each distinct type to a column with the name `columnName_type`.
- `make_struct` — Converts a column to a struct with keys for each distinct type.
- `project:type` — Retains only values of the specified type.

The other mode for `resolveChoice` is to specify a single resolution for all `ChoiceTypes`. You can use this in cases where the complete list of `ChoiceTypes` is unknown before execution. In addition to the actions listed preceding, this mode also supports the following action:

- `match_catalog` — Attempts to cast each `ChoiceType` to the corresponding type in the specified catalog table.

Examples:

Resolve the `user.id` column by casting to an int, and make the `address` field retain only structs.

```
{{{  
    df.resolveChoice(specs = Seq(("user.id", "cast:int"), ("address", "project:struct")))  
}}}
```

Resolve all `ChoiceTypes` by converting each choice to a separate column.

```
{{{  
    df.resolveChoice(choiceOption = Some(ChoiceOption("make_cols")))  
}}}
```

Resolve all `ChoiceTypes` by casting to the types in the specified catalog table.

```
{{{  
    df.resolveChoice(choiceOption = Some(ChoiceOption("match_catalog")),  
                      database = Some("my_database"),  
                      tableName = Some("my_table"))  
}}}
```

def schema

```
def schema : Schema
```

Returns the schema of this `DynamicFrame`.

The returned schema is guaranteed to contain every field that is present in a record in this `DynamicFrame`. But in a small number of cases, it might also contain additional fields. You can use the [unnest \(p. 409\)](#) method to "tighten" the schema based on the records in this `DynamicFrame`.

def selectField

```
def selectField( fieldName : String,  
                transformationContext : String = "",  
                callSite : CallSite = CallSite("Not provided", ""),  
                stageThreshold : Long = 0,  
                totalThreshold : Long = 0  
) : DynamicFrame
```

Returns a single field as a `DynamicFrame`.

def selectFields

```
def selectFields( paths : Seq[String],  
                 transformationContext : String = "",  
                 callSite : CallSite = CallSite("Not provided", ""),  
                 stageThreshold : Long = 0,  
                 totalThreshold : Long = 0  
) : DynamicFrame
```

- `paths` — The sequence of column names to select.

Returns a new `DynamicFrame` containing the specified columns.

Note

You can only use the `selectFields` method to select top-level columns. You can use the [applyMapping \(p. 399\)](#) method to select nested columns.

`def show`

```
def show( numRows : Int = 20 ) : Unit
```

- `numRows` — The number of rows to print.

Prints rows from this `DynamicFrame` in JSON format.

`def spigot`

```
def spigot( path : String,
            options : JsonOptions = new JsonOptions("{}"),
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

Passthrough transformation that returns the same records but writes out a subset of records as a side effect.

- `path` — The path in Amazon S3 to write output to, in the form `s3://bucket//path`.
- `options` — An optional `JsonOptions` map describing the sampling behavior.

Returns a `DynamicFrame` that contains the same records as this one.

By default, writes 100 arbitrary records to the location specified by `path`. You can customize this behavior by using the `options` map. Valid keys include the following:

- `topk` — Specifies the total number of records written out. The default is 100.
- `prob` — Specifies the probability (as a decimal) that an individual record is included. Default is 1.

For example, the following call would sample the dataset by selecting each record with a 20 percent probability and stopping after 200 records have been written.

```
{{{  
    df.spigot("s3://my_bucket/my_path", JsonOptions(Map("topk" -> 200, "prob" -> 0.2)))  
}}}
```

`def splitFields`

```
def splitFields( paths : Seq[String],
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : Seq[DynamicFrame]
```

- `paths` — The paths to include in the first `DynamicFrame`.

Returns a sequence of two DynamicFrames. The first DynamicFrame contains the specified paths, and the second contains all other columns.

[def splitRows](#)

```
def splitRows( paths : Seq[String],  
              values : Seq[Any],  
              operators : Seq[String],  
              transformationContext : String,  
              callSite : CallSite,  
              stageThreshold : Long,  
              totalThreshold : Long  
            ) : Seq[DynamicFrame]
```

Splits rows based on predicates that compare columns to constants.

- **paths** — The columns to use for comparison.
- **values** — The constant values to use for comparison.
- **operators** — The operators to use for comparison.

Returns a sequence of two DynamicFrames. The first contains rows for which the predicate is true and the second contains those for which it is false.

Predicates are specified using three sequences: 'paths' contains the (possibly nested) column names, 'values' contains the constant values to compare to, and 'operators' contains the operators to use for comparison. All three sequences must be the same length: The nth operator is used to compare the nth column with the nth value.

Each operator must be one of "!=","=",<=",<,>=" or ">".

As an example, the following call would split a DynamicFrame so that the first output frame would contain records of people over 65 from the United States, and the second would contain all other records.

```
{{{  
    df.splitRows(Seq("age", "address.country"), Seq(65, "USA"), Seq("&gt;=", "="))  
}}}
```

[def stageErrorsCount](#)

```
def stageErrorsCount
```

Returns the number of error records created while computing this DynamicFrame. This excludes errors from previous operations that were passed into this DynamicFrame as input.

[def toDF](#)

```
def toDF( specs : Seq[ResolveSpec] = Seq.empty[ResolveSpec] ) : DataFrame
```

Converts this DynamicFrame to an Apache Spark SQL DataFrame with the same schema and records.

Note

Because DataFrames don't support ChoiceTypes, this method automatically converts ChoiceType columns into StructTypes. For more information and options for resolving choice, see [resolveChoice \(p. 405\)](#).

def unbox

```
def unbox( path : String,
          format : String,
          optionString : String = "{}",
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided"),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

- **path** — The column to parse. Must be a string or binary.
- **format** — The format to use for parsing.
- **optionString** — Options to pass to the format, such as the CSV separator.

Parses an embedded string or binary column according to the specified format. Parsed columns are nested under a struct with the original column name.

For example, suppose that you have a CSV file with an embedded JSON column.

```
name, age, address
Sally, 36, {"state": "NE", "city": "Omaha"}
...
```

After an initial parse, you would get a `DynamicFrame` with the following schema.

```
{{{  
  root  
  |-- name: string  
  |-- age: int  
  |-- address: string  
}}}
```

You can call `unbox` on the address column to parse the specific components.

```
{{{  
  df.unbox("address", "json")  
}}}
```

This gives us a `DynamicFrame` with the following schema.

```
{{{  
  root  
  |-- name: string  
  |-- age: int  
  |-- address: struct  
    |-- state: string  
    |-- city: string  
}}}
```

def unnest

```
def unnest( transformationContext : String = "",
           callSite : CallSite = CallSite("Not Provided"),
           stageThreshold : Long = 0,
           totalThreshold : Long = 0
         ) : DynamicFrame
```

Returns a new `DynamicFrame` with all nested structures flattened. Names are constructed using the `'.'` (period) character.

For example, suppose that you have a `DynamicFrame` with the following schema.

```
{{{  
    root  
    |-- name: string  
    |-- age: int  
    |-- address: struct  
    |    |-- state: string  
    |    |-- city: string  
}}}
```

The following call unnests the address struct.

```
{{{  
    df.unnest()  
}}}
```

The resulting schema is as follows.

```
{{{  
    root  
    |-- name: string  
    |-- age: int  
    |-- address.state: string  
    |-- address.city: string  
}}}
```

This method also unnests nested structs inside of arrays. But for historical reasons, the names of such fields are prepended with the name of the enclosing array and `".val"`.

[def withFrameSchema](#)

```
def withFrameSchema( getSchema : () => Schema ) : DynamicFrame
```

- `getSchema` — A function that returns the schema to use. Specified as a zero-parameter function to defer potentially expensive computation.

Sets the schema of this `DynamicFrame` to the specified value. This is primarily used internally to avoid costly schema recomputation. The passed-in schema must contain all columns present in the data.

[def withName](#)

```
def withName( name : String ) : DynamicFrame
```

- `name` — The new name to use.

Returns a copy of this `DynamicFrame` with a new name.

[def withTransformationContext](#)

```
def withTransformationContext( ctx : String ) : DynamicFrame
```

Returns a copy of this DynamicFrame with the specified transformation context.

The DynamicFrame Object

Package: com.amazonaws.services.glue

```
object DynamicFrame
```

def apply

```
def apply( df : DataFrame,  
          glueContext : GlueContext  
        ) : DynamicFrame
```

def emptyDynamicFrame

```
def emptyDynamicFrame( glueContext : GlueContext ) : DynamicFrame
```

def fromPythonRDD

```
def fromPythonRDD( rdd : JavaRDD[Array[Byte]],  
                   glueContext : GlueContext  
                 ) : DynamicFrame
```

def ignoreErrors

```
def ignoreErrors( fn : DynamicRecord => DynamicRecord ) : DynamicRecord
```

def inlineErrors

```
def inlineErrors( msg : String,  
                  callSite : CallSite  
                ) : (DynamicRecord => DynamicRecord)
```

def newFrameWithErrors

```
def newFrameWithErrors( prevFrame : DynamicFrame,  
                       rdd : RDD[DynamicRecord],  
                       name : String = "",  
                       transformationContext : String = "",  
                       callSite : CallSite,  
                       stageThreshold : Long,  
                       totalThreshold : Long  
                     ) : DynamicFrame
```

AWS Glue Scala DynamicRecord Class

Topics

- [def addField \(p. 412\)](#)
- [def dropField \(p. 412\)](#)
- [def setError \(p. 412\)](#)
- [def isError \(p. 413\)](#)
- [def getError \(p. 413\)](#)

- [def clearError \(p. 413\)](#)
- [def write \(p. 413\)](#)
- [def readFields \(p. 413\)](#)
- [def clone \(p. 413\)](#)
- [def schema \(p. 413\)](#)
- [def getRoot \(p. 413\)](#)
- [def toJson \(p. 413\)](#)
- [def getFieldNode \(p. 414\)](#)
- [def getField \(p. 414\)](#)
- [def hashCode \(p. 414\)](#)
- [def equals \(p. 414\)](#)
- [DynamicRecord Object \(p. 414\)](#)
- [RecordTraverser Trait \(p. 414\)](#)

Package: com.amazonaws.services.glue

```
class DynamicRecord extends Serializable with Writable with Cloneable
```

A `DynamicRecord` is a self-describing data structure that represents a row of data in the dataset that is being processed. It is self-describing in the sense that you can get the schema of the row that is represented by the `DynamicRecord` by inspecting the record itself. A `DynamicRecord` is similar to a `Row` in Apache Spark.

def addField

```
def addField( path : String,
              dynamicNode : DynamicNode
            ) : Unit
```

Adds a [DynamicNode \(p. 429\)](#) to the specified path.

- `path` — The path for the field to be added.
- `dynamicNode` — The [DynamicNode \(p. 429\)](#) to be added at the specified path.

def dropField

```
def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

Drops a [DynamicNode \(p. 429\)](#) from the specified path and returns the dropped node if there is not an array in the specified path.

- `path` — The path to the field to drop.
- `underRename` — True if `dropField` is called as part of a rename transform, or false otherwise (false by default).

Returns a `scala.Option[DynamicNode]`.

def setError

```
def setError( error : Error )
```

Sets this record as an error record, as specified by the `error` parameter.

Returns a `DynamicRecord`.

`def isError`

```
def isError
```

Checks whether this record is an error record.

`def getError`

```
def getError
```

Gets the `Error` if the record is an error record. Returns `scala.Some(Some(Error))` if this record is an error record, or otherwise `scala.None`.

`def clearError`

```
def clearError
```

Set the `Error` to `scala.None`.

`def write`

```
override def write( out : DataOutput ) : Unit
```

`def readFields`

```
override def readFields( in : DataInput ) : Unit
```

`def clone`

```
override def clone : DynamicRecord
```

Clones this record to a new `DynamicRecord` and returns it.

`def schema`

```
def schema
```

Gets the Schema by inspecting the record.

`def getRoot`

```
def getRoot : ObjectNode
```

Gets the root `ObjectNode` for the record.

`def toJson`

```
def toJson : String
```

Gets the JSON string for the record.

def getFieldNode

```
def getFieldNode( path : String ) : Option[DynamicNode]
```

Gets the field's value at the specified path as an option of DynamicNode.

Returns `scala.Some Some (DynamicNode (p. 429))` if the field exists, or otherwise `scala.None.None`.

def getField

```
def getField( path : String ) : Option[Any]
```

Gets the field's value at the specified path as an option of DynamicNode.

Returns `scala.Some Some (value)`.

def hashCode

```
override def hashCode : Int
```

def equals

```
override def equals( other : Any )
```

DynamicRecord Object

```
object DynamicRecord
```

def apply

```
def apply( row : Row,
          schema : SparkStructType )
```

Apply method to convert an Apache Spark SQL Row to a [DynamicRecord \(p. 411\)](#).

- `row` — A Spark SQL Row.
- `schema` — The Schema of that row.

Returns a `DynamicRecord`.

RecordTraverser Trait

```
trait RecordTraverser {
  def nullValue(): Unit
  def byteValue(value: Byte): Unit
  def binaryValue(value: Array[Byte]): Unit
  def booleanValue(value: Boolean): Unit
  def shortValue(value: Short) : Unit
```

```
def intValue(value: Int) : Unit
def longValue(value: Long) : Unit
def floatValue(value: Float): Unit
def doubleValue(value: Double): Unit
def decimalValue(value: BigDecimal): Unit
def stringValue(value: String): Unit
def dateValue(value: Date): Unit
def timestampValue(value: Timestamp): Unit
def objectStart(length: Int): Unit
def objectKey(key: String): Unit
def objectEnd(): Unit
def mapStart(length: Int): Unit
def mapKey(key: String): Unit
def mapEnd(): Unit
def arrayStart(length: Int): Unit
def arrayEnd(): Unit
}
```

AWS Glue Scala GlueContext APIs

Package: com.amazonaws.services.glue

```
class GlueContext extends SQLContext(sc) (
    @transient val sc : SparkContext,
    val defaultSourcePartitioner : PartitioningStrategy )
```

GlueContext is the entry point for reading and writing a [DynamicFrame \(p. 397\)](#) from and to Amazon Simple Storage Service (Amazon S3), the AWS Glue Data Catalog, JDBC, and so on. This class provides utility functions to create [DataSource trait \(p. 396\)](#) and [DataSink \(p. 395\)](#) objects that can in turn be used to read and write DynamicFrames.

You can also use GlueContext to set a target number of partitions (default 20) in the DynamicFrame if the number of partitions created from the source is less than a minimum threshold for partitions (default 10).

def getCatalogSink

```
def getCatalogSink( database : String,
                    tableName : String,
                    redshiftTmpDir : String = "",
                    transformationContext : String = ""
                    additionalOptions: JsonOptions = JsonOptions.empty,
                    catalogId: String = null
                  ) : DataSink
```

Creates a [DataSink \(p. 395\)](#) that writes to a location specified in a table that is defined in the Data Catalog.

- **database** — The database name in the Data Catalog.
- **tableName** — The table name in the Data Catalog.
- **redshiftTmpDir** — The temporary staging directory to be used with certain data sinks. Set to empty by default.
- **transformationContext** — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.
- **additionalOptions** – Additional options provided to AWS Glue.
- **catalogId** — The catalog ID (account ID) of the Data Catalog being accessed. When null, the default account ID of the caller is used.

Returns the DataSink.

def getCatalogSource

```
def getCatalogSource( database : String,
                     tableName : String,
                     redshiftTmpDir : String = "",
                     transformationContext : String = ""
                     pushDownPredicate : String = " "
                     additionalOptions: JsonOptions = JsonOptions.empty,
                     catalogId: String = null
                   ) : DataSource
```

Creates a [DataSource trait \(p. 396\)](#) that reads data from a table definition in the Data Catalog.

- **database** — The database name in the Data Catalog.
- **tableName** — The table name in the Data Catalog.
- **redshiftTmpDir** — The temporary staging directory to be used with certain data sinks. Set to empty by default.
- **transformationContext** — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.
- **pushDownPredicate** — Filters partitions without having to list and read all the files in your dataset. For more information, see [Pre-Filtering Using Pushdown Predicates \(p. 302\)](#).
- **additionalOptions** — Additional options provided to AWS Glue.
- **catalogId** — The catalog ID (account ID) of the Data Catalog being accessed. When null, the default account ID of the caller is used.

Returns the DataSource.

def getJDBCSink

```
def getJDBCSink( catalogConnection : String,
                 options : JsonOptions,
                 redshiftTmpDir : String = "",
                 transformationContext : String = "",
                 catalogId: String = null
               ) : DataSink
```

Creates a [DataSink \(p. 395\)](#) that writes to a JDBC database that is specified in a Connection object in the Data Catalog. The Connection object has information to connect to a JDBC sink, including the URL, user name, password, VPC, subnet, and security groups.

- **catalogConnection** — The name of the connection in the Data Catalog that contains the JDBC URL to write to.
- **options** — A string of JSON name-value pairs that provide additional information that is required to write to a JDBC data store. This includes:
 - **dbtable** (required) — The name of the JDBC table. For JDBC data stores that support schemas within a database, specify schema.table-name. If a schema is not provided, then the default "public" schema is used. The following example shows an options parameter that points to a schema named test and a table named test_table in database test_db.

```
options = JsonOptions("""{"dbtable": "test.test_table", "database": "test_db"}""")
```

- **database** (required) — The name of the JDBC database.

- Any additional options passed directly to the SparkSQL JDBC writer. For more information, see [Redshift data source for Spark](#).
- `redshiftTmpDir` — A temporary staging directory to be used with certain data sinks. Set to empty by default.
- `transformationContext` — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.
- `catalogId` — The catalog ID (account ID) of the Data Catalog being accessed. When null, the default account ID of the caller is used.

The following is an example.

```
getJDBCSink(catalogConnection = "my-connection-name", options = JsonOptions("""{"dbtable": "my-jdbc-table", "database": "my-jdbc-db"}"""), redshiftTmpDir = "", transformationContext = "datasink4")
```

Returns the `DataSink`.

[def getSink](#)

```
def getSink( connectionType : String,
            options : JsonOptions,
            transformationContext : String = ""
          ) : DataSink
```

Creates a [DataSink \(p. 395\)](#) that writes data to a destination like Amazon S3, JDBC, or the AWS Glue Data Catalog.

- `connectionType` — The type of the connection.
- `options` — A string of JSON name-value pairs that provide additional information to establish the connection with the data sink.
- `transformationContext` — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.

Returns the `DataSink`.

[def getSinkWithFormat](#)

```
def getSinkWithFormat( connectionType : String,
                      options : JsonOptions,
                      transformationContext : String = "",
                      format : String = null,
                      formatOptions : JsonOptions = JsonOptions.empty
                    ) : DataSink
```

Creates a [DataSink \(p. 395\)](#) that writes data to a destination like Amazon S3, JDBC, or the Data Catalog, and also sets the format for the data to be written out to the destination.

- `connectionType` — The type of the connection. Refer to [DataSink \(p. 395\)](#) for a list of supported connection types.
- `options` — A string of JSON name-value pairs that provide additional information to establish a connection with the data sink.
- `transformationContext` — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.

- `format` — The format of the data to be written out to the destination.
- `formatOptions` — A string of JSON name-value pairs that provide additional options for formatting data at the destination. See [Format Options \(p. 299\)](#).

Returns the `DataSink`.

`def getSource`

```
def getSource( connectionType : String,
              connectionOptions : JsonOptions,
              transformationContext : String = ""
              pushDownPredicate
            ) : DataSource
```

Creates a [DataSource trait \(p. 396\)](#) that reads data from a source like Amazon S3, JDBC, or the AWS Glue Data Catalog.

- `connectionType` — The type of the data source. Can be one of "s3", "mysql", "redshift", "oracle", "sqlserver", "postgresql", "dynamodb", "parquet", or "orc".
- `connectionOptions` — A string of JSON name-value pairs that provide additional information for establishing a connection with the data source.
 - `connectionOptions` when the `connectionType` is "s3":
 - `paths` (required) — List of Amazon S3 paths to read.
 - `compressionType` (optional) — Compression type of the data. This is generally not required if the data has a standard file extension. Possible values are "gzip" and "bzip".
 - `exclusions` (optional) — A string containing a JSON list of glob patterns to exclude. For example, "[**.pdf\]" excludes all PDF files.
 - `maxBand` (optional) — This advanced option controls the duration in seconds after which AWS Glue expects an Amazon S3 listing to be consistent. Files with modification timestamps falling within the last `maxBand` seconds are tracked when using job bookmarks to account for Amazon S3 eventual consistency. It is rare to set this option. The default is 900 seconds.
 - `maxFilesInBand` (optional) — This advanced option specifies the maximum number of files to save from the last `maxBand` seconds. If this number is exceeded, extra files are skipped and processed only in the next job run.
 - `groupFiles` (optional) — Grouping files is enabled by default when the input contains more than 50,000 files. To disable grouping with fewer than 50,000 files, set this parameter to "inPartition". To disable grouping when there are more than 50,000 files, set this parameter to "none".
 - `groupSize` (optional) — The target group size in bytes. The default is computed based on the input data size and the size of your cluster. When there are fewer than 50,000 input files, `groupFiles` must be set to "inPartition" for this option to take effect.
 - `recurse` (optional) — If set to true, recursively read files in any subdirectory of the specified paths.
 - `connectionOptions` when the `connectionType` is "dynamodb":
 - `dynamodb.input.tableName` (required) — The DynamoDB table from which to read.
 - `dynamodb.throughput.read.percent` (optional) — The percentage of reserved capacity units (RCU) to use. The default is set to "0.5". Acceptable values are from "0.1" to "1.5", inclusive.
 - `connectionOptions` when the `connectionType` is "parquet" or "orc":
 - `paths` (required) — List of Amazon S3 paths to read.
 - Any additional options are passed directly to the SparkSQL `DataSource`. For more information, see [Redshift data source for Spark](#).
 - `connectionOptions` when the `connectionType` is "redshift":
 - `url` (required) — The JDBC URL for an Amazon Redshift database.
 - `dbtable` (required) — The Amazon Redshift table to read.

- *redshiftTmpDir* (required) — The Amazon S3 path where temporary data can be staged when copying out of Amazon Redshift.
- *user* (required) — The username to use when connecting to the Amazon Redshift cluster.
- *password* (required) — The password to use when connecting to the Amazon Redshift cluster.
- *transformationContext* — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.
- *pushDownPredicate* — Predicate on partition columns.

Returns the `DataSource`.

`def getSourceWithFormat`

```
def getSourceWithFormat( connectionType : String,
                        options : JsonOptions,
                        transformationContext : String = "",
                        format : String = null,
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSource
```

Creates a [DataSource trait](#) (p. 396) that reads data from a source like Amazon S3, JDBC, or the AWS Glue Data Catalog, and also sets the format of data stored in the source.

- *connectionType* — The type of the data source. Can be one of "s3", "mysql", "redshift", "oracle", "sqlserver", "postgresql", "dynamodb", "parquet", or "orc".
- *options* — A string of JSON name-value pairs that provide additional information for establishing a connection with the data source.
- *transformationContext* — The transformation context that is associated with the sink to be used by job bookmarks. Set to empty by default.
- *format* — The format of the data that is stored at the source. When the *connectionType* is "s3", you can also specify *format*. Can be one of "avro", "csv", "grokLog", "ion", "json", "xml", "parquet", or "orc".
- *formatOptions* — A string of JSON name-value pairs that provide additional options for parsing data at the source. See [Format Options](#) (p. 299).

Returns the `DataSource`.

`def getSparkSession`

```
def getSparkSession : SparkSession
```

Gets the `SparkSession` object associated with this `GlueContext`. Use this `SparkSession` object to register tables and UDFs for use with `DataFrame` created from `DynamicFrames`.

Returns the `SparkSession`.

`def purgeTable`

```
def purgeTable( databaseName: String,
                tableName: String,
                options: JsonOptions = JsonOptions.empty,
                transformationContext : String = ,
                catalogId: String = "" ) : Unit
```

Deletes files from Amazon S3 for the specified catalog's database and table. If all files in a partition are deleted, that partition is deleted from the catalog too.

If you want to be able to recover deleted objects, you can enable [object versioning](#) on the Amazon S3 bucket. When an object is deleted from a bucket that doesn't have object versioning enabled, the object can't be recovered. For more information about how to recover deleted objects in a version enabled bucket, see [How can I retrieve an Amazon S3 object that was deleted?](#) in the AWS Support Knowledge Center.

- `databaseName` — The name of the database to use.
- `tableName` — The name of the table to use.
- `options` — A string of JSON name-value pairs that provide additional information for this transformation.
- `transformationContext` — The transformation context to use (optional). Used in the manifest file path. Set to empty by default.
- `catalogId` — The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog).

Returns the `Unit`.

Example

```
val pushDownString: String = """(month=='march')"""
val options: JsonOptions = new JsonOptions(
    s"""{"partitionPredicate": $pushDownString,
       |"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],
       |"manifestFilePath": "s3a://aws-glue-example-path/storage-transforms/storage-
example/manifest/"}""".stripMargin)
glueContext.purgeTable(nameSpace, tableName, options)
```

def purgeS3Path

```
def purgeS3Path( s3Path: String,
                 options: JsonOptions = JsonOptions.empty,
                 transformationContext : String = "" ) : Unit
```

Deletes files from the specified Amazon S3 path recursively.

If you want to be able to recover deleted objects, you can enable [object versioning](#) on the Amazon S3 bucket. When an object is deleted from a bucket that doesn't have object versioning enabled, the object can't be recovered. For more information about how to recover deleted objects in a version enabled bucket, see [How can I retrieve an Amazon S3 object that was deleted?](#) in the AWS Support Knowledge Center.

- `s3Path` — The path in Amazon S3 of the files to be transitioned in the format `s3://<bucket>/<prefix>/`
- `options` — A string of JSON name-value pairs that provide additional information for this transformation.
- `transformationContext` — The transformation context to use (optional). Used in the manifest file path. Set to empty by default.

Returns the `Unit`.

Example

```
val options: JsonOptions = new JsonOptions(
```

```
s"""\\"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],  
| "manifestFilePath": "s3a://aws-glue-example-path/storage-transforms/storage-  
example/manifest/"}}""".stripMargin)  
glueContext.purgeS3Path(baseLocation, options)
```

def transitionTable

```
def transitionTable( databaseName: String,  
                    tableName: String,  
                    transitionTo: String,  
                    options: JsonOptions = JsonOptions.empty,  
                    transformationContext : String = "",  
                    catalogId: String = "" ) : Unit
```

Transitions the storage class of the files stored on Amazon S3 for the specified catalog's database and table.

You can transition between any two storage classes. For the `GLACIER` and `DEEP_ARCHIVE` storage classes, you can transition to these classes. However, you would use an `S3 RESTORE` to transition from `GLACIER` and `DEEP_ARCHIVE` storage classes.

If you're running AWS Glue ETL jobs that read files or partitions from Amazon S3, you can exclude some Amazon S3 storage class types. For more information, see [Excluding Amazon S3 Storage Classes](#).

- `databaseName` — The name of the database to use.
- `tableName` — The name of the table to use.
- `transitionTo` — The [Amazon S3 storage class](#) to transition to.
- `options` — A string of JSON name-value pairs that provide additional information for this transformation.
- `transformationContext` — The transformation context to use (optional). Used in the manifest file path. Set to empty by default.
- `catalogId` — The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog).

Returns the `Unit`.

Example

```
val pushDownString: String = """(month=='march')"""
val options: JsonOptions = new JsonOptions(
    s"""\\"partitionPredicate": $pushDownString,  
| \"retentionPeriod": 1,  
| \"manifestFilePath": "s3a://aws-glue-example-path/storage-transforms/storage-  
example/manifest/",  
| \"accountId": "123456789012", "roleArn": "arn:aws:iam::123456789012:user/  
example"}""".stripMargin)
glueContext.transitionTable(nameSpace, tableName, "STANDARD_IA", options)
```

def transitionS3Path

```
def transitionS3Path( s3Path: String,  
                     transitionTo: String,  
                     options: JsonOptions = JsonOptions.empty,  
                     transformationContext : String = "" ) : Unit
```

Transitions the storage class of the files in the specified Amazon S3 path recursively.

You can transition between any two storage classes. For the `GLACIER` and `DEEP_ARCHIVE` storage classes, you can transition to these classes. However, you would use an `S3 RESTORE` to transition from `GLACIER` and `DEEP_ARCHIVE` storage classes.

If you're running AWS Glue ETL jobs that read files or partitions from Amazon S3, you can exclude some Amazon S3 storage class types. For more information, see [Excluding Amazon S3 Storage Classes](#).

- `s3Path` — The path in Amazon S3 of the files to be transitioned in the format `s3://<bucket>/<prefix>/`
- `transitionTo` — The [Amazon S3 storage class](#) to transition to.
- `options` — A string of JSON name-value pairs that provide additional information for this transformation.
- `transformationContext` — The transformation context to use (optional). Used in the manifest file path. Set to empty by default.

Returns the `Unit`.

Example

```
val options: JsonOptions = new JsonOptions(  
    s""""{"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],  
           | "manifestFilePath": "s3a://aws-glue-example-path/storage-transforms/storage-  
           example/manifest/",  
           | "accountId": "123456789012", "roleArn": "arn:aws:iam::123456789012:user/  
           example"}""".stripMargin)  
    glueContext.transitionS3Path(baseLocation, "REDUCED_REDUNDANCY", options)
```

def this

```
def this( sc : SparkContext,  
          minPartitions : Int,  
          targetPartitions : Int )
```

Creates a `GlueContext` object using the specified `SparkContext`, minimum partitions, and target partitions.

- `sc` — The `SparkContext`.
- `minPartitions` — The minimum number of partitions.
- `targetPartitions` — The target number of partitions.

Returns the `GlueContext`.

def this

```
def this( sc : SparkContext )
```

Creates a `GlueContext` object with the provided `SparkContext`. Sets the minimum partitions to 10 and target partitions to 20.

- `sc` — The `SparkContext`.

Returns the `GlueContext`.

def this

```
def this( sparkContext : JavaSparkContext )
```

Creates a `GlueContext` object with the provided `JavaSparkContext`. Sets the minimum partitions to 10 and target partitions to 20.

- `sparkContext` — The `JavaSparkContext`.

Returns the `GlueContext`.

MappingSpec

Package: `com.amazonaws.services.glue`

MappingSpec Case Class

```
case class MappingSpec( sourcePath: SchemaPath,
                        sourceType: DataType,
                        targetPath: SchemaPath,
                        targetType: DataType
                      ) extends Product4[String, String, String, String] {
  override def _1: String = sourcePath.toString
  override def _2: String = ExtendedTypeName.fromDataType(sourceType)
  override def _3: String = targetPath.toString
  override def _4: String = ExtendedTypeName.fromDataType(targetType)
}
```

- `sourcePath` — The `SchemaPath` of the source field.
- `sourceType` — The `DataType` of the source field.
- `targetPath` — The `SchemaPath` of the target field.
- `targetType` — The `DataType` of the target field.

A `MappingSpec` specifies a mapping from a source path and a source data type to a target path and a target data type. The value at the source path in the source frame appears in the target frame at the target path. The source data type is cast to the target data type.

It extends from `Product4` so that you can handle any `Product4` in your `applyMapping` interface.

MappingSpec Object

```
object MappingSpec
```

The `MappingSpec` object has the following members:

val orderingByTarget

```
val orderingByTarget: Ordering[MappingSpec]
```

def apply

```
def apply( sourcePath : String,
          sourceType : DataType,
```

```
    targetPath : String,  
    targetType : DataType  
) : MappingSpec
```

Creates a `MappingSpec`.

- `sourcePath` — A string representation of the source path.
- `sourceType` — The source `DataType`.
- `targetPath` — A string representation of the target path.
- `targetType` — The target `DataType`.

Returns a `MappingSpec`.

def apply

```
def apply( sourcePath : String,  
          sourceTypeString : String,  
          targetPath : String,  
          targetTypeString : String  
) : MappingSpec
```

Creates a `MappingSpec`.

- `sourcePath` — A string representation of the source path.
- `sourceType` — A string representation of the source data type.
- `targetPath` — A string representation of the target path.
- `targetType` — A string representation of the target data type.

Returns a `MappingSpec`.

def apply

```
def apply( product : Product4[String, String, String, String] ) : MappingSpec
```

Creates a `MappingSpec`.

- `product` — The `Product4` of the source path, source data type, target path, and target data type.

Returns a `MappingSpec`.

AWS Glue Scala ResolveSpec APIs

Topics

- [ResolveSpec Object \(p. 424\)](#)
- [ResolveSpec Case Class \(p. 425\)](#)

Package: com.amazonaws.services.glue

ResolveSpec Object

ResolveSpec

```
object ResolveSpec
```

def

```
def apply( path : String,  
          action : String  
        ) : ResolveSpec
```

Creates a ResolveSpec.

- **path** — A string representation of the choice field that needs to be resolved.
- **action** — A resolution action. The action can be one of the following: `Project`, `KeepAsStruct`, or `Cast`.

Returns the ResolveSpec.

def

```
def apply( product : Product2[String, String] ) : ResolveSpec
```

Creates a ResolveSpec.

- **product** — `Product2` of: source path, resolution action.

Returns the ResolveSpec.

ResolveSpec Case Class

```
case class ResolveSpec extends Product2[String, String] {  
    path : SchemaPath,  
    action : String }
```

Creates a ResolveSpec.

- **path** — The `SchemaPath` of the choice field that needs to be resolved.
- **action** — A resolution action. The action can be one of the following: `Project`, `KeepAsStruct`, or `Cast`.

ResolveSpec def Methods

```
def _1 : String
```

```
def _2 : String
```

AWS Glue Scala ArrayNode APIs

Package: `com.amazonaws.services.glue.types`

ArrayNode Case Class

ArrayNode

```
case class ArrayNode extends DynamicNode ( value : ArrayBuffer[DynamicNode] )
```

ArrayNode def Methods

```
def add( node : DynamicNode )
```

```
def clone
```

```
def equals( other : Any )
```

```
def get( index : Int ) : Option[DynamicNode]
```

```
def getValue
```

```
def hashCode : Int
```

```
def isEmpty : Boolean
```

```
def nodeType
```

```
def remove( index : Int )
```

```
def this
```

```
def toIterator : Iterator[DynamicNode]
```

```
def toJson : String
```

```
def update( index : Int, node : DynamicNode )
```

AWS Glue Scala BinaryNode APIs

Package: com.amazonaws.services.glue.types

BinaryNode Case Class

BinaryNode

```
case class BinaryNode extends ScalarNode(value, TypeCode.BINARY) ( value : Array[Byte] )
```

BinaryNode val Fields

- ordering

BinaryNode def Methods

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

AWS Glue Scala BooleanNode APIs

Package: com.amazonaws.services.glue.types

BooleanNode Case Class

BooleanNode

```
case class BooleanNode extends ScalarNode(value, TypeCode.BOOLEAN) (
    value : Boolean )
```

BooleanNode val Fields

- ordering

BooleanNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala ByteNode APIs

Package: com.amazonaws.services.glue.types

ByteNode Case Class

ByteNode

```
case class ByteNode extends ScalarNode(value, TypeCode.BYTE) (
    value : Byte )
```

ByteNode val Fields

- ordering

ByteNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala DateNode APIs

Package: com.amazonaws.services.glue.types

DateNode Case Class

DateNode

```
case class DateNode extends ScalarNode(value, TypeCode.DATE)  (
    value : Date )
```

DateNode val Fields

- ordering

DateNode def Methods

```
def equals( other : Any )
```

```
def this( value : Int )
```

AWS Glue Scala DecimalNode APIs

Package: com.amazonaws.services.glue.types

DecimalNode Case Class

DecimalNode

```
case class DecimalNode extends ScalarNode(value, TypeCode.DECIMAL)  (
    value : BigDecimal )
```

DecimalNode val Fields

- ordering

DecimalNode def Methods

```
def equals( other : Any )
```

```
def this( value : Decimal )
```

AWS Glue Scala DoubleNode APIs

Package: com.amazonaws.services.glue.types

DoubleNode Case Class

DoubleNode

```
case class DoubleNode extends ScalarNode(value, TypeCode.DOUBLE)  (
    value : Double )
```

DoubleNode val Fields

- ordering

DoubleNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala DynamicNode APIs

Topics

- [DynamicNode Class \(p. 429\)](#)
- [DynamicNode Object \(p. 429\)](#)

Package: `com.amazonaws.services.glue.types`

DynamicNode Class

DynamicNode

```
class DynamicNode extends Serializable with Cloneable
```

DynamicNode def Methods

```
def getValue : Any
```

Get plain value and bind to the current record:

```
def nodeType : TypeCode
```

```
def toJson : String
```

Method for debug:

```
def toRow( schema : Schema,
           options : Map[String, ResolveOption]
         ) : Row
```

```
def typeName : String
```

DynamicNode Object

DynamicNode

```
object DynamicNode
```

DynamicNode def Methods

```
def quote( field : String,
           useQuotes : Boolean
         ) : String
```

```
def quote( node : DynamicNode,
```

```
    useQuotes : Boolean
) : String
```

AWS Glue Scala FloatNode APIs

Package: com.amazonaws.services.glue.types

FloatNode Case Class

FloatNode

```
case class FloatNode extends ScalarNode(value, TypeCode.FLOAT) (
    value : Float )
```

FloatNode val Fields

- ordering

FloatNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala IntegerNode APIs

Package: com.amazonaws.services.glue.types

IntegerNode Case Class

IntegerNode

```
case class IntegerNode extends ScalarNode(value, TypeCode.INT) (
    value : Int )
```

IntegerNode val Fields

- ordering

IntegerNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala LongNode APIs

Package: com.amazonaws.services.glue.types

LongNode Case Class

LongNode

```
case class LongNode extends ScalarNode(value, TypeCode.LONG) (
    value : Long )
```

LongNode val Fields

- ordering

LongNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala MapLikeNode APIs

Package: com.amazonaws.services.glue.types

MapLikeNode Class

MapLikeNode

```
class MapLikeNode extends DynamicNode  (
    value : mutable.Map[String, DynamicNode] )
```

MapLikeNode def Methods

```
def clear : Unit
```

```
def get( name : String ) : Option[DynamicNode]
```

```
def getValue
```

```
def has( name : String ) : Boolean
```

```
def isEmpty : Boolean
```

```
def put( name : String,
        node : DynamicNode
      ) : Option[DynamicNode]
```

```
def remove( name : String ) : Option[DynamicNode]
```

```
def toIterator : Iterator[(String, DynamicNode)]
```

```
def toJson : String
```

```
def toJson( useQuotes : Boolean ) : String
```

Example: Given this JSON:

```
{"foo": "bar"}
```

If `useQuotes == true`, `toJson` yields `{"foo": "bar"}`. If `useQuotes == false`, `toJson` yields `{foo: bar}` @return.

AWS Glue Scala MapNode APIs

Package: `com.amazonaws.services.glue.types`

MapNode Case Class

MapNode

```
case class MapNode extends MapLikeNode(value)  (
    value : mutable.Map[String, DynamicNode] )
```

MapNode def Methods

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala NullNode APIs

Topics

- [NullNode Class \(p. 432\)](#)
- [NullNode Case Object \(p. 432\)](#)

Package: `com.amazonaws.services.glue.types`

NullNode Class

NullNode

```
class NullNode
```

NullNode Case Object

NullNode

```
case object NullNode extends NullNode
```

AWS Glue Scala ObjectNode APIs

Topics

- [ObjectNode Object \(p. 433\)](#)
- [ObjectNode Case Class \(p. 433\)](#)

Package: com.amazonaws.services.glue.types

ObjectNode Object

ObjectNode

```
object ObjectNode
```

ObjectNode def Methods

```
def apply( frameKeys : Set[String],  
          v1 : mutable.Map[String, DynamicNode],  
          v2 : mutable.Map[String, DynamicNode],  
          resolveWith : String  
        ) : ObjectNode
```

ObjectNode Case Class

ObjectNode

```
case class ObjectNode extends MapLikeNode(value) {  
    val value : mutable.Map[String, DynamicNode] }
```

ObjectNode def Methods

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala ScalarNode APIs

Topics

- [ScalarNode Class \(p. 433\)](#)
- [ScalarNode Object \(p. 434\)](#)

Package: com.amazonaws.services.glue.types

ScalarNode Class

ScalarNode

```
class ScalarNode extends DynamicNode {  
    value : Any,  
    scalarType : TypeCode }
```

ScalarNode def Methods

```
def compare( other : Any,  
            operator : String  
          ) : Boolean
```

```
def getValue
```

```
def hashCode : Int
```

```
def nodeType
```

```
def toJson
```

ScalarNode Object

ScalarNode

```
object ScalarNode
```

ScalarNode def Methods

```
def apply( v : Any ) : DynamicNode
```

```
def compare( tv : Ordered[T],  
            other : T,  
            operator : String  
          ) : Boolean
```

```
def compareAny( v : Any,  
                y : Any,  
                o : String )
```

```
def withEscapedSpecialCharacters( jsonToEscape : String ) : String
```

AWS Glue Scala ShortNode APIs

Package: com.amazonaws.services.glue.types

ShortNode Case Class

ShortNode

```
case class ShortNode extends ScalarNode(value, TypeCode.SHORT) {  
    value : Short }
```

ShortNode val Fields

- ordering

ShortNode def Methods

```
def equals( other : Any )
```

AWS Glue Scala StringNode APIs

Package: com.amazonaws.services.glue.types

StringNode Case Class

StringNode

```
case class StringNode extends ScalarNode(value, TypeCode.STRING)  (  
    value : String )
```

StringNode val Fields

- ordering

StringNode def Methods

```
def equals( other : Any )
```

```
def this( value : UTF8String )
```

AWS Glue Scala TimestampNode APIs

Package: com.amazonaws.services.glue.types

TimestampNode Case Class

TimestampNode

```
case class TimestampNode extends ScalarNode(value, TypeCode.TIMESTAMP)  (  
    value : Timestamp )
```

TimestampNode val Fields

- ordering

TimestampNode def Methods

```
def equals( other : Any )
```

```
def this( value : Long )
```

AWS Glue Scala GlueArgParser APIs

Package: com.amazonaws.services.glue.util

GlueArgParser Object

GlueArgParser

```
object GlueArgParser
```

This is strictly consistent with the Python version of `utils.getResolvedOptions` in the `AWSGlueDataplanePython` package.

GlueArgParser def Methods

```
def getResolvedOptions( args : Array[String],  
                      options : Array[String]  
                    ) : Map[String, String]
```

```
def initParser( userOptionsSet : mutable.Set[String] ) : ArgumentParser
```

AWS Glue Scala Job APIs

Package: com.amazonaws.services.glue.util

Job Object

Job

```
object Job
```

Job def Methods

```
def commit
```

```
def init( jobName : String,  
         glueContext : GlueContext,  
         args : java.util.Map[String, String] = Map[String, String]().asJava  
       ) : this.type
```

```
def init( jobName : String,  
         glueContext : GlueContext,  
         endpoint : String,  
         args : java.util.Map[String, String]  
       ) : this.type
```

```
def isInitialized
```

```
def reset
```

```
def runId
```

Matching Records with AWS Lake Formation FindMatches

AWS Lake Formation provides machine learning capabilities to create custom transforms to cleanse your data. There is currently one available transform named FindMatches. The FindMatches transform enables you to identify duplicate or matching records in your dataset, even when the records do not have a common unique identifier and no fields match exactly. This will not require writing any code or knowing how machine learning works. FindMatches can be useful in many different problems, such as:

- **Matching Customers:** Linking customer records across different customer databases, even when many customer fields do not match exactly across the databases (e.g. different name spelling, address differences, missing or inaccurate data, etc).
- **Matching Products:** Matching products in your catalog against other product sources, such as product catalog against a competitor's catalog, where entries are structured differently.
- **Improving Fraud Detection:** Identifying duplicate customer accounts, determining when a newly created account is (or might be) a match for a previously known fraudulent user.
- **Other Matching Problems:** Match addresses, movies, parts lists, etc etc. In general, if a human being could look at your database rows and determine that they were a match, there is a really good chance that the FindMatches transform can help you.

You can create these transforms when you create a job. The transform that you create is based on a source data store schema and example data that you label (we call this process “teaching” a transform). In this process we generate a file which you label and then upload back which the transform would in a manner learn from). After you teach your transform, you can call it from your Spark-based AWS Glue job (PySpark or Scala Spark) and use it in other scripts with a compatible source data store.

After the transform is created, it is stored in AWS Glue. On the AWS Glue console, you can manage the transforms that you create. On the AWS Glue **ML transforms** tab, you can edit and continue to teach your machine learning transform. For more information about managing transforms on the console, see [Working with Machine Learning Transforms on the AWS Glue Console \(p. 444\)](#).

Types of Machine Learning Transforms

You can create machine learning transforms to cleanse your data. You can call these transforms from your ETL script. Your data passes from transform to transform in a data structure called a *DynamicFrame*, which is an extension to an Apache Spark SQL *DataFrame*. The *DynamicFrame* contains your data, and you reference its schema to process your data.

The following types of machine learning transforms are available:

Find matches

Finds duplicate records in the source data. You teach this machine learning transform by labeling example datasets to indicate which rows match. The machine learning transform learns which rows should be matches the more you teach it with example labeled data. Depending on how you configure the transform, the output is one of the following:

- A copy of the input table plus a `match_id` column filled in with values that indicate matching sets of records. The `match_id` column is an arbitrary identifier. Any records which have the same `match_id` have been identified as matching to each other. Records with different `match_id`'s do not match.
- A copy of the input table with duplicate rows removed. If multiple duplicates are found, then the record with the lowest primary key is kept.

Find Matches Transform

You can use the `FindMatches` transform to find duplicate records in the source data. A labeling file is generated or provided to help teach the transform.

Getting Started Using the Find Matches Transform

Follow these steps to get started with the `FindMatches` transform:

1. Create a table in the AWS Glue Data Catalog for the source data that is to be cleaned. For information about how to create a crawler, see [Working with Crawlers on the AWS Glue Console](#).

If your source data is a text-based file such as a comma-separated values (CSV) file, consider the following:

- Keep your input record CSV file and labeling file in separate folders. Otherwise, the AWS Glue crawler might consider them as multiple parts of the same table and create tables in the Data Catalog incorrectly.
 - Unless your CSV file includes ASCII characters only, ensure that UTF-8 without BOM (byte order mark) encoding is used for the CSV files. Microsoft Excel often adds a BOM in the beginning of UTF-8 CSV files. To remove it, open the CSV file in a text editor, and resave the file as **UTF-8 without BOM**.
2. On the AWS Glue console, create a job, and choose the **Find matches** transform type.

Important

The data source table that you choose for the job can't have more than 100 columns.

3. Tell AWS Glue to generate a labeling file by choosing **Generate labeling file**. AWS Glue takes the first pass at grouping similar records for each `labeling_set_id` so that you can review those groupings. You label matches in the `label` column.
 - If you already have a labeling file, that is, an example of records that indicate matching rows, upload the file to Amazon Simple Storage Service (Amazon S3). For information about the format of the labeling file, see [Labeling File Format \(p. 439\)](#). Proceed to step 4.
4. Download the labeling file and label the file as described in the [Labeling \(p. 439\)](#) section.
5. Upload the corrected labelled file. AWS Glue runs tasks to teach the transform how to find matches.

On the **Machine learning transforms** list page, choose the **History** tab. This page indicates when AWS Glue performs the following tasks:

- **Import labels**
 - **Export labels**
 - **Generate labels**
 - **Estimate quality**
6. To create a better transform, you can iteratively download, label, and upload the labelled file. In the initial runs, a lot more records might be mismatched. But AWS Glue learns as you continue to teach it by verifying the labeling file.
 7. Evaluate and tune your transform by evaluating performance and results of finding matches. For more information, see [Tuning Machine Learning Transforms in AWS Glue \(p. 441\)](#).

Labeling

When `FindMatches` generates a labeling file, records are selected from your source table. Based on previous training, `FindMatches` identifies the most valuable records to learn from.

The act of *labeling* is editing a labeling file (we suggest using a spreadsheet such as Microsoft Excel) and adding identifiers, or labels, into the `label` column that identifies matching and nonmatching records. It is important to have a clear and consistent definition of a match in your source data. `FindMatches` learns from which records you designate as matches (or not) and uses your decisions to learn how to find duplicate records.

When a labeling file is generated by `FindMatches`, approximately 100 records are generated. These 100 records are typically divided into 10 *labeling sets*, where each labeling set is identified by a unique `labeling_set_id` generated by `FindMatches`. Each labeling set should be viewed as a separate labeling task independent of the other labeling sets. Your task is to identify matching and non-matching records within each labeling set.

Tips for Editing Labeling Files in a Spreadsheet

When editing the labeling file in a spreadsheet application, consider the following:

- The file might not open with column fields fully expanded. You might need to expand the `labeling_set_id` and `label` columns to see content in those cells.
- If the primary key column is a number, such as a long data type, the spreadsheet might interpret it as a number and change the value. This key value must be treated as text. To correct this problem, format all the cells in the primary key column as **Text data**.

Labeling File Format

The labeling file that is generated by AWS Glue to teach your `FindMatches` transform uses the following format. If you generate your own file for AWS Glue, it must follow this format as well:

- It is a comma-separated values (CSV) file.
- It must be encoded in UTF-8. If you edit the file using Microsoft Windows, it might be encoded with cp1252.
- It must be in an Amazon S3 location to pass it to AWS Glue.
- Use a moderate number of rows for each labeling task. 10–20 rows per task are recommended, although 2–30 rows per task are acceptable. Tasks larger than 50 rows are not recommended and may cause poor results or system failure.
- If you have already-labeled data consisting of pairs of records labeled as a "match" or a "no-match", this is fine. These labeled pairs can be represented as labeling sets of size 2. In this case label both records with, for instance, a letter "A" if they match, but label one as "A" and one as "B" if they do not match.

Note

Because it has additional columns, the labeling file has a different schema from a file that contains your source data. Place the labeling file in a different folder from any transform input CSV file so that the AWS Glue crawler does not consider it when it creates tables in the Data Catalog. Otherwise, the tables created by the AWS Glue crawler might not correctly represent your data.

- The first two columns (`labeling_set_id`, `label`) are required by AWS Glue. The remaining columns must match the schema of the data that is to be processed.
- For each `labeling_set_id`, you identify all matching records by using the same label. A label is a unique string placed in the `label` column. We recommend using labels that contain simple characters, such as A, B, C, and so on. Labels are case sensitive and are entered in the `label` column.

- Rows that contain the same `labeling_set_id` and the same label are understood to be labeled as a match.
- Rows that contain the same `labeling_set_id` and a different label are understood to be labeled as *not* a match
- Rows that contain a different `labeling_set_id` are understood to be conveying no information for or against matching.

The following is an example of labeling the data:

<code>labeling_set_id</code>	<code>label</code>	<code>first_name</code>	<code>last_name</code>	<code>Birthday</code>
ABC123	A	John	Doe	04/01/1980
ABC123	B	Jane	Smith	04/03/1980
ABC123	A	Johnny	Doe	04/01/1980
ABC123	A	Jon	Doe	04/01/1980
DEF345	A	Richard	Jones	12/11/1992
DEF345	A	Rich	Jones	11/12/1992
DEF345	B	Sarah	Jones	12/11/1992
DEF345	C	Richie	Jones Jr.	05/06/2017
DEF345	B	Sarah	Jones-Walker	12/11/1992
GHI678	A	Robert	Miller	1/3/1999
GHI678	A	Bob	Miller	1/3/1999
XYZABC	A	William	Robinson	2/5/2001
XYZABC	B	Andrew	Robinson	2/5/1971

- In the above example we identify John/Johnny/Jon Doe as being a match and we teach the system that these records do not match Jane Smith. Separately, we teach the system that Richard and Rich Jones are the same person, but that these records are not a match to Sarah Jones/Jones-Walker and Richie Jones Jr.
- As you can see, the scope of the labels is limited to the `labeling_set_id`. So labels do not cross `labeling_set_id` boundaries. For example, a label "A" in `labeling_set_id` 1 does not have any relation to label "A" in `labeling_set_id` 2.
- If a record does not have any matches within a labeling set, then assign it a unique label. For instance, Jane Smith does not match any record in labeling set ABC123, so it is the only record in that labeling set with the label of B.
- The labeling set "GHI678" shows that a labeling set can consist of just two records which are given the same label to show that they match. Similarly, "XYZABC" shows two records given different labels to show that they do not match.
- Note that sometimes a labeling set may contain no matches (that is, you give every record in the labeling set a different label) or a labeling set might all be "the same" (you gave them all the same label). This is okay as long as your labeling sets collectively contain examples of records that are and are not "the same" by your criteria.

Important

Confirm that the IAM role that you pass to AWS Glue has access to the Amazon S3 bucket that contains the labeling file. By convention, AWS Glue policies grant permission to Amazon S3 buckets or folders whose names are prefixed with **aws-glue-**. If your labeling files are in a different location, add permission to that location in the IAM role.

Tuning Machine Learning Transforms in AWS Glue

You can tune your machine learning transforms in AWS Glue to improve the results of your data-cleansing jobs to meet your objectives. To improve your transform, you can teach it by generating a labeling set, adding labels, and then repeating these steps several times until you get your desired results. You can also tune by changing some machine learning parameters.

For more information about machine learning transforms, see [Matching Records with AWS Lake Formation FindMatches \(p. 437\)](#).

Topics

- [Machine Learning Measurements \(p. 441\)](#)
- [Deciding Between Precision and Recall \(p. 442\)](#)
- [Deciding Between Accuracy and Cost \(p. 442\)](#)
- [Teaching the Find Matches Transform \(p. 443\)](#)

Machine Learning Measurements

To understand the measurements that are used to tune your machine learning transform, you should be familiar with the following terminology:

True positive (TP)

A match in the data that the transform correctly found, sometimes called a *hit*.

True negative (TN)

A nonmatch in the data that the transform correctly rejected.

False positive (FP)

A nonmatch in the data that the transform incorrectly classified as a match, sometimes called a *false alarm*.

False negative (FN)

A match in the data that the transform didn't find, sometimes called a *miss*.

For more information about the terminology that is used in machine learning, see [Confusion matrix in Wikipedia](#).

To tune your machine learning transforms, you can change the value of the following measurements in the **Advanced properties** of the transform.

- **Precision** measures how well the transform finds true positives from the total true positives possible. For more information, see [Precision and recall](#) in Wikipedia.
- **Recall** measures how well the transform finds true positives from the total records in the source data. For more information, see [Precision and recall](#) in Wikipedia.
- **Accuracy** measures how well the transform finds true positives and true negatives. Increasing accuracy requires more machine resources and cost. But it also results in increased recall. For more information, see [Accuracy and precision](#) in Wikipedia.

- **Cost** measures how many compute resources (and thus money) are consumed to run the transform.

Deciding Between Precision and Recall

Each `FindMatches` transform contains a `precision-recall` parameter. You use this parameter to specify one of the following:

- If you are more concerned about the transform falsely reporting that two records match when they actually don't match, then you should emphasize *precision*.
- If you are more concerned about the transform failing to detect records that really do match, then you should emphasize *recall*.

You can make this trade-off on the AWS Glue console or by using the AWS Glue machine learning API operations.

When to Favor Precision

Favor precision if you are more concerned about the risk that `FindMatches` results in a pair of records matching when they don't actually match. To favor precision, choose a *higher* precision-recall trade-off value. With a higher value, the `FindMatches` transform requires more evidence to decide that a pair of records should be matched. The transform is tuned to bias toward saying that records do not match.

For example, suppose that you're using `FindMatches` to detect duplicate items in a video catalog, and you provide a higher precision-recall value to the transform. If your transform incorrectly detects that *Star Wars: A New Hope* is the same as *Star Wars: The Empire Strikes Back*, a customer who wants *A New Hope* might be shown *The Empire Strikes Back*. This would be a poor customer experience.

However, if the transform fails to detect that *Star Wars: A New Hope* and *Star Wars: Episode IV—A New Hope* are the same item, the customer might be confused at first but might eventually recognize them as the same. It would be a mistake, but not as bad as the previous scenario.

When to Favor Recall

Favor recall if you are more concerned about the risk that the `FindMatches` transform results might fail to detect a pair of records that actually do match. To favor recall, choose a *lower* precision-recall trade-off value. With a lower value, the `FindMatches` transform requires less evidence to decide that a pair of records should be matched. The transform is tuned to bias toward saying that records do match.

For example, this might be a priority for a security organization. Suppose that you are matching customers against a list of known defrauders, and it is important to determine whether a customer is a defrauder. You are using `FindMatches` to match the defrauder list against the customer list. Every time `FindMatches` detects a match between the two lists, a human auditor is assigned to verify that the person is, in fact, a defrauder. Your organization might prefer to choose recall over precision. In other words, you would rather have the auditors manually review and reject some cases when the customer is not a defrauder than fail to identify that a customer is, in fact, on the defrauder list.

How to Favor Both Precision and Recall

The best way to improve both precision and recall is to label more data. As you label more data, the overall accuracy of the `FindMatches` transform improves, thus improving both precision and recall. Nevertheless, even with the most accurate transform, there is always a gray area where you need to experiment with favoring precision or recall, or choose a value in the middle.

Deciding Between Accuracy and Cost

Each `FindMatches` transform contains an `accuracy-cost` parameter. You can use this parameter to specify one of the following:

- If you are more concerned with the transform accurately reporting that two records match, then you should emphasize *accuracy*.
- If you are more concerned about the cost or speed of running the transform, then you should emphasize *lower cost*.

You can make this trade-off on the AWS Glue console or by using the AWS Glue machine learning API operations.

When to Favor Accuracy

Favor accuracy if you are more concerned about the risk that the `find matches` results won't contain matches. To favor accuracy, choose a *higher* accuracy-cost trade-off value. With a higher value, the `FindMatches` transform requires more time to do a more thorough search for correctly matching records. Note that this parameter doesn't make it less likely to falsely call a nonmatching record pair a match. The transform is tuned to bias towards spending more time finding matches.

When to Favor Cost

Favor cost if you are more concerned about the cost of running the `find matches` transform and less about how many matches are found. To favor cost, choose a *lower* accuracy-cost trade-off value. With a lower value, the `FindMatches` transform requires fewer resources to run. The transform is tuned to bias towards finding fewer matches. If the results are acceptable when favoring lower cost, use this setting.

How to Favor Both Accuracy and Lower Cost

It takes more machine time to examine more pairs of records to determine whether they might be matches. If you want to reduce cost without reducing quality, here are some steps you can take:

- Eliminate records in your data source that you aren't concerned about matching.
- Eliminate columns from your data source that you are sure aren't useful for making a match/no-match decision. A good way of deciding this is to eliminate columns that you don't think affect your own decision about whether a set of records is "the same."

Teaching the Find Matches Transform

Each `FindMatches` transform must be taught what should be considered a match and what should not be considered a match. You teach your transform by adding labels to a file and uploading your choices to AWS Glue.

You can orchestrate this labeling on the AWS Glue console or by using the AWS Glue machine learning API operations.

How Many Times Should I Add Labels? How Many Labels Do I Need?

The answers to these questions are mostly up to you. You must evaluate whether `FindMatches` is delivering the level of accuracy that you need and whether you think the extra labeling effort is worth it for you. The best way to decide this is to look at the "Precision," "Recall," and "Area under the precision recall curve" metrics that you can generate when you choose **Estimate quality** on the AWS Glue console. After you label more sets of tasks, rerun these metrics and verify whether they have improved. If, after labeling a few sets of tasks, you don't see improvement on the metric that you are focusing on, the transform quality might have reached a plateau.

Why Are Both True Positive and True Negative Labels Needed?

The `FindMatches` transform needs both positive and negative examples to learn what you think is a match. If you are labeling `FindMatches`-generated training data (for example, using the **I do not have labels** option), `FindMatches` tries to generate a set of "label set ids" for you. Within each task, you

give the same “label” to some records and different “labels” to other records. In other words, the tasks generally are not either all the same or all different (but it’s okay if a particular task is all “the same” or all “not the same”).

If you are teaching your `FindMatches` transform using the **Upload labels from S3** option, try to include both examples of matching and nonmatching records. It’s acceptable to have only one type. These labels help you build a more accurate `FindMatches` transform, but you still need to label some records that you generate using the **Generate labeling file** option.

How Can I Enforce That the Transform Matches Exactly as I Taught It?

The `FindMatches` transform learns from the labels that you provide, so it might generate records pairs that don’t respect the provided labels. To enforce that the `FindMatches` transform respects your labels, select **EnforceProvidedLabels** in **FindMatchesParameter**.

What Techniques Can You Use When an ML Transform Identifies Items as Matches That Are Not True Matches?

You can use the following techniques:

- Increase the `precisionRecallTradeoff` to a higher value. This eventually results in finding fewer matches, but it should also break up your big cluster when it reaches a high enough value.
- Take the output rows corresponding to the incorrect results and reformat them as a labeling set (removing the `match_id` column and adding a `labeling_set_id` and `label` column). If necessary, break up (subdivide) into multiple labeling sets to ensure that the labeler can keep each labeling set in mind while assigning labels. Then, correctly label the matching sets and upload the label file and append it to your existing labels. This might teach your transformer enough about what it is looking for to understand the pattern.
- (Advanced) Finally, look at that data to see if there is a pattern that you can detect that the system is not noticing. Preprocess that data using standard AWS Glue functions to *normalize* the data. Highlight what you want the algorithm to learn from by separating data that you know to be differently important into their own columns. Or construct combined columns from columns whose data you know to be related.

Working with Machine Learning Transforms on the AWS Glue Console

You can use AWS Glue to create custom machine learning transforms that can be used to cleanse your data. You can use these transforms when you create a job on the AWS Glue console.

For information about how to create a machine learning transform, see [Matching Records with AWS Lake Formation FindMatches \(p. 437\)](#).

Topics

- [Transform Properties \(p. 444\)](#)
- [Adding and Editing Machine Learning Transforms \(p. 445\)](#)
- [Viewing Transform Details \(p. 445\)](#)

Transform Properties

To view an existing machine learning transform, sign in to the AWS Management Console, and open the AWS Glue console at <https://console.aws.amazon.com/glue/>. Then choose the **ML transforms** tab.

The **Machine Learning Transforms** list displays the following properties for each transform:

Transform name

The unique name you gave the transform when you created it.

Transform ID

A unique identifier of the transform.

Type

The type of machine learning transform; for example, **Find matching records**.

Glue version

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#).

Status

Indicates whether the transform is **Ready** or **Needs teaching**. To run a machine learning transform successfully in a job, it must be **Ready**.

When you create a **FindMatches** transform, you specify the following configuration information:

Primary key

The name of a column that uniquely identifies rows in the source table.

Type

The type of machine learning transform; for example, **Find matches**.

Merge matching records

Indicates whether the transform is to remove duplicates in the target. The record with the lowest primary key value is written to the output of the transform.

Adding and Editing Machine Learning Transforms

You can view, delete, set up and teach, or tune a transform on the AWS Glue console. Select the check box next to the transform in the list, choose **Action**, and then choose the action that you want to take.

To add a new machine learning transform, choose the **Jobs** tab, and then choose **Add job**. Follow the instructions in the **Add job** wizard to add a job with a machine learning transform such as **FindMatches**. For more information, see [Matching Records with AWS Lake Formation FindMatches \(p. 437\)](#).

Viewing Transform Details

Transform details include the information that you defined when you created the transform. To view the details of a transform, select the transform in the **Machine learning transforms** list, and review the information on the following tabs:

- History
- Details
- Estimate quality

History

The **History** tab shows your transform task run history. Several types of tasks are run to teach a transform. For each task, the run metrics include the following:

- **Run ID** is an identifier created by AWS Glue for each run of this task.
- **Task type** shows the type of task run.
- **Status** shows the success of each task listed with the most recent run at the top.
- **Error** shows the details of an error message if the run was not successful.
- **Start time** shows the date and time (local time) that the task started.
- **Execution time** shows the length of time during which the job run consumed resources. The amount is calculated from when the job run starts consuming resources until it finishes.
- **Last modified** shows the date and time (local time) that the task was last modified.
- **Logs** links to the logs written to `stdout` for this job run.

The **Logs** link takes you to Amazon CloudWatch Logs. There you can view the details about the tables that were created in the AWS Glue Data Catalog and any errors that were encountered. You can manage your log retention period on the CloudWatch console. The default log retention is **Never Expire**. For more information about how to change the retention period, see [Change Log Data Retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

- **Error logs** links to the logs written to `stderr` for this task run.

This link takes you to CloudWatch Logs, where you can see details about any errors that were encountered. You can manage your log retention period on the CloudWatch console. The default log retention is **Never Expire**. For more information about how to change the retention period, see [Change Log Data Retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

- **Download label file** shows a link to Amazon S3 for a generated labeling file.

Details

The **Details** tab includes attributes of your transform. It shows you the details about the transform definition, including the following:

- **Transform name** shows the name of the transform.
- **Type** lists the type of the transform.
- **Status** displays whether the transform is ready to be used in a script or job.
- **Force output to match labels** displays whether the transform forces the output to match the labels provided by the user.

Estimate quality

The **Estimate quality** tab shows the metrics that you use to measure the quality of the transform. Estimates are calculated by comparing the transform match predictions using a subset of your labeled data against the labels you have provided. These estimates are approximate. You can invoke an **Estimate quality** task run from this tab.

The **Estimate quality** tab shows the metrics from the last **Estimate quality** run including the following properties:

- **Area under the Precision-Recall curve** is a single number estimating the upper bound of the overall quality of the transform. It is independent of the choice made for the precision-recall parameter. Higher values indicate that you have a more attractive precision-recall tradeoff.

- **Precision** estimates how often the transform is correct when it predicts a match.
- **Recall upper limit** estimates that for an actual match, how often the transform predicts the match.
- **Max F1** estimates the transform's accuracy between 0 and 1, where 1 is the best accuracy. For more information, see [F1 score](#) in Wikipedia.

For information about understanding quality estimates versus true quality, see [Quality Estimates Versus End-to-End \(True\) Quality \(p. 447\)](#).

For more information about tuning your transform, see [Tuning Machine Learning Transforms in AWS Glue \(p. 441\)](#).

Quality Estimates Versus End-to-End (True) Quality

In the `FindMatches` machine learning transform, AWS Glue estimates the quality of your transform by presenting the internal machine-learned model with a number of pairs of records that you provided matching labels for but that the model has not seen before. These quality estimates are a function of the quality of the machine-learned model (which is influenced by the number of records that you label to "teach" the transform). The end-to-end, or *true* recall (which is not automatically calculated by the `FindMatches` transform) is also influenced by the `FindMatches` filtering mechanism that proposes a wide variety of possible matches to the machine-learned model.

You can tune this filtering method primarily by using the **Lower Cost-Accuracy** slider. As you move this slider closer to the **Accuracy** end, the system does a more thorough and expensive search for pairs of records that might be matches. More pairs of records are fed to your machine-learned model, and your `FindMatches` transform's end-to-end or true recall gets closer to the estimated recall metric. As a result, changes in the end-to-end quality of your matches as a result of changes in your matches's cost/accuracy tradeoff will typically not be reflected in the quality estimate.

Tutorial: Creating a Machine Learning Transform with AWS Glue

This tutorial guides you through the actions to create and manage a machine learning (ML) transform using AWS Glue. Before using this tutorial, you should be familiar with using the AWS Glue console to add crawlers and jobs and edit scripts. You should also be familiar with finding and downloading files on the Amazon Simple Storage Service (Amazon S3) console.

In this example, you create a `FindMatches` transform to find matching records, teach it how to identify matching and nonmatching records, and use it in an AWS Glue job. The AWS Glue job writes a new Amazon S3 file with an additional column named `match_id`.

The source data used by this tutorial is a file named `dblp_acm_records.csv`. This file is a modified version of academic publications (DBLP and ACM) available from the original [DBLP ACM dataset](#). The `dblp_acm_records.csv` file is a comma-separated values (CSV) file in UTF-8 format with no byte-order mark (BOM).

A second file, `dblp_acm_labels.csv`, is an example labeling file that contains both matching and nonmatching records used to teach the transform as part of the tutorial.

Topics

- [Step 1: Crawl the Source Data \(p. 448\)](#)
- [Step 2: Add a Machine Learning Transform \(p. 448\)](#)
- [Step 3: Teach Your Machine Learning Transform \(p. 448\)](#)
- [Step 4: Estimate the Quality of Your Machine Learning Transform \(p. 449\)](#)

- [Step 5: Add and Run a Job with Your Machine Learning Transform \(p. 449\)](#)
- [Step 6: Verify Output Data from Amazon S3 \(p. 451\)](#)

Step 1: Crawl the Source Data

First, crawl the source Amazon S3 CSV file to create a corresponding metadata table in the Data Catalog.

Important

To direct the crawler to create a table for only the CSV file, store the CSV source data in a different Amazon S3 folder from other files.

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Crawlers, Add crawler**.
3. Follow the wizard to create and run a crawler named `demo-crawl-db1p-acm` with output to database `demo-db-db1p-acm`. When running the wizard, create the database `demo-db-db1p-acm` if it doesn't already exist. Choose an Amazon S3 include path to sample data in the current AWS Region. For example, for us-east-1, the Amazon S3 include path to the source file is `s3://ml-transforms-public-datasets-us-east-1/db1p-acm/records/db1p_acm_records.csv`.

If successful, the crawler creates the table `db1p_acm_records_csv` with the following columns: `id`, `title`, `authors`, `venue`, `year`, and `source`.

Step 2: Add a Machine Learning Transform

Next, add a machine learning transform that is based on the schema of your data source table created by the crawler named `demo-crawl-db1p-acm`.

1. On the AWS Glue console, in the navigation pane, choose **ML Transforms, Add transform**. Then follow the wizard to create a `Find matches` transform with the following properties.
 - a. For **Transform name**, enter `demo-xform-db1p-acm`. This is the name of the transform that is used to find matches in the source data.
 - b. For **IAM role**, choose an IAM role that has permission to the Amazon S3 source data, labeling file, and AWS Glue API operations. For more information, see [Create an IAM Role for AWS Glue](#) in the *AWS Glue Developer Guide*.
 - c. For **Data source**, choose the table named `db1p_acm_records_csv` in database `demo-db-db1p-acm`.
 - d. For **Primary key**, choose the primary key column for the table, `id`.
 - e. For **Predictive columns**, choose the `title`, `authors`, `venue`, `year`, and `source` columns in the data source to act as predictive columns.
2. In the wizard, choose **Finish** and return to the **ML transforms** list.

Step 3: Teach Your Machine Learning Transform

Next, you teach your machine learning transform using the tutorial sample labeling file.

You can't use a machine language transform in an extract, transform, and load (ETL) job until its status is **Ready for use**. To get your transform ready, you must teach it how to identify matching and nonmatching records by providing examples of matching and nonmatching records. To teach your transform, you can **Generate a label file**, add labels, and then **Upload label file**. In this tutorial, you can use the example labeling file named `db1p_acm_labels.csv`. For more information about the labeling process, see [Labeling \(p. 439\)](#).

1. On the AWS Glue console, in the navigation pane, choose **ML Transforms**.
2. Choose the `demo-xform-dblp-acm` transform, and then choose **Action**, **Teach**. Follow the wizard to teach your `Find matches` transform.
3. On the transform properties page, choose **I have labels**. Choose an Amazon S3 path to the sample labeling file in the current AWS Region. For example, for us-east-1, upload the provided labeling file from the Amazon S3 path `s3://ml-transforms-public-datasets-us-east-1/dblp-acm/labels/dblp_acm_labels.csv` with the option to **overwrite** existing labels. The labeling file must be located in Amazon S3 in the same Region as the AWS Glue console.

When you upload a labeling file, a task is started in AWS Glue to add or overwrite the labels used to teach the transform how to process the data source.

4. On the final page of the wizard, choose **Finish**, and return to the **ML transforms** list.

Step 4: Estimate the Quality of Your Machine Learning Transform

Next, you can estimate the quality of your machine learning transform. The quality depends on how much labeling you have done. For more information about estimating quality, see [Estimate quality \(p. 446\)](#).

1. On the AWS Glue console, in the navigation pane, choose **ML Transforms**.
2. Choose the `demo-xform-dblp-acm` transform, and choose the **Estimate quality** tab. This tab displays the current quality estimates, if available, for the transform.
3. Choose **Estimate quality** to start a task to estimate the quality of the transform. The accuracy of the quality estimate is based on the labeling of the source data.
4. Navigate to the **History** tab. In this pane, task runs are listed for the transform, including the **Estimating quality** task. For more details about the run, choose **Logs**. Check that the run status is **Succeeded** when it finishes.

Step 5: Add and Run a Job with Your Machine Learning Transform

In this step, you use your machine learning transform to add and run a job in AWS Glue. When the transform `demo-xform-dblp-acm` is **Ready for use**, you can use it in an ETL job.

1. On the AWS Glue console, in the navigation pane, choose **Jobs**.
2. Choose **Add job**, and follow the steps in the wizard to create an ETL Spark job with a generated script. Choose the following property values for your transform:
 - a. For **Name**, choose the example job in this tutorial, `demo-etl-dblp-acm`.
 - b. For **IAM role**, choose an IAM role with permission to the Amazon S3 source data, labeling file, and AWS Glue API operations. For more information, see [Create an IAM Role for AWS Glue](#) in the *AWS Glue Developer Guide*.
 - c. For **ETL language**, choose **Scala**. This is the programming language in the ETL script.
 - d. For **Script file name**, choose `demo-etl-dblp-acm`. This is the file name of the Scala script (same as the job name).
 - e. For **Data source**, choose `dblp_acm_records_csv`. The data source you choose must match the machine learning transform data source schema.
 - f. For **Transform type**, choose **Find matching records** to create a job using a machine learning transform.

- g. Clear **Remove duplicate records**. You don't want to remove duplicate records because the output records written have an additional `match_id` field added.
 - h. For **Transform**, choose **demo-xform-dblp-acm**, the machine learning transform used by the job.
 - i. For **Create tables in your data target**, choose to create tables with the following properties:
 - **Data store type** — **Amazon S3**
 - **Format** — **CSV**
 - **Compression type** — **None**
 - **Target path** — The Amazon S3 path where the output of the job is written (in the current console AWS Region)
3. Choose **Save job and edit script** to display the script editor page.
 4. Edit the script to add a statement to cause the job output to the **Target path** to be written to a single partition file. Add this statement immediately following the statement that runs the `FindMatches` transform. The statement is similar to the following.

```
val single_partition = findmatches1.repartition(1)
```

You must modify the `.writeDynamicFrame(findmatches1)` statement to write the output as `.writeDynamicFrame(single_partition)`.

5. After you edit the script, choose **Save**. The modified script looks similar to the following code, but customized for your environment.

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.ml.FindMatches
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    // @type: DataSource
    // @args: [database = "demo-db-dblp-acm", table_name = "dblp_acm_records_csv",
    transformation_ctx = "datasource0"]
    // @return: datasource0
    // @inputs: []
    val datasource0 = glueContext.getCatalogSource(database = "demo-db-dblp-acm",
    tableName = "dblp_acm_records_csv", redshiftTmpDir = "", transformationContext =
    "datasource0").getDynamicFrame()
    // @type: FindMatches
    // @args: [transformId = "tfm-123456789012", emitFusion = false,
    survivorComparisonField = "<primary_id>", transformation_ctx = "findmatches1"]
    // @return: findmatches1
    // @inputs: [frame = datasource0]
    val findmatches1 = FindMatches.apply(frame = datasource0, transformId =
    "tfm-123456789012", transformationContext = "findmatches1")

    // Repartition the previous DynamicFrame into a single partition.
    val single_partition = findmatches1.repartition(1)
```

```
// @type: DataSink
// @args: [connection_type = "s3", connection_options = {"path": "s3://aws-glue-ml-transforms-data/sal"}, format = "csv", transformation_ctx = "datasink2"]
// @return: datasink2
// @inputs: [frame = findmatches1]
val datasink2 = glueContext.getSinkWithFormat(connectionType =
"s3", options = JsonOptions("""{"path": "s3://aws-glue-ml-transforms-data/sal"}"""), transformationContext = "datasink2", format =
"csv").writeDynamicFrame(single_partition)
Job.commit()
}
}
```

6. Choose **Run job** to start the job run. Check the status of the job in the jobs list. When the job finishes, in the **ML transform, History** tab, there is a new **Run ID** row added of type **ETL job**.
7. Navigate to the **Jobs, History** tab. In this pane, job runs are listed. For more details about the run, choose **Logs**. Check that the run status is **Succeeded** when it finishes.

Step 6: Verify Output Data from Amazon S3

In this step, you check the output of the job run in the Amazon S3 bucket that you chose when you added the job. You can download the output file to your local machine and verify that matching records were identified.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Download the target output file of the job `demo-etl-db1p-acm`. Open the file in a spreadsheet application (you might need to add a file extension `.csv` for the file to properly open).

The following image shows an excerpt of the output in Microsoft Excel.

A	B	C	D	E	F	G	
id	title	authors	venue	year	source	match_id	
2	journals/sigmod/Mackay99	Semantic Integration of Environmental Models for Application to Global Information S.D. Scott Mackay	SIGMOD Record	1999	DBLP	0	
3	309852	Semantic integration of environmental models for application to global information s'D. Scott Mackay	ACM SIGMOD Recor	1999	ACM	0	
4	conf/vldb/Poosala196	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. VLDB	1996	DBLP	1	
5	673321	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. VLDB	1996	ACM	1	
6	conf/vldb/PalpanasCP02	Incremental Maintenance for Non-Distributive Aggregate Functions	Themistoklis Palpanas, Richar VLDB	2002	DBLP	2	
7	conf/vldb/GardarinGT96	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Zhao-Hui Tang, Georges Gardarin, Rober VLDB	1996	DBLP	3	
8	673484	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Georges Gardarin, Jean-Rober Very Large Data Bas	1996	ACM	3	
9	673135	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	Very Large Data Bas	1995	ACM	4
10	conf/vldb/Hoel95	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	VLDB	1995	DBLP	4
11	304219	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	International Confe	1999	ACM	5
12	conf/sigmod/Keim99	Efficient Geometry-based Similarity Search of 3D Spatial Databases	Daniel A. Keim	SIGMOD Conference	1999	DBLP	5
13	journals/sigmod/Ouksel02	Mining the World Wide Web: An Information Search Approach - Book Review	Aris M. Ouksel	SIGMOD Record	2002	DBLP	6
14	journals/vldb/Seshadri98	Enhanced Abstract Data Types in Object-Relational Databases	Praveen Seshadri	VLDB J.	1998	DBLP	7
15	765531	Enhanced abstract data types in object-relational databases	Praveen Seshadri	The VLDB Journal &	1998	ACM	7
16	journals/sigmod/Ramamritham96	Report on DART '96: Databases: Active and Real-Time (Concepts meet Practice)	Nandit Soparkar, Krithi Ramamritham	SIGMOD Record	1997	DBLP	8
17	262769	Report on DART '96: databases: active and real-time (concepts meet practice)	Krithi Ramamritham, Nandit S ACM SIGMOD Recor	1997	ACM	8	
18	234902	UniSQL's next-generation object-relational database management system	Albert D'Andrea, Phil Janus	ACM SIGMOD Recor	1996	ACM	9
19	journals/sigmod/DAndrea96	UniSQL's Next-Generation Object-Relational Database Management System	Phil Janus, Albert D'Andrea	SIGMOD Record	1996	DBLP	9

The data source and target file both have 4,911 records. However, the `Find matches` transform adds another column named `match_id` to identify matching records in the output. Rows with the same `match_id` are considered matching records.

3. Sort the output file by `match_id` to easily see which records are matches. Compare the values in the other columns to see if you agree with the results of the `Find matches` transform. If you don't, you can continue to teach the transform by adding more labels.

You can also sort the file by another field, such as `title`, to see if records with similar titles have the same `match_id`.

AWS Glue API

Contents

- [Security APIs in AWS Glue \(p. 459\)](#)
 - [Data Types \(p. 459\)](#)
 - [DataCatalogEncryptionSettings Structure \(p. 459\)](#)
 - [EncryptionAtRest Structure \(p. 460\)](#)
 - [ConnectionPasswordEncryption Structure \(p. 460\)](#)
 - [EncryptionConfiguration Structure \(p. 461\)](#)
 - [S3Encryption Structure \(p. 461\)](#)
 - [CloudWatchEncryption Structure \(p. 461\)](#)
 - [JobBookmarksEncryption Structure \(p. 461\)](#)
 - [SecurityConfiguration Structure \(p. 462\)](#)
 - [Operations \(p. 462\)](#)
 - [GetDataCatalogEncryptionSettings Action \(Python: get_data_catalog_encryption_settings\) \(p. 462\)](#)
 - [PutDataCatalogEncryptionSettings Action \(Python: put_data_catalog_encryption_settings\) \(p. 463\)](#)
 - [PutResourcePolicy Action \(Python: put_resource_policy\) \(p. 463\)](#)
 - [GetResourcePolicy Action \(Python: get_resource_policy\) \(p. 464\)](#)
 - [DeleteResourcePolicy Action \(Python: delete_resource_policy\) \(p. 465\)](#)
 - [CreateSecurityConfiguration Action \(Python: create_security_configuration\) \(p. 465\)](#)
 - [DeleteSecurityConfiguration Action \(Python: delete_security_configuration\) \(p. 466\)](#)
 - [GetSecurityConfiguration Action \(Python: get_security_configuration\) \(p. 466\)](#)
 - [GetSecurityConfigurations Action \(Python: get_security_configurations\) \(p. 467\)](#)
- [Catalog API \(p. 467\)](#)
 - [Database API \(p. 468\)](#)
 - [Data Types \(p. 468\)](#)
 - [Database Structure \(p. 468\)](#)
 - [DatabaseInput Structure \(p. 468\)](#)
 - [PrincipalPermissions Structure \(p. 469\)](#)
 - [DataLakePrincipal Structure \(p. 469\)](#)
 - [Operations \(p. 469\)](#)
 - [CreateDatabase Action \(Python: create_database\) \(p. 470\)](#)
 - [UpdateDatabase Action \(Python: update_database\) \(p. 470\)](#)
 - [DeleteDatabase Action \(Python: delete_database\) \(p. 471\)](#)
 - [GetDatabase Action \(Python: get_database\) \(p. 471\)](#)
 - [GetDatabases Action \(Python: get_databases\) \(p. 472\)](#)
 - [Table API \(p. 473\)](#)
 - [Data Types \(p. 473\)](#)
 - [Table Structure \(p. 473\)](#)
 - [TableInput Structure \(p. 474\)](#)
 - [Column Structure \(p. 475\)](#)
 - [StorageDescriptor Structure \(p. 476\)](#)
 - [SerDeInfo Structure \(p. 477\)](#)

- [Order Structure \(p. 477\)](#)
- [SkewedInfo Structure \(p. 478\)](#)
- [TableVersion Structure \(p. 478\)](#)
- [TableError Structure \(p. 478\)](#)
- [TableVersionError Structure \(p. 478\)](#)
- [SortCriterion Structure \(p. 479\)](#)
- [Operations \(p. 479\)](#)
- [CreateTable Action \(Python: create_table\) \(p. 479\)](#)
- [UpdateTable Action \(Python: update_table\) \(p. 480\)](#)
- [DeleteTable Action \(Python: delete_table\) \(p. 481\)](#)
- [BatchDeleteTable Action \(Python: batch_delete_table\) \(p. 481\)](#)
- [GetTable Action \(Python: get_table\) \(p. 482\)](#)
- [GetTables Action \(Python: get_tables\) \(p. 483\)](#)
- [GetTableVersion Action \(Python: get_table_version\) \(p. 484\)](#)
- [GetTableVersions Action \(Python: get_table_versions\) \(p. 484\)](#)
- [DeleteTableVersion Action \(Python: delete_table_version\) \(p. 485\)](#)
- [BatchDeleteTableVersion Action \(Python: batch_delete_table_version\) \(p. 486\)](#)
- [SearchTables Action \(Python: search_tables\) \(p. 487\)](#)
- [Partition API \(p. 488\)](#)
 - [Data Types \(p. 488\)](#)
 - [Partition Structure \(p. 488\)](#)
 - [PartitionInput Structure \(p. 489\)](#)
 - [PartitionSpecWithSharedStorageDescriptor Structure \(p. 489\)](#)
 - [PartitionListComposingSpec Structure \(p. 489\)](#)
 - [PartitionSpecProxy Structure \(p. 490\)](#)
 - [PartitionValueList Structure \(p. 490\)](#)
 - [Segment Structure \(p. 490\)](#)
 - [PartitionError Structure \(p. 491\)](#)
 - [Operations \(p. 491\)](#)
 - [CreatePartition Action \(Python: create_partition\) \(p. 491\)](#)
 - [BatchCreatePartition Action \(Python: batch_create_partition\) \(p. 492\)](#)
 - [UpdatePartition Action \(Python: update_partition\) \(p. 493\)](#)
 - [DeletePartition Action \(Python: delete_partition\) \(p. 493\)](#)
 - [BatchDeletePartition Action \(Python: batch_delete_partition\) \(p. 494\)](#)
 - [GetPartition Action \(Python: get_partition\) \(p. 495\)](#)
 - [GetPartitions Action \(Python: get_partitions\) \(p. 495\)](#)
 - [BatchGetPartition Action \(Python: batch_get_partition\) \(p. 498\)](#)
- [Connection API \(p. 499\)](#)
 - [Data Types \(p. 499\)](#)
 - [Connection Structure \(p. 499\)](#)
 - [ConnectionInput Structure \(p. 501\)](#)
 - [PhysicalConnectionRequirements Structure \(p. 501\)](#)
 - [GetConnectionsFilter Structure \(p. 502\)](#)
 - [Operations \(p. 502\)](#)
 - [CreateConnection Action \(Python: create_connection\) \(p. 502\)](#)

- [DeleteConnection Action \(Python: delete_connection\) \(p. 503\)](#)
- [GetConnection Action \(Python: get_connection\) \(p. 503\)](#)
- [GetConnections Action \(Python: get_connections\) \(p. 504\)](#)
- [UpdateConnection Action \(Python: update_connection\) \(p. 505\)](#)
- [BatchDeleteConnection Action \(Python: batch_delete_connection\) \(p. 505\)](#)
- [User-Defined Function API \(p. 506\)](#)
 - [Data Types \(p. 506\)](#)
 - [UserDefinedFunction Structure \(p. 506\)](#)
 - [UserDefinedFunctionInput Structure \(p. 507\)](#)
 - [Operations \(p. 507\)](#)
 - [CreateUserDefinedFunction Action \(Python: create_user_defined_function\) \(p. 507\)](#)
 - [UpdateUserDefinedFunction Action \(Python: update_user_defined_function\) \(p. 508\)](#)
 - [DeleteUserDefinedFunction Action \(Python: delete_user_defined_function\) \(p. 509\)](#)
 - [GetUserDefinedFunction Action \(Python: get_user_defined_function\) \(p. 509\)](#)
 - [GetUserDefinedFunctions Action \(Python: get_user_defined_functions\) \(p. 510\)](#)
- [Importing an Athena Catalog to AWS Glue \(p. 511\)](#)
 - [Data Types \(p. 511\)](#)
 - [CatalogImportStatus Structure \(p. 511\)](#)
 - [Operations \(p. 511\)](#)
 - [ImportCatalogToGlue Action \(Python: import_catalog_to_glue\) \(p. 511\)](#)
 - [GetCatalogImportStatus Action \(Python: get_catalog_import_status\) \(p. 512\)](#)
- [Crawlers and Classifiers API \(p. 512\)](#)
 - [Classifier API \(p. 513\)](#)
 - [Data Types \(p. 513\)](#)
 - [Classifier Structure \(p. 513\)](#)
 - [GrokClassifier Structure \(p. 513\)](#)
 - [XMLClassifier Structure \(p. 514\)](#)
 - [JsonClassifier Structure \(p. 515\)](#)
 - [CsvClassifier Structure \(p. 515\)](#)
 - [CreateGrokClassifierRequest Structure \(p. 516\)](#)
 - [UpdateGrokClassifierRequest Structure \(p. 516\)](#)
 - [CreateXMLClassifierRequest Structure \(p. 517\)](#)
 - [UpdateXMLClassifierRequest Structure \(p. 517\)](#)
 - [CreateJsonClassifierRequest Structure \(p. 517\)](#)
 - [UpdateJsonClassifierRequest Structure \(p. 518\)](#)
 - [CreateCsvClassifierRequest Structure \(p. 518\)](#)
 - [UpdateCsvClassifierRequest Structure \(p. 518\)](#)
 - [Operations \(p. 519\)](#)
 - [CreateClassifier Action \(Python: create_classifier\) \(p. 519\)](#)
 - [DeleteClassifier Action \(Python: delete_classifier\) \(p. 520\)](#)
 - [GetClassifier Action \(Py⁴⁵⁴: get_classifier\) \(p. 520\)](#)
 - [GetClassifiers Action \(Python: get_classifiers\) \(p. 521\)](#)
 - [UpdateClassifier Action \(Python: update_classifier\) \(p. 521\)](#)

- [Crawler API \(p. 522\)](#)
 - [Data Types \(p. 522\)](#)
 - [Crawler Structure \(p. 522\)](#)
 - [Schedule Structure \(p. 523\)](#)
 - [CrawlerTargets Structure \(p. 523\)](#)
 - [S3Target Structure \(p. 524\)](#)
 - [JdbcTarget Structure \(p. 524\)](#)
 - [DynamoDBTarget Structure \(p. 524\)](#)
 - [CatalogTarget Structure \(p. 525\)](#)
 - [CrawlerMetrics Structure \(p. 525\)](#)
 - [SchemaChangePolicy Structure \(p. 525\)](#)
 - [LastCrawlInfo Structure \(p. 526\)](#)
 - [Operations \(p. 526\)](#)
 - [CreateCrawler Action \(Python: create_crawler\) \(p. 527\)](#)
 - [DeleteCrawler Action \(Python: delete_crawler\) \(p. 528\)](#)
 - [GetCrawler Action \(Python: get_crawler\) \(p. 528\)](#)
 - [GetCrawlers Action \(Python: get_crawlers\) \(p. 529\)](#)
 - [GetCrawlerMetrics Action \(Python: get_crawler_metrics\) \(p. 529\)](#)
 - [UpdateCrawler Action \(Python: update_crawler\) \(p. 530\)](#)
 - [StartCrawler Action \(Python: start_crawler\) \(p. 531\)](#)
 - [StopCrawler Action \(Python: stop_crawler\) \(p. 531\)](#)
 - [BatchGetCrawlers Action \(Python: batch_get_crawlers\) \(p. 532\)](#)
 - [ListCrawlers Action \(Python: list_crawlers\) \(p. 532\)](#)
 - [Crawler Scheduler API \(p. 533\)](#)
 - [Data Types \(p. 533\)](#)
 - [Schedule Structure \(p. 533\)](#)
 - [Operations \(p. 533\)](#)
 - [UpdateCrawlerSchedule Action \(Python: update_crawler_schedule\) \(p. 533\)](#)
 - [StartCrawlerSchedule Action \(Python: start_crawler_schedule\) \(p. 534\)](#)
 - [StopCrawlerSchedule Action \(Python: stop_crawler_schedule\) \(p. 534\)](#)
 - [Autogenerating ETL Scripts API \(p. 535\)](#)
 - [Data Types \(p. 535\)](#)
 - [CodeGenNode Structure \(p. 535\)](#)
 - [CodeGenNodeArg Structure \(p. 536\)](#)
 - [CodeGenEdge Structure \(p. 536\)](#)
 - [Location Structure \(p. 536\)](#)
 - [CatalogEntry Structure \(p. 537\)](#)
 - [MappingEntry Structure \(p. 537\)](#)
 - [Operations \(p. 537\)](#)
 - [CreateScript Action \(Python: create_script\) \(p. 537\)](#)
 - [GetDataflowGraph Action \(Python: get_dataflow_graph\) \(p. 538\)](#)
 - [GetMapping Action \(Python: get_mapping\) \(p. 539\)](#)
 - [GetPlan Action \(Python: get_plan\) \(p. 539\)](#)
 - [Jobs API \(p. 540\)](#)
 - [Jobs \(p. 540\)](#)
-

- [Data Types \(p. 540\)](#)
- [Job Structure \(p. 540\)](#)
- [ExecutionProperty Structure \(p. 542\)](#)
- [NotificationProperty Structure \(p. 543\)](#)
- [JobCommand Structure \(p. 543\)](#)
- [ConnectionsList Structure \(p. 543\)](#)
- [JobUpdate Structure \(p. 543\)](#)
- [Operations \(p. 545\)](#)
- [CreateJob Action \(Python: create_job\) \(p. 545\)](#)
- [UpdateJob Action \(Python: update_job\) \(p. 548\)](#)
- [GetJob Action \(Python: get_job\) \(p. 548\)](#)
- [GetJobs Action \(Python: get_jobs\) \(p. 549\)](#)
- [DeleteJob Action \(Python: delete_job\) \(p. 549\)](#)
- [ListJobs Action \(Python: list_jobs\) \(p. 550\)](#)
- [BatchGetJobs Action \(Python: batch_get_jobs\) \(p. 550\)](#)
- [Job Runs \(p. 551\)](#)
 - [Data Types \(p. 551\)](#)
 - [JobRun Structure \(p. 551\)](#)
 - [Predecessor Structure \(p. 554\)](#)
 - [JobBookmarkEntry Structure \(p. 554\)](#)
 - [BatchStopJobRunSuccessfulSubmission Structure \(p. 554\)](#)
 - [BatchStopJobRunError Structure \(p. 555\)](#)
 - [Operations \(p. 555\)](#)
 - [StartJobRun Action \(Python: start_job_run\) \(p. 555\)](#)
 - [BatchStopJobRun Action \(Python: batch_stop_job_run\) \(p. 557\)](#)
 - [GetJobRun Action \(Python: get_job_run\) \(p. 558\)](#)
 - [GetJobRuns Action \(Python: get_job_runs\) \(p. 558\)](#)
 - [GetJobBookmark Action \(Python: get_job_bookmark\) \(p. 559\)](#)
 - [GetJobBookmarks Action \(Python: get_job_bookmarks\) \(p. 559\)](#)
 - [ResetJobBookmark Action \(Python: reset_job_bookmark\) \(p. 560\)](#)
- [Triggers \(p. 561\)](#)
 - [Data Types \(p. 561\)](#)
 - [Trigger Structure \(p. 561\)](#)
 - [TriggerUpdate Structure \(p. 562\)](#)
 - [Predicate Structure \(p. 562\)](#)
 - [Condition Structure \(p. 562\)](#)
 - [Action Structure \(p. 563\)](#)
 - [Operations \(p. 564\)](#)
 - [CreateTrigger Action \(Python: create_trigger\) \(p. 564\)](#)
 - [StartTrigger Action \(Python: start_trigger\) \(p. 565\)](#)
 - [GetTrigger Action \(Python: get_trigger\) \(p. 566\)](#)
 - [GetTriggers Action \(Python: get_triggers\) \(p. 566\)](#)
 - [UpdateTrigger Action \(Python: update_trigger\) \(p. 567\)](#)
 - [StopTrigger Action \(Python: stop_trigger\) \(p. 567\)](#)
 - [DeleteTrigger Action \(Python: delete_trigger\) \(p. 568\)](#)

- [ListTriggers Action \(Python: list_triggers\) \(p. 568\)](#)
 - [BatchGetTriggers Action \(Python: batch_get_triggers\) \(p. 569\)](#)
 - [Workflows \(p. 570\)](#)
 - [Data Types \(p. 570\)](#)
 - [JobNodeDetails Structure \(p. 570\)](#)
 - [CrawlerNodeDetails Structure \(p. 570\)](#)
 - [TriggerNodeDetails Structure \(p. 570\)](#)
 - [Crawl Structure \(p. 571\)](#)
 - [Node Structure \(p. 571\)](#)
 - [Edge Structure \(p. 572\)](#)
 - [WorkflowGraph Structure \(p. 572\)](#)
 - [WorkflowRun Structure \(p. 572\)](#)
 - [WorkflowRunStatistics Structure \(p. 573\)](#)
 - [Workflow Structure \(p. 573\)](#)
 - [Operations \(p. 574\)](#)
 - [CreateWorkflow Action \(Python: create_workflow\) \(p. 574\)](#)
 - [UpdateWorkflow Action \(Python: update_workflow\) \(p. 575\)](#)
 - [DeleteWorkflow Action \(Python: delete_workflow\) \(p. 576\)](#)
 - [ListWorkflows Action \(Python: list_workflows\) \(p. 576\)](#)
 - [BatchGetWorkflows Action \(Python: batch_get_workflows\) \(p. 577\)](#)
 - [GetWorkflowRun Action \(Python: get_workflow_run\) \(p. 577\)](#)
 - [GetWorkflowRuns Action \(Python: get_workflow_runs\) \(p. 578\)](#)
 - [GetWorkflowRunProperties Action \(Python: get_workflow_run_properties\) \(p. 579\)](#)
 - [PutWorkflowRunProperties Action \(Python: put_workflow_run_properties\) \(p. 579\)](#)
 - [Development Endpoints API \(p. 580\)](#)
 - [Data Types \(p. 580\)](#)
 - [DevEndpoint Structure \(p. 580\)](#)
 - [DevEndpointCustomLibraries Structure \(p. 583\)](#)
 - [Operations \(p. 583\)](#)
 - [CreateDevEndpoint Action \(Python: create_dev_endpoint\) \(p. 583\)](#)
 - [UpdateDevEndpoint Action \(Python: update_dev_endpoint\) \(p. 587\)](#)
 - [DeleteDevEndpoint Action \(Python: delete_dev_endpoint\) \(p. 588\)](#)
 - [GetDevEndpoint Action \(Python: get_dev_endpoint\) \(p. 588\)](#)
 - [GetDevEndpoints Action \(Python: get_dev_endpoints\) \(p. 589\)](#)
 - [BatchGetDevEndpoints Action \(Python: batch_get_dev_endpoints\) \(p. 590\)](#)
 - [ListDevEndpoints Action \(Python: list_dev_endpoints\) \(p. 590\)](#)
 - [AWS Glue Machine Learning API \(p. 591\)](#)
 - [Data Types \(p. 591\)](#)
 - [TransformParameters Structure \(p. 592\)](#)
 - [EvaluationMetrics Structure \(p. 592\)](#)
 - [MLTransform Structure \(p. 592\)](#)
 - [FindMatchesParameters Structure \(p. 594\)](#)
 - [FindMatchesMetrics Structure \(p. 595\)](#)
 - [ConfusionMatrix Structure \(p. 596\)](#)
 - [GlueTable Structure \(p. 596\)](#)
-

- [TaskRun Structure \(p. 597\)](#)
- [TransformFilterCriteria Structure \(p. 597\)](#)
- [TransformSortCriteria Structure \(p. 598\)](#)
- [TaskRunFilterCriteria Structure \(p. 598\)](#)
- [TaskRunSortCriteria Structure \(p. 599\)](#)
- [TaskRunProperties Structure \(p. 599\)](#)
- [FindMatchesTaskRunProperties Structure \(p. 599\)](#)
- [ImportLabelsTaskRunProperties Structure \(p. 600\)](#)
- [ExportLabelsTaskRunProperties Structure \(p. 600\)](#)
- [LabelingSetGenerationTaskRunProperties Structure \(p. 600\)](#)
- [SchemaColumn Structure \(p. 600\)](#)
- [Operations \(p. 601\)](#)
- [CreateMLTransform Action \(Python: create_ml_transform\) \(p. 601\)](#)
- [UpdateMLTransform Action \(Python: update_ml_transform\) \(p. 603\)](#)
- [DeleteMLTransform Action \(Python: delete_ml_transform\) \(p. 605\)](#)
- [GetMLTransform Action \(Python: get_ml_transform\) \(p. 605\)](#)
- [GetMLTransforms Action \(Python: get_ml_transforms\) \(p. 607\)](#)
- [StartMLEvaluationTaskRun Action \(Python: start_ml_evaluation_task_run\) \(p. 608\)](#)
- [StartMLLabelingSetGenerationTaskRun Action \(Python: start_ml_labeling_set_generation_task_run\) \(p. 609\)](#)
- [GetMLTaskRun Action \(Python: get_ml_task_run\) \(p. 609\)](#)
- [GetMLTaskRuns Action \(Python: get_ml_task_runs\) \(p. 611\)](#)
- [CancelMLTaskRun Action \(Python: cancel_ml_task_run\) \(p. 611\)](#)
- [StartExportLabelsTaskRun Action \(Python: start_export_labels_task_run\) \(p. 612\)](#)
- [StartImportLabelsTaskRun Action \(Python: start_import_labels_task_run\) \(p. 613\)](#)
- [Tagging APIs in AWS Glue \(p. 614\)](#)
 - [Data Types \(p. 614\)](#)
 - [Tag Structure \(p. 614\)](#)
 - [Operations \(p. 614\)](#)
 - [TagResource Action \(Python: tag_resource\) \(p. 614\)](#)
 - [UntagResource Action \(Python: untag_resource\) \(p. 615\)](#)
 - [GetTags Action \(Python: get_tags\) \(p. 616\)](#)
- [Common Data Types \(p. 616\)](#)
 - [Tag Structure \(p. 616\)](#)
 - [DecimalNumber Structure \(p. 617\)](#)
 - [ErrorDetail Structure \(p. 617\)](#)
 - [PropertyPredicate Structure \(p. 617\)](#)
 - [ResourceUri Structure \(p. 617\)](#)
 - [String Patterns \(p. 618\)](#)
- [Exceptions \(p. 618\)](#)
 - [AccessDeniedException Structure \(p. 618\)](#)
 - [AlreadyExistsException Structure \(p. 618\)](#)
 - [ConcurrentModificationException Structure \(p. 619\)](#)
 - [ConcurrentRunsExceededException Structure \(p. 619\)](#)
 - [CrawlerNotRunningException Structure \(p. 619\)](#)
 - [CrawlerRunningException Structure \(p. 619\)](#)

- [CrawlerStoppingException Structure \(p. 619\)](#)
- [EntityNotFoundException Structure \(p. 620\)](#)
- [GlueEncryptionException Structure \(p. 620\)](#)
- [IdempotentParameterMismatchException Structure \(p. 620\)](#)
- [InternalServiceException Structure \(p. 620\)](#)
- [InvalidExecutionEngineException Structure \(p. 620\)](#)
- [InvalidInputException Structure \(p. 621\)](#)
- [InvalidTaskStatusTransitionException Structure \(p. 621\)](#)
- [JobDefinitionErrorException Structure \(p. 621\)](#)
- [JobRunInTerminalStateException Structure \(p. 621\)](#)
- [JobRunInvalidStateException Structure \(p. 621\)](#)
- [JobRunNotInTerminalStateException Structure \(p. 622\)](#)
- [LateRunnerException Structure \(p. 622\)](#)
- [NoScheduleException Structure \(p. 622\)](#)
- [OperationTimeoutException Structure \(p. 622\)](#)
- [ResourceNumberLimitExceededException Structure \(p. 622\)](#)
- [SchedulerNotRunningException Structure \(p. 623\)](#)
- [SchedulerRunningException Structure \(p. 623\)](#)
- [SchedulerTransitioningException Structure \(p. 623\)](#)
- [UnrecognizedRunnerException Structure \(p. 623\)](#)
- [ValidationException Structure \(p. 623\)](#)
- [VersionMismatchException Structure \(p. 624\)](#)

Security APIs in AWS Glue

The Security API describes the security data types, and the API related to security in AWS Glue.

Data Types

- [DataCatalogEncryptionSettings Structure \(p. 459\)](#)
- [EncryptionAtRest Structure \(p. 460\)](#)
- [ConnectionPasswordEncryption Structure \(p. 460\)](#)
- [EncryptionConfiguration Structure \(p. 461\)](#)
- [S3Encryption Structure \(p. 461\)](#)
- [CloudWatchEncryption Structure \(p. 461\)](#)
- [JobBookmarksEncryption Structure \(p. 461\)](#)
- [SecurityConfiguration Structure \(p. 462\)](#)

DataCatalogEncryptionSettings Structure

Contains configuration information for maintaining Data Catalog security.

Fields

- [EncryptionAtRest – An \[EncryptionAtRest \\(p. 460\\)\]\(#\) object.](#)

Specifies the encryption-at-rest configuration for the Data Catalog.

- **ConnectionPasswordEncryption** – A [ConnectionPasswordEncryption \(p. 460\)](#) object.

When connection password protection is enabled, the Data Catalog uses a customer-provided key to encrypt the password as part of `CreateConnection` or `UpdateConnection` and store it in the `ENCRYPTED_PASSWORD` field in the connection properties. You can enable catalog encryption or only password encryption.

EncryptionAtRest Structure

Specifies the encryption-at-rest configuration for the Data Catalog.

Fields

- **CatalogEncryptionMode** – *Required*: UTF-8 string (valid values: `DISABLED` | `SSE-KMS="SSEKMS"`).
The encryption-at-rest mode for encrypting Data Catalog data.
- **SseAwsKmsKeyId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the AWS KMS key to use for encryption at rest.

ConnectionPasswordEncryption Structure

The data structure used by the Data Catalog to encrypt the password as part of `CreateConnection` or `UpdateConnection` and store it in the `ENCRYPTED_PASSWORD` field in the connection properties. You can enable catalog encryption or only password encryption.

When a `CreateConnection` request arrives containing a password, the Data Catalog first encrypts the password using your AWS KMS key. It then encrypts the whole connection object again if catalog encryption is also enabled.

This encryption requires that you set AWS KMS key permissions to enable or restrict access on the password key according to your security requirements. For example, you might want only administrators to have decrypt permission on the password key.

Fields

- **ReturnConnectionPasswordEncrypted** – *Required*: Boolean.
When the `ReturnConnectionPasswordEncrypted` flag is set to "true", passwords remain encrypted in the responses of `GetConnection` and `GetConnections`. This encryption takes effect independently from catalog encryption.
- **AwsKmsKeyId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
An AWS KMS key that is used to encrypt the connection password.

If connection password protection is enabled, the caller of `CreateConnection` and `UpdateConnection` needs at least `kms:Encrypt` permission on the specified AWS KMS key, to encrypt passwords before storing them in the Data Catalog.

You can set the decrypt permission to enable or restrict access on the password key according to your security requirements.

EncryptionConfiguration Structure

Specifies an encryption configuration.

Fields

- **S3Encryption** – An array of [S3Encryption \(p. 461\)](#) objects.
The encryption configuration for Amazon Simple Storage Service (Amazon S3) data.
- **CloudWatchEncryption** – A [CloudWatchEncryption \(p. 461\)](#) object.
The encryption configuration for Amazon CloudWatch.
- **JobBookmarksEncryption** – A [JobBookmarksEncryption \(p. 461\)](#) object.
The encryption configuration for job bookmarks.

S3Encryption Structure

Specifies how Amazon Simple Storage Service (Amazon S3) data should be encrypted.

Fields

- **S3EncryptionMode** – UTF-8 string (valid values: DISABLED | SSE-KMS="SSEKMS" | SSE-S3="SSES3").
The encryption mode to use for Amazon S3 data.
- **KmsKeyArn** – UTF-8 string, matching the [Custom string pattern #14 \(p. 618\)](#).
The Amazon Resource Name (ARN) of the KMS key to be used to encrypt the data.

CloudWatchEncryption Structure

Specifies how Amazon CloudWatch data should be encrypted.

Fields

- **CloudWatchEncryptionMode** – UTF-8 string (valid values: DISABLED | SSE-KMS="SSEKMS").
The encryption mode to use for CloudWatch data.
- **KmsKeyArn** – UTF-8 string, matching the [Custom string pattern #14 \(p. 618\)](#).
The Amazon Resource Name (ARN) of the KMS key to be used to encrypt the data.

JobBookmarksEncryption Structure

Specifies how job bookmark data should be encrypted.

Fields

- **JobBookmarksEncryptionMode** – UTF-8 string (valid values: DISABLED | CSE-KMS="CSEKMS").
The encryption mode to use for job bookmarks data.
- **KmsKeyArn** – UTF-8 string, matching the [Custom string pattern #14 \(p. 618\)](#).

The Amazon Resource Name (ARN) of the KMS key to be used to encrypt the data.

SecurityConfiguration Structure

Specifies a security configuration.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the security configuration.

- **CreatedTimeStamp** – Timestamp.

The time at which this security configuration was created.

- **EncryptionConfiguration** – An [EncryptionConfiguration \(p. 461\)](#) object.

The encryption configuration associated with this security configuration.

Operations

- [GetDataCatalogEncryptionSettings Action \(Python: get_data_catalog_encryption_settings\) \(p. 462\)](#)
- [PutDataCatalogEncryptionSettings Action \(Python: put_data_catalog_encryption_settings\) \(p. 463\)](#)
- [PutResourcePolicy Action \(Python: put_resource_policy\) \(p. 463\)](#)
- [GetResourcePolicy Action \(Python: get_resource_policy\) \(p. 464\)](#)
- [DeleteResourcePolicy Action \(Python: delete_resource_policy\) \(p. 465\)](#)
- [CreateSecurityConfiguration Action \(Python: create_security_configuration\) \(p. 465\)](#)
- [DeleteSecurityConfiguration Action \(Python: delete_security_configuration\) \(p. 466\)](#)
- [GetSecurityConfiguration Action \(Python: get_security_configuration\) \(p. 466\)](#)
- [GetSecurityConfigurations Action \(Python: get_security_configurations\) \(p. 467\)](#)

GetDataCatalogEncryptionSettings Action (Python: get_data_catalog_encryption_settings)

Retrieves the security configuration for a specified catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog to retrieve the security configuration for. If none is provided, the AWS account ID is used by default.

Response

- **DataCatalogEncryptionSettings** – A [DataCatalogEncryptionSettings \(p. 459\)](#) object.

The requested security configuration.

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutDataCatalogEncryptionSettings Action (Python: `put_data_catalog_encryption_settings`)

Sets the security configuration for a specified catalog. After the configuration has been set, the specified encryption is applied to every catalog write thereafter.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog to set the security configuration for. If none is provided, the AWS account ID is used by default.

- `DataCatalogEncryptionSettings` – *Required:* A [DataCatalogEncryptionSettings \(p. 459\)](#) object.

The security configuration to set.

Response

- *No Response parameters.*

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutResourcePolicy Action (Python: `put_resource_policy`)

Sets the Data Catalog resource policy for access control.

Request

- `PolicyInJson` – *Required:* UTF-8 string, not less than 2 or more than 10240 bytes long.
Contains the policy document to set, in JSON format.
- `PolicyHashCondition` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The hash value returned when the previous policy was set using `PutResourcePolicy`. Its purpose is to prevent concurrent modifications of a policy. Do not use this parameter if no previous policy has been set.

- `PolicyExistsCondition` – UTF-8 string (valid values: `MUST_EXIST` | `NOT_EXIST` | `NONE`).

A value of `MUST_EXIST` is used to update a policy. A value of `NOT_EXIST` is used to create a new policy. If a value of `NONE` or a null value is used, the call will not depend on the existence of a policy.

Response

- `PolicyHash` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A hash of the policy that has just been set. This must be included in a subsequent call that overwrites or updates this policy.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ConditionCheckFailureException`

GetResourcePolicy Action (Python: get_resource_policy)

Retrieves a specified resource policy.

Request

- *No Request parameters.*

Response

- `PolicyInJson` – UTF-8 string, not less than 2 or more than 10240 bytes long.
 - Contains the requested policy document, in JSON format.
- `PolicyHash` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
 - Contains the hash value associated with this policy.
- `CreateTime` – Timestamp.
 - The date and time at which the policy was created.
- `UpdateTime` – Timestamp.
 - The date and time at which the policy was last updated.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

- `InvalidArgumentException`

DeleteResourcePolicy Action (Python: `delete_resource_policy`)

Deletes a specified policy.

Request

- `PolicyHashCondition` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The hash value returned when this policy was set.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`
- `ConditionCheckFailureException`

CreateSecurityConfiguration Action (Python: `create_security_configuration`)

Creates a new security configuration. A security configuration is a set of security properties that can be used by AWS Glue. You can use a security configuration to encrypt data at rest. For information about using security configurations in AWS Glue, see [Encrypting Data Written by Crawlers, Jobs, and Development Endpoints](#).

Request

- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name for the new security configuration.
- `EncryptionConfiguration` – *Required:* An [EncryptionConfiguration \(p. 461\)](#) object.
The encryption configuration for the new security configuration.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name assigned to the new security configuration.
- `CreatedTimestamp` – Timestamp.

The time at which the new security configuration was created.

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteSecurityConfiguration Action (Python: delete_security_configuration)

Deletes a specified security configuration.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the security configuration to delete.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfiguration Action (Python: get_security_configuration)

Retrieves a specified security configuration.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the security configuration to retrieve.

Response

- `SecurityConfiguration` – A [SecurityConfiguration \(p. 462\)](#) object.

The requested security configuration.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfigurations Action (Python: get_security_configurations)

Retrieves a list of all security configurations.

Request

- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum number of results to return.
- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation call.

Response

- `SecurityConfigurations` – An array of [SecurityConfiguration \(p. 462\)](#) objects.
A list of security configurations.
- `NextToken` – UTF-8 string.
A continuation token, if there are more security configurations to return.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

Catalog API

The Catalog API describes the data types and API related to working with catalogs in AWS Glue.

Topics

- [Database API \(p. 468\)](#)
- [Table API \(p. 473\)](#)
- [Partition API \(p. 488\)](#)
- [Connection API \(p. 499\)](#)

- [User-Defined Function API \(p. 506\)](#)
- [Importing an Athena Catalog to AWS Glue \(p. 511\)](#)

Database API

The Database API describes database data types, and includes the API for creating, deleting, locating, updating, and listing databases.

Data Types

- [Database Structure \(p. 468\)](#)
- [DatabaseInput Structure \(p. 468\)](#)
- [PrincipalPermissions Structure \(p. 469\)](#)
- [DataLakePrincipal Structure \(p. 469\)](#)

Database Structure

The `Database` object represents a logical grouping of tables that might reside in a Hive metastore or an RDBMS.

Fields

- **Name** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database. For Hive compatibility, this is folded to lowercase when it is stored.
- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the database.
- **LocationUri** – Uniform resource identifier (uri), not less than 1 or more than 1024 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

The location of the database (for example, an HDFS path).
- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define parameters and properties of the database.
- **CreateTime** – Timestamp.

The time at which the metadata database was created in the catalog.
- **CreateTableDefaultPermissions** – An array of [PrincipalPermissions \(p. 469\)](#) objects.

Creates a set of default permissions on the table for principals.

DatabaseInput Structure

The structure used to create or update a database.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database. For Hive compatibility, this is folded to lowercase when it is stored.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the database.

- **LocationUri** – Uniform resource identifier (uri), not less than 1 or more than 1024 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

The location of the database (for example, an HDFS path).

- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define parameters and properties of the database.

These key-value pairs define parameters and properties of the database.

- **CreateTableDefaultPermissions** – An array of [PrincipalPermissions \(p. 469\)](#) objects.

Creates a set of default permissions on the table for principals.

PrincipalPermissions Structure

Permissions granted to a principal.

Fields

- **Principal** – A [DataLakePrincipal \(p. 469\)](#) object.

The principal who is granted permissions.

- **Permissions** – An array of UTF-8 strings.

The permissions that are granted to the principal.

DataLakePrincipal Structure

The AWS Lake Formation principal.

Fields

- **DataLakePrincipalIdentifier** – UTF-8 string, not less than 1 or more than 255 bytes long.

An identifier for the AWS Lake Formation principal.

Operations

- [CreateDatabase Action \(Python: create_database\) \(p. 470\)](#)

- [UpdateDatabase Action \(Python: update_database\) \(p. 470\)](#)
- [DeleteDatabase Action \(Python: delete_database\) \(p. 471\)](#)
- [GetDatabase Action \(Python: get_database\) \(p. 471\)](#)
- [GetDatabases Action \(Python: get_databases\) \(p. 472\)](#)

CreateDatabase Action (Python: create_database)

Creates a new database in a Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which to create the database. If none is provided, the AWS account ID is used by default.

- **DatabaseInput** – *Required:* A [DatabaseInput \(p. 468\)](#) object.

The metadata for the database.

Response

- *No Response parameters.*

Errors

- [InvalidInputException](#)
- [AlreadyExistsException](#)
- [ResourceNumberLimitExceededException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)

UpdateDatabase Action (Python: update_database)

Updates an existing database definition in a Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the metadata database resides. If none is provided, the AWS account ID is used by default.

- **Name** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database to update in the catalog. For Hive compatibility, this is folded to lowercase.

- **DatabaseInput** – *Required:* A [DatabaseInput \(p. 468\)](#) object.

A [DatabaseInput](#) object specifying the new definition of the metadata database in the catalog.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteDatabase Action (Python: delete_database)

Removes a specified database from a Data Catalog.

Note

After completing this operation, you no longer have access to the tables (and all table versions and partitions that might belong to the tables) and the user-defined functions in the deleted database. AWS Glue deletes these "orphaned" resources asynchronously in a timely manner, at the discretion of the service.

To ensure the immediate deletion of all related resources, before calling `DeleteDatabase`, use `DeleteTableVersion` or `BatchDeleteTableVersion`, `DeletePartition` or `BatchDeletePartition`, `DeleteUserDefinedFunction`, and `DeleteTable` or `BatchDeleteTable`, to delete any resources that belong to the database.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the database resides. If none is provided, the AWS account ID is used by default.

- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database to delete. For Hive compatibility, this must be all lowercase.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetDatabase Action (Python: get_database)

Retrieves the definition of a specified database.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the database resides. If none is provided, the AWS account ID is used by default.

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database to retrieve. For Hive compatibility, this should be all lowercase.

Response

- **Database** – A [Database \(p. 468\)](#) object.

The definition of the specified database in the Data Catalog.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetDatabases Action (Python: `get_databases`)

Retrieves all databases defined in a given Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog from which to retrieve Databases. If none is provided, the AWS account ID is used by default.

- **NextToken** – UTF-8 string.

A continuation token, if this is a continuation call.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of databases to return in one response.

Response

- **DatabaseList** – *Required*: An array of [Database \(p. 468\)](#) objects.

A list of Database objects from the specified catalog.

- **NextToken** – UTF-8 string.

A continuation token for paginating the returned list of tokens, returned if the current segment of the list is not the last.

Errors

- [InvalidInputException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)

Table API

The Table API describes data types and operations associated with tables.

Data Types

- [Table Structure \(p. 473\)](#)
- [TableInput Structure \(p. 474\)](#)
- [Column Structure \(p. 475\)](#)
- [StorageDescriptor Structure \(p. 476\)](#)
- [SerDeInfo Structure \(p. 477\)](#)
- [Order Structure \(p. 477\)](#)
- [SkewedInfo Structure \(p. 478\)](#)
- [TableVersion Structure \(p. 478\)](#)
- [TableError Structure \(p. 478\)](#)
- [TableVersionError Structure \(p. 478\)](#)
- [SortCriterion Structure \(p. 479\)](#)

Table Structure

Represents a collection of related data organized in columns and rows.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The table name. For Hive compatibility, this must be entirely lowercase.
- **DatabaseName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the database where the table metadata resides. For Hive compatibility, this must be all lowercase.
- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
A description of the table.
- **Owner** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The owner of the table.
- **CreateTime** – Timestamp.

The time when the table definition was created in the Data Catalog.

- `UpdateTime` – Timestamp.

The last time that the table was updated.

- `LastAccessTime` – Timestamp.

The last time that the table was accessed. This is usually taken from HDFS, and might not be reliable.

- `LastAnalyzedTime` – Timestamp.

The last time that column statistics were computed for this table.

- `Retention` – Number (integer), not more than None.

The retention time for this table.

- `StorageDescriptor` – A [StorageDescriptor \(p. 476\)](#) object.

A storage descriptor containing information about the physical storage of this table.

- `PartitionKeys` – An array of [Column \(p. 475\)](#) objects.

A list of columns by which the table is partitioned. Only primitive types are supported as partition keys.

When you create a table used by Amazon Athena, and you do not specify any `partitionKeys`, you must at least set the value of `partitionKeys` to an empty list. For example:

`"PartitionKeys": []`

- `ViewOriginalText` – UTF-8 string, not more than 409600 bytes long.

If the table is a view, the original text of the view; otherwise `null`.

- `ViewExpandedText` – UTF-8 string, not more than 409600 bytes long.

If the table is a view, the expanded text of the view; otherwise `null`.

- `TableType` – UTF-8 string, not more than 255 bytes long.

The type of this table (`EXTERNAL_TABLE`, `VIRTUAL_VIEW`, etc.).

- `Parameters` – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define properties associated with the table.

- `CreatedBy` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The person or entity who created the table.

- `IsRegisteredWithLakeFormation` – Boolean.

Indicates whether the table has been registered with AWS Lake Formation.

TableInput Structure

A structure used to define a table.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The table name. For Hive compatibility, this is folded to lowercase when it is stored.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the table.

- **Owner** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The table owner.

- **LastAccessTime** – Timestamp.

The last time that the table was accessed.

- **LastAnalyzedTime** – Timestamp.

The last time that column statistics were computed for this table.

- **Retention** – Number (integer), not more than None.

The retention time for this table.

- **StorageDescriptor** – A [StorageDescriptor \(p. 476\)](#) object.

A storage descriptor containing information about the physical storage of this table.

- **PartitionKeys** – An array of [Column \(p. 475\)](#) objects.

A list of columns by which the table is partitioned. Only primitive types are supported as partition keys.

When you create a table used by Amazon Athena, and you do not specify any `partitionKeys`, you must at least set the value of `partitionKeys` to an empty list. For example:

```
"PartitionKeys": []
```

- **ViewOriginalText** – UTF-8 string, not more than 409600 bytes long.

If the table is a view, the original text of the view; otherwise `null`.

- **ViewExpandedText** – UTF-8 string, not more than 409600 bytes long.

If the table is a view, the expanded text of the view; otherwise `null`.

- **TableType** – UTF-8 string, not more than 255 bytes long.

The type of this table (`EXTERNAL_TABLE`, `VIRTUAL_VIEW`, etc.).

- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define properties associated with the table.

Column Structure

A column in a Table.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the column.

- **Type** – UTF-8 string, not more than 131072 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The data type of the column.

- **Comment** – Comment string, not more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A free-form text comment.

- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define properties associated with the column.

StorageDescriptor Structure

Describes the physical storage of table data.

Fields

- **Columns** – An array of [Column \(p. 475\)](#) objects.

A list of the columns in the table.

- **Location** – Location string, not more than 2056 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

The physical location of the table. By default, this takes the form of the warehouse location, followed by the database location in the warehouse, followed by the table name.

- **InputFormat** – Format string, not more than 128 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The input format: `SequenceFileInputFormat` (binary), or `TextInputFormat`, or a custom format.

- **OutputFormat** – Format string, not more than 128 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The output format: `SequenceFileOutputFormat` (binary), or `IgnoreKeyTextOutputFormat`, or a custom format.

- **Compressed** – Boolean.

`True` if the data in the table is compressed, or `False` if not.

- **NumberOfBuckets** – Number (integer).

Must be specified if the table contains any dimension columns.

- **SerdeInfo** – A [SerDeInfo \(p. 477\)](#) object.

The serialization/deserialization (SerDe) information.

- **BucketColumns** – An array of UTF-8 strings.

A list of reducer grouping columns, clustering columns, and bucketing columns in the table.

- **SortColumns** – An array of [Order \(p. 477\)](#) objects.

A list specifying the sort order of each bucket in the table.

- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

The user-supplied properties in key-value form.

- **SkewedInfo** – A [SkewedInfo \(p. 478\)](#) object.

The information about values that appear frequently in a column (skewed values).

- **StoredAsSubDirectories** – Boolean.

True if the table data is stored in subdirectories, or False if not.

SerDeInfo Structure

Information about a serialization/deserialization program (SerDe) that serves as an extractor and loader.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the SerDe.

- **SerializationLibrary** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Usually the class that implements the SerDe. An example is `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`.

- **Parameters** – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define initialization parameters for the SerDe.

Order Structure

Specifies the sort order of a sorted column.

Fields

- **Column** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the column.

- **SortOrder** – *Required*: Number (integer), not more than 1.

Indicates that the column is sorted in ascending order (== 1), or in descending order (==0).

SkewedInfo Structure

Specifies skewed values in a table. Skewed values are those that occur with very high frequency.

Fields

- **SkewedColumnNames** – An array of UTF-8 strings.
A list of names of columns that contain skewed values.
- **SkewedColumnValues** – An array of UTF-8 strings.
A list of values that appear so frequently as to be considered skewed.
- **SkewedColumnValueLocationMaps** – A map array of key-value pairs.
Each key is a UTF-8 string.
Each value is a UTF-8 string.
A mapping of skewed values to the columns that contain them.

TableVersion Structure

Specifies a version of a table.

Fields

- **Table** – A [Table \(p. 473\)](#) object.
The table in question.
- **VersionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID value that identifies this table version. A VersionId is a string representation of an integer. Each version is incremented by 1.

TableError Structure

An error record for table operations.

Fields

- **TableName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the table. For Hive compatibility, this must be entirely lowercase.
- **ErrorDetail** – An [ErrorDetail \(p. 617\)](#) object.
The details about the error.

TableVersionError Structure

An error record for table-version operations.

Fields

- **TableName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table in question.

- **VersionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID value of the version in question. A **VersionID** is a string representation of an integer. Each version is incremented by 1.

- **ErrorDetail** – An [ErrorDetail \(p. 617\)](#) object.

The details about the error.

SortCriterion Structure

Specifies a field to sort by and a sort order.

Fields

- **FieldName** – Value string, not more than 1024 bytes long.

The name of the field on which to sort.

- **Sort** – UTF-8 string (valid values: `ASC="ASCENDING"` | `DESC="DESCENDING"`).

An ascending or descending sort.

Operations

- [CreateTable Action \(Python: create_table\) \(p. 479\)](#)
- [UpdateTable Action \(Python: update_table\) \(p. 480\)](#)
- [DeleteTable Action \(Python: delete_table\) \(p. 481\)](#)
- [BatchDeleteTable Action \(Python: batch_delete_table\) \(p. 481\)](#)
- [GetTable Action \(Python: get_table\) \(p. 482\)](#)
- [GetTables Action \(Python: get_tables\) \(p. 483\)](#)
- [GetTableVersion Action \(Python: get_table_version\) \(p. 484\)](#)
- [GetTableVersions Action \(Python: get_table_versions\) \(p. 484\)](#)
- [DeleteTableVersion Action \(Python: delete_table_version\) \(p. 485\)](#)
- [BatchDeleteTableVersion Action \(Python: batch_delete_table_version\) \(p. 486\)](#)
- [SearchTables Action \(Python: search_tables\) \(p. 487\)](#)

CreateTable Action (Python: create_table)

Creates a new table definition in the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which to create the Table. If none is supplied, the AWS account ID is used by default.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The catalog database in which to create the new table. For Hive compatibility, this name is entirely lowercase.

- **TableInput** – *Required:* A [TableInput \(p. 474\)](#) object.

The TableInput object that defines the metadata table to create in the catalog.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateTable Action (Python: update_table)

Updates a metadata table in the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the table resides. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the table resides. For Hive compatibility, this name is entirely lowercase.

- **TableInput** – *Required:* A [TableInput \(p. 474\)](#) object.

An updated TableInput object to define the metadata table in the catalog.

- **SkipArchive** – Boolean.

By default, UpdateTable always creates an archived version of the table before updating it. However, if skipArchive is set to true, UpdateTable does not create the archived version.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

DeleteTable Action (Python: delete_table)

Removes a table definition from the Data Catalog.

Note

After completing this operation, you no longer have access to the table versions and partitions that belong to the deleted table. AWS Glue deletes these "orphaned" resources asynchronously in a timely manner, at the discretion of the service.

To ensure the immediate deletion of all related resources, before calling `DeleteTable`, use `DeleteTableVersion` or `BatchDeleteTableVersion`, and `DeletePartition` or `BatchDeletePartition`, to delete any resources that belong to the table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the table resides. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the table resides. For Hive compatibility, this name is entirely lowercase.

- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table to be deleted. For Hive compatibility, this name is entirely lowercase.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchDeleteTable Action (Python: batch_delete_table)

Deletes multiple tables at once.

Note

After completing this operation, you no longer have access to the table versions and partitions that belong to the deleted table. AWS Glue deletes these "orphaned" resources asynchronously in a timely manner, at the discretion of the service.

To ensure the immediate deletion of all related resources, before calling `BatchDeleteTable`, use `DeleteTableVersion` or `BatchDeleteTableVersion`, and `DeletePartition` or `BatchDeletePartition`, to delete any resources that belong to the table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the table resides. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the tables to delete reside. For Hive compatibility, this name is entirely lowercase.

- `TablesToDelete` – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the table to delete.

Response

- `Errors` – An array of [TableError \(p. 478\)](#) objects.

A list of errors encountered in attempting to delete the specified tables.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetTable Action (Python: `get_table`)

Retrieves the `Table` definition in a Data Catalog for a specified table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the table resides. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database in the catalog in which the table resides. For Hive compatibility, this name is entirely lowercase.

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table for which to retrieve the definition. For Hive compatibility, this name is entirely lowercase.

Response

- **Table** – A [Table \(p. 473\)](#) object.

The **Table** object that defines the specified table.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetTables Action (Python: `get_tables`)

Retrieves the definitions of some or all of the tables in a given Database.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the tables reside. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in the catalog whose tables to list. For Hive compatibility, this name is entirely lowercase.

- **Expression** – UTF-8 string, not more than 2048 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A regular expression pattern. If present, only those tables whose names match the pattern are returned.

- **NextToken** – UTF-8 string.

A continuation token, included if this is a continuation call.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of tables to return in a single response.

Response

- **TableList** – An array of [Table \(p. 473\)](#) objects.

A list of the requested **Table** objects.

- **NextToken** – UTF-8 string.

A continuation token, present if the current list segment is not the last.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

GetTableVersion Action (Python: `get_table_version`)

Retrieves a specified version of a table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the tables reside. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in the catalog in which the table resides. For Hive compatibility, this name is entirely lowercase.

- `TableName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table. For Hive compatibility, this name is entirely lowercase.

- `VersionId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID value of the table version to be retrieved. A `VersionID` is a string representation of an integer. Each version is incremented by 1.

Response

- `TableVersion` – A [TableVersion \(p. 478\)](#) object.

The requested table version.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetTableVersions Action (Python: `get_table_versions`)

Retrieves a list of strings that identify available versions of a specified table.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the tables reside. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in the catalog in which the table resides. For Hive compatibility, this name is entirely lowercase.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table. For Hive compatibility, this name is entirely lowercase.

- **NextToken** – UTF-8 string.

A continuation token, if this is not the first call.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of table versions to return in one response.

Response

- **TableVersions** – An array of [TableVersion \(p. 478\)](#) objects.

A list of strings identifying available versions of the specified table.

- **NextToken** – UTF-8 string.

A continuation token, if the list of available versions does not include the last one.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteTableVersion Action (Python: `delete_table_version`)

Deletes a specified version of a table.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the tables reside. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in the catalog in which the table resides. For Hive compatibility, this name is entirely lowercase.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table. For Hive compatibility, this name is entirely lowercase.

- **VersionId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the table version to be deleted. A **VersionID** is a string representation of an integer. Each version is incremented by 1.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchDeleteTableVersion Action (Python: `batch_delete_table_version`)

Deletes a specified batch of versions of a table.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the tables reside. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in the catalog in which the table resides. For Hive compatibility, this name is entirely lowercase.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table. For Hive compatibility, this name is entirely lowercase.

- **VersionIds** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the IDs of versions to be deleted. A **VersionID** is a string representation of an integer. Each version is incremented by 1.

Response

- **Errors** – An array of [TableVersionError \(p. 478\)](#) objects.

A list of errors encountered while trying to delete the specified table versions.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

SearchTables Action (Python: search_tables)

Searches a set of tables based on properties in the table metadata as well as on the parent database. You can search against text or filter conditions.

You can only get tables that you have access to based on the security policies defined in Lake Formation. You need at least a read-only access to the table for it to be returned. If you do not have access to all the columns in the table, these columns will not be searched against when returning the list of tables back to you. If you have access to the columns but not the data in the columns, those columns and the associated metadata for those columns will be included in the search.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A unique identifier, consisting of `account_id/datalake`.

- `NextToken` – UTF-8 string.

A continuation token, included if this is a continuation call.

- `Filters` – An array of [PropertyPredicate \(p. 617\)](#) objects.

A list of key-value pairs, and a comparator used to filter the search results. Returns all entities matching the predicate.

- `SearchText` – Value string, not more than 1024 bytes long.

A string used for a text search.

Specifying a value in quotes filters based on an exact match to the value.

- `SortCriteria` – An array of [SortCriterion \(p. 479\)](#) objects, not more than 1 structures.

A list of criteria for sorting the results by a field name, in an ascending or descending order.

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum number of tables to return in a single response.

Response

- `NextToken` – UTF-8 string.

A continuation token, present if the current list segment is not the last.

- `TableList` – An array of [Table \(p. 473\)](#) objects.

A list of the requested Table objects. The SearchTables response returns only the tables that you have access to.

Errors

- [InternalServiceException](#)
- [InvalidInputException](#)
- [OperationTimeoutException](#)

Partition API

The Partition API describes data types and operations used to work with partitions.

Data Types

- [Partition Structure \(p. 488\)](#)
- [PartitionInput Structure \(p. 489\)](#)
- [PartitionSpecWithSharedStorageDescriptor Structure \(p. 489\)](#)
- [PartitionListComposingSpec Structure \(p. 489\)](#)
- [PartitionSpecProxy Structure \(p. 490\)](#)
- [PartitionValueList Structure \(p. 490\)](#)
- [Segment Structure \(p. 490\)](#)
- [PartitionError Structure \(p. 491\)](#)

Partition Structure

Represents a slice of table data.

Fields

- **Values** – An array of UTF-8 strings.
The values of the partition.
- **DatabaseName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the catalog database in which to create the partition.
- **TableName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the database table in which to create the partition.
- **CreationTime** – Timestamp.
The time at which the partition was created.
- **LastAccessTime** – Timestamp.
The last time at which the partition was accessed.
- **StorageDescriptor** – A [StorageDescriptor \(p. 476\)](#) object.
Provides information about the physical location where the partition is stored.
- **Parameters** – A map array of key-value pairs.
Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define partition parameters.

- `LastAnalyzedTime` – Timestamp.

The last time at which column statistics were computed for this partition.

PartitionInput Structure

The structure used to create and update a partition.

Fields

- `Values` – An array of UTF-8 strings.

The values of the partition. Although this parameter is not required by the SDK, you must specify this parameter for a valid input.

The values for the keys for the new partition must be passed as an array of String objects that must be ordered in the same order as the partition keys appearing in the Amazon S3 prefix. Otherwise AWS Glue will add the values to the wrong keys.

- `LastAccessTime` – Timestamp.

The last time at which the partition was accessed.

- `StorageDescriptor` – A [StorageDescriptor \(p. 476\)](#) object.

Provides information about the physical location where the partition is stored.

- `Parameters` – A map array of key-value pairs.

Each key is a Key string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define partition parameters.

- `LastAnalyzedTime` – Timestamp.

The last time at which column statistics were computed for this partition.

PartitionSpecWithSharedStorageDescriptor Structure

A partition specification for partitions that share a physical location.

Fields

- `StorageDescriptor` – A [StorageDescriptor \(p. 476\)](#) object.

The shared physical storage information.

- `Partitions` – An array of [Partition \(p. 488\)](#) objects.

A list of the partitions that share this physical location.

PartitionListComposingSpec Structure

Lists the related partitions.

Fields

- **Partitions** – An array of [Partition \(p. 488\)](#) objects.
A list of the partitions in the composing specification.

PartitionSpecProxy Structure

Provides a root path to specified partitions.

Fields

- **DatabaseName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The catalog database in which the partitions reside.
- **TableName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the table that contains the partitions.
- **RootPath** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The root path of the proxy for addressing the partitions.
- **PartitionSpecWithSharedSD** – A [PartitionSpecWithSharedStorageDescriptor \(p. 489\)](#) object.
A specification of partitions that share the same physical storage location.
- **PartitionListComposingSpec** – A [PartitionListComposingSpec \(p. 489\)](#) object.
Specifies a list of partitions.

PartitionValueList Structure

Contains a list of values defining partitions.

Fields

- **Values** – *Required:* An array of UTF-8 strings.
The list of values.

Segment Structure

Defines a non-overlapping region of a table's partitions, allowing multiple requests to be executed in parallel.

Fields

- **SegmentNumber** – *Required:* Number (integer), not more than None.
The zero-based index number of the segment. For example, if the total number of segments is 4, `SegmentNumber` values range from 0 through 3.
- **TotalSegments** – *Required:* Number (integer), not less than 1 or more than 10.

The total number of segments.

PartitionError Structure

Contains information about a partition error.

Fields

- **PartitionValues** – An array of UTF-8 strings.

The values that define the partition.

- **ErrorDetail** – An [ErrorDetail \(p. 617\)](#) object.

The details about the partition error.

Operations

- [CreatePartition Action \(Python: create_partition\) \(p. 491\)](#)
- [BatchCreatePartition Action \(Python: batch_create_partition\) \(p. 492\)](#)
- [UpdatePartition Action \(Python: update_partition\) \(p. 493\)](#)
- [DeletePartition Action \(Python: delete_partition\) \(p. 493\)](#)
- [BatchDeletePartition Action \(Python: batch_delete_partition\) \(p. 494\)](#)
- [GetPartition Action \(Python: get_partition\) \(p. 495\)](#)
- [GetPartitions Action \(Python: get_partitions\) \(p. 495\)](#)
- [BatchGetPartition Action \(Python: batch_get_partition\) \(p. 498\)](#)

CreatePartition Action (Python: create_partition)

Creates a new partition.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The AWS account ID of the catalog in which the partition is to be created.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the metadata database in which the partition is to be created.

- **TableName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the metadata table in which the partition is to be created.

- **PartitionInput** – *Required:* A [PartitionInput \(p. 489\)](#) object.

A PartitionInput structure defining the partition to be created.

Response

- *No Response parameters.*

Errors

- `InvalidArgumentException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

BatchCreatePartition Action (Python: `batch_create_partition`)

Creates one or more partitions in a batch operation.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the catalog in which the partition is to be created. Currently, this should be the AWS account ID.

- `DatabaseName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the metadata database in which the partition is to be created.

- `TableName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the metadata table in which the partition is to be created.

- `PartitionInputList` – *Required:* An array of [PartitionInput \(p. 489\)](#) objects, not more than 100 structures.

A list of `PartitionInput` structures that define the partitions to be created.

Response

- `Errors` – An array of [PartitionError \(p. 491\)](#) objects.

The errors encountered when trying to create the requested partitions.

Errors

- `InvalidArgumentException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdatePartition Action (Python: update_partition)

Updates a partition.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the partition to be updated resides. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the table in question resides.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table in which the partition to be updated is located.

- **PartitionValueList** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the values defining the partition.

- **PartitionInput** – *Required*: A [PartitionInput \(p. 489\)](#) object.

The new partition object to update the partition to.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartition Action (Python: delete_partition)

Deletes a specified partition.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the partition to be deleted resides. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the table in question resides.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table that contains the partition to be deleted.

- **PartitionValues** – *Required*: An array of UTF-8 strings.

The values that define the partition.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchDeletePartition Action (Python: batch_delete_partition)

Deletes one or more partitions in a batch operation.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the partition to be deleted resides. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which the table in question resides.

- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table that contains the partitions to be deleted.

- **PartitionsToDelete** – *Required*: An array of [PartitionValueList \(p. 490\)](#) objects, not more than 25 structures.

A list of `PartitionInput` structures that define the partitions to be deleted.

Response

- **Errors** – An array of [PartitionError \(p. 491\)](#) objects.

The errors encountered when trying to delete the requested partitions.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`

- `InternalServiceException`
- `OperationTimeoutException`

GetPartition Action (Python: get_partition)

Retrieves information about a specified partition.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the partition in question resides. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the partition resides.

- `TableName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the partition's table.

- `PartitionValues` – *Required*: An array of UTF-8 strings.

The values that define the partition.

Response

- `Partition` – A [Partition \(p. 488\)](#) object.

The requested information, in the form of a `Partition` object.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetPartitions Action (Python: get_partitions)

Retrieves information about the partitions in a table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the partitions in question reside. If none is provided, the AWS account ID is used by default.

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the partitions reside.

- **TableName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the partitions' table.

- **Expression** – Predicate string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

An expression that filters the partitions to be returned.

The expression uses SQL syntax similar to the SQL `WHERE` filter clause. The SQL statement parser [JSQLParser](#) parses the expression.

Operators: The following are the operators that you can use in the Expression API call:

=

Checks whether the values of the two operands are equal; if yes, then the condition becomes true.

Example: Assume 'variable a' holds 10 and 'variable b' holds 20.

$(a = b)$ is not true.

< >

Checks whether the values of two operands are equal; if the values are not equal, then the condition becomes true.

Example: $(a < > b)$ is true.

>

Checks whether the value of the left operand is greater than the value of the right operand; if yes, then the condition becomes true.

Example: $(a > b)$ is not true.

<

Checks whether the value of the left operand is less than the value of the right operand; if yes, then the condition becomes true.

Example: $(a < b)$ is true.

>=

Checks whether the value of the left operand is greater than or equal to the value of the right operand; if yes, then the condition becomes true.

Example: $(a >= b)$ is not true.

<=

Checks whether the value of the left operand is less than or equal to the value of the right operand; if yes, then the condition becomes true.

Example: $(a <= b)$ is true.

AND, OR, IN, BETWEEN, LIKE, NOT, IS NULL

Logical operators.

Supported Partition Key Types: The following are the supported partition keys.

- **string**

- date
- timestamp
- int
- bigint
- long
- tinyint
- smallint
- decimal

If an invalid type is encountered, an exception is thrown.

The following list shows the valid operators on each type. When you define a crawler, the `partitionKey` type is created as a `STRING`, to be compatible with the catalog partitions.

Sample API Call:

Example

The table `twitter_partition` has three partitions:

```
year = 2015
      year = 2016
      year = 2017
```

Example

Get partition `year` equal to 2015

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year=='2015'"
```

Example

Get partition `year` between 2016 and 2018 (exclusive)

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year>'2016' AND year<'2018'"
```

Example

Get partition `year` between 2015 and 2018 (inclusive). The following API calls are equivalent to each other:

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year>='2015' AND year<='2018'"

aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year BETWEEN 2015 AND 2018"

aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year IN (2015,2016,2017,2018)"
```

Example

A wildcard partition filter, where the following call output is partition year=2017. A regular expression is not supported in LIKE.

```
aws glue get-partitions --database-name dbname --table-name twitter_partition  
--expression "year LIKE '%7'"
```

- **NextToken** – UTF-8 string.
A continuation token, if this is not the first call to retrieve these partitions.
- **Segment** – A [Segment \(p. 490\)](#) object.
The segment of the table's partitions to scan in this request.
- **MaxResults** – Number (integer), not less than 1 or more than 1000.
The maximum number of partitions to return in a single response.

Response

- **Partitions** – An array of [Partition \(p. 488\)](#) objects.
A list of requested partitions.
- **NextToken** – UTF-8 string.
A continuation token, if the returned list of partitions does not include the last one.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

BatchGetPartition Action (Python: batch_get_partition)

Retrieves partitions in a batch request.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the Data Catalog where the partitions in question reside. If none is supplied, the AWS account ID is used by default.
- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the catalog database where the partitions reside.
- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the partitions' table.

- **PartitionsToGet** – *Required:* An array of [PartitionValueList \(p. 490\)](#) objects, not more than 1000 structures.

A list of partition values identifying the partitions to retrieve.

Response

- **Partitions** – An array of [Partition \(p. 488\)](#) objects.

A list of the requested partitions.

- **UnprocessedKeys** – An array of [PartitionValueList \(p. 490\)](#) objects, not more than 1000 structures.

A list of the partition values in the request for which partitions were not returned.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

Connection API

The Connection API describes AWS Glue connection data types, and the API for creating, deleting, updating, and listing connections.

Data Types

- [Connection Structure \(p. 499\)](#)
- [ConnectionInput Structure \(p. 501\)](#)
- [PhysicalConnectionRequirements Structure \(p. 501\)](#)
- [GetConnectionsFilter Structure \(p. 502\)](#)

Connection Structure

Defines a connection to a data source.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the connection definition.
- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
The description of the connection.
- **ConnectionType** – UTF-8 string (valid values: `JDBC` | `SFTP`).

The type of the connection. Currently, only JDBC is supported; SFTP is not supported.

- **MatchCriteria** – An array of UTF-8 strings, not more than 10 strings.

A list of criteria that can be used in selecting this connection.

- **ConnectionProperties** – A map array of key-value pairs, not more than 100 pairs.

Each key is a UTF-8 string (valid values: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING).

Each value is a Value string, not more than 1024 bytes long.

These key-value pairs define parameters for the connection:

- **HOST** - The host URI: either the fully qualified domain name (FQDN) or the IPv4 address of the database host.
 - **PORT** - The port number, between 1024 and 65535, of the port on which the database host is listening for database connections.
 - **USER_NAME** - The name under which to log in to the database. The value string for **USER_NAME** is "USERNAME".
 - **PASSWORD** - A password, if one is used, for the user name.
 - **ENCRYPTED_PASSWORD** - When you enable connection password protection by setting ConnectionPasswordEncryption in the Data Catalog encryption settings, this field stores the encrypted password.
 - **JDBC_DRIVER_JAR_URI** - The Amazon Simple Storage Service (Amazon S3) path of the JAR file that contains the JDBC driver to use.
 - **JDBC_DRIVER_CLASS_NAME** - The class name of the JDBC driver to use.
 - **JDBC_ENGINE** - The name of the JDBC engine to use.
 - **JDBC_ENGINE_VERSION** - The version of the JDBC engine to use.
 - **CONFIG_FILES** - (Reserved for future use.)
 - **INSTANCE_ID** - The instance ID to use.
 - **JDBC_CONNECTION_URL** - The URL for the JDBC connection.
 - **JDBC_ENFORCE_SSL** - A Boolean string (true, false) specifying whether Secure Sockets Layer (SSL) with hostname matching is enforced for the JDBC connection on the client. The default is false.
 - **CUSTOM_JDBC_CERT** - An Amazon S3 location specifying the customer's root certificate. AWS Glue uses this root certificate to validate the customer's certificate when connecting to the customer database. AWS Glue only handles X.509 certificates. The certificate provided must be DER-encoded and supplied in Base64 encoding PEM format.
 - **SKIP_CUSTOM_JDBC_CERT_VALIDATION** - By default, this is `false`. AWS Glue validates the Signature algorithm and Subject Public Key Algorithm for the customer certificate. The only permitted algorithms for the Signature algorithm are SHA256withRSA, SHA384withRSA or SHA512withRSA. For the Subject Public Key Algorithm, the key length must be at least 2048. You can set the value of this property to `true` to skip AWS Glue's validation of the customer certificate.
 - **CUSTOM_JDBC_CERT_STRING** - A custom JDBC certificate string which is used for domain match or distinguished name match to prevent a man-in-the-middle attack. In Oracle database, this is used as the `SSL_SERVER_CERT_DN`; in Microsoft SQL Server, this is used as the `hostNameInCertificate`.
 - **PhysicalConnectionRequirements** – A [PhysicalConnectionRequirements \(p. 501\)](#) object.
- A map of physical connection requirements, such as virtual private cloud (VPC) and `SecurityGroup`, that are needed to make this connection successfully.
- **CreationTime** – Timestamp.

The time that this connection definition was created.

- `LastUpdatedTime` – Timestamp.

The last time that this connection definition was updated.

- `LastUpdatedBy` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The user, group, or role that last updated this connection definition.

ConnectionInput Structure

A structure that is used to specify a connection to create or update.

Fields

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the connection.

- `Description` – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

The description of the connection.

- `ConnectionType` – *Required*: UTF-8 string (valid values: JDBC | SFTP).

The type of the connection. Currently, only JDBC is supported; SFTP is not supported.

- `MatchCriteria` – An array of UTF-8 strings, not more than 10 strings.

A list of criteria that can be used in selecting this connection.

- `ConnectionProperties` – *Required*: A map array of key-value pairs, not more than 100 pairs.

Each key is a UTF-8 string (valid values: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING).

Each value is a Value string, not more than 1024 bytes long.

These key-value pairs define parameters for the connection.

- `PhysicalConnectionRequirements` – A [PhysicalConnectionRequirements \(p. 501\)](#) object.

A map of physical connection requirements, such as virtual private cloud (VPC) and `SecurityGroup`, that are needed to successfully make this connection.

PhysicalConnectionRequirements Structure

Specifies the physical requirements for a connection.

Fields

- `SubnetId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The subnet ID used by the connection.

- **SecurityGroupIdList** – An array of UTF-8 strings, not more than 50 strings.

The security group ID list used by the connection.

- **AvailabilityZone** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The connection's Availability Zone. This field is redundant because the specified subnet implies the Availability Zone to be used. Currently the field must be populated, but it will be deprecated in the future.

GetConnectionsFilter Structure

Filters the connection definitions that are returned by the `GetConnections` API operation.

Fields

- **MatchCriteria** – An array of UTF-8 strings, not more than 10 strings.

A criteria string that must match the criteria recorded in the connection definition for that connection definition to be returned.

- **ConnectionType** – UTF-8 string (valid values: `JDBC` | `SFTP`).

The type of connections to return. Currently, only `JDBC` is supported; `SFTP` is not supported.

Operations

- [CreateConnection Action \(Python: create_connection\) \(p. 502\)](#)
- [DeleteConnection Action \(Python: delete_connection\) \(p. 503\)](#)
- [GetConnection Action \(Python: get_connection\) \(p. 503\)](#)
- [GetConnections Action \(Python: get_connections\) \(p. 504\)](#)
- [UpdateConnection Action \(Python: update_connection\) \(p. 505\)](#)
- [BatchDeleteConnection Action \(Python: batch_delete_connection\) \(p. 505\)](#)

CreateConnection Action (Python: `create_connection`)

Creates a connection definition in the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which to create the connection. If none is provided, the AWS account ID is used by default.

- **ConnectionInput** – *Required:* A [ConnectionInput \(p. 501\)](#) object.

A `ConnectionInput` object defining the connection to create.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

DeleteConnection Action (Python: `delete_connection`)

Deletes a connection from the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the Data Catalog in which the connection resides. If none is provided, the AWS account ID is used by default.
- `ConnectionName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the connection to delete.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`

GetConnection Action (Python: `get_connection`)

Retrieves a connection definition from the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the Data Catalog in which the connection resides. If none is provided, the AWS account ID is used by default.
- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the connection definition to retrieve.

- `HidePassword` – Boolean.

Allows you to retrieve the connection metadata without returning the password. For instance, the AWS Glue console uses this flag to retrieve the connection, and does not display the password. Set

this parameter when the caller might not have permission to use the AWS KMS key to decrypt the password, but it does have permission to access the rest of the connection properties.

Response

- **Connection** – A [Connection \(p. 499\)](#) object.

The requested connection definition.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

GetConnections Action (Python: `get_connections`)

Retrieves a list of connection definitions from the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the connections reside. If none is provided, the AWS account ID is used by default.

- **Filter** – A [GetConnectionsFilter \(p. 502\)](#) object.

A filter that controls which connections are returned.

- **HidePassword** – Boolean.

Allows you to retrieve the connection metadata without returning the password. For instance, the AWS Glue console uses this flag to retrieve the connection, and does not display the password. Set this parameter when the caller might not have permission to use the AWS KMS key to decrypt the password, but it does have permission to access the rest of the connection properties.

- **NextToken** – UTF-8 string.

A continuation token, if this is a continuation call.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of connections to return in one response.

Response

- **ConnectionList** – An array of [Connection \(p. 499\)](#) objects.

A list of requested connection definitions.

- **NextToken** – UTF-8 string.

A continuation token, if the list of connections returned does not include the last of the filtered connections.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

UpdateConnection Action (Python: `update_connection`)

Updates a connection definition in the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the connection resides. If none is provided, the AWS account ID is used by default.

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the connection definition to update.

- `ConnectionInput` – *Required*: A [ConnectionInput \(p. 501\)](#) object.

A ConnectionInput object that redefines the connection in question.

Response

- *No Response parameters.*

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

BatchDeleteConnection Action (Python: `batch_delete_connection`)

Deletes a list of connection definitions from the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which the connections reside. If none is provided, the AWS account ID is used by default.

- **ConnectionNameList** – *Required*: An array of UTF-8 strings, not more than 25 strings.

A list of names of the connections to delete.

Response

- **Succeeded** – An array of UTF-8 strings.

A list of names of the connection definitions that were successfully deleted.

- **Errors** – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a [ErrorDetail \(p. 617\)](#) object.

A map of the names of connections that were not successfully deleted to error details.

Errors

- `InternalServiceException`
- `OperationTimeoutException`

User-Defined Function API

The User-Defined Function API describes AWS Glue data types and operations used in working with functions.

Data Types

- [UserDefinedFunction Structure \(p. 506\)](#)
- [UserDefinedFunctionInput Structure \(p. 507\)](#)

UserDefinedFunction Structure

Represents the equivalent of a Hive user-defined function (UDF) definition.

Fields

- **FunctionName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the function.

- **ClassName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The Java class that contains the function code.

- **OwnerName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The owner of the function.

- **OwnerType** – UTF-8 string (valid values: `USER` | `ROLE` | `GROUP`).

The owner type.

- **CreateTime** – Timestamp.

The time at which the function was created.

- **ResourceUrIs** – An array of [ResourceUri \(p. 617\)](#) objects, not more than 1000 structures.

The resource URIs for the function.

UserDefinedFunctionInput Structure

A structure used to create or update a user-defined function.

Fields

- **FunctionName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the function.

- **ClassName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The Java class that contains the function code.

- **OwnerName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The owner of the function.

- **OwnerType** – UTF-8 string (valid values: `USER` | `ROLE` | `GROUP`).

The owner type.

- **ResourceUrIs** – An array of [ResourceUri \(p. 617\)](#) objects, not more than 1000 structures.

The resource URIs for the function.

Operations

- [CreateUserDefinedFunction Action \(Python: create_user_defined_function\) \(p. 507\)](#)
- [UpdateUserDefinedFunction Action \(Python: update_user_defined_function\) \(p. 508\)](#)
- [DeleteUserDefinedFunction Action \(Python: delete_user_defined_function\) \(p. 509\)](#)
- [GetUserDefinedFunction Action \(Python: get_user_defined_function\) \(p. 509\)](#)
- [GetUserDefinedFunctions Action \(Python: get_user_defined_functions\) \(p. 510\)](#)

CreateUserDefinedFunction Action (Python: create_user_defined_function)

Creates a new function definition in the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog in which to create the function. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database in which to create the function.

- **FunctionInput** – *Required:* An [UserDefinedFunctionInput \(p. 507\)](#) object.

A FunctionInput object that defines the function to create in the Data Catalog.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

[UpdateUserDefinedFunction Action \(Python: update_user_defined_function\)](#)

Updates an existing function definition in the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the function to be updated is located. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the function to be updated is located.

- **FunctionName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the function.

- **FunctionInput** – *Required:* An [UserDefinedFunctionInput \(p. 507\)](#) object.

A FunctionInput object that redefines the function in the Data Catalog.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteUserDefinedFunction Action (Python: `delete_user_defined_function`)

Deletes an existing function definition from the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the function to be deleted is located. If none is supplied, the AWS account ID is used by default.

- `DatabaseName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the function is located.

- `FunctionName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the function definition to be deleted.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetUserDefinedFunction Action (Python: `get_user_defined_function`)

Retrieves a specified function definition from the Data Catalog.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the function to be retrieved is located. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the function is located.

- **FunctionName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the function.

Response

- **UserDefinedFunction** – An [UserDefinedFunction \(p. 506\)](#) object.

The requested function definition.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetUserDefinedFunctions Action (Python: `get_user_defined_functions`)

Retrieves multiple function definitions from the Data Catalog.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the Data Catalog where the functions to be retrieved are located. If none is provided, the AWS account ID is used by default.

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the catalog database where the functions are located.

- **Pattern** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

An optional function-name pattern string that filters the function definitions returned.

- **NextToken** – UTF-8 string.

A continuation token, if this is a continuation call.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of functions to return in one response.

Response

- **UserDefinedFunctions** – An array of [UserDefinedFunction \(p. 506\)](#) objects.

A list of requested function definitions.

- `NextToken` – UTF-8 string.

A continuation token, if the list of functions returned does not include the last requested function.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

Importing an Athena Catalog to AWS Glue

The Migration API describes AWS Glue data types and operations having to do with migrating an Athena Data Catalog to AWS Glue.

Data Types

- [CatalogImportStatus Structure \(p. 511\)](#)

CatalogImportStatus Structure

A structure containing migration status information.

Fields

- `ImportCompleted` – Boolean.

`True` if the migration has completed, or `False` otherwise.
- `ImportTime` – Timestamp.

The time that the migration was started.
- `ImportedBy` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the person who initiated the migration.

Operations

- [ImportCatalogToGlue Action \(Python: import_catalog_to_glue\) \(p. 511\)](#)
- [GetCatalogImportStatus Action \(Python: get_catalog_import_status\) \(p. 512\)](#)

ImportCatalogToGlue Action (Python: import_catalog_to_glue)

Imports an existing Amazon Athena Data Catalog to AWS Glue

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the catalog to import. Currently, this should be the AWS account ID.

Response

- *No Response parameters.*

Errors

- `InternalServiceException`
- `OperationTimeoutException`

[GetCatalogImportStatus Action \(Python: get_catalog_import_status\)](#)

Retrieves the status of a migration operation.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the catalog to migrate. Currently, this should be the AWS account ID.

Response

- **ImportStatus** – A [CatalogImportStatus \(p. 511\)](#) object.

The status of the specified catalog migration.

Errors

- `InternalServiceException`
- `OperationTimeoutException`

Crawlers and Classifiers API

The Crawler and Classifiers API describes the AWS Glue crawler and classifier data types, and includes the API for creating, deleting, updating, and listing crawlers or classifiers.

Topics

- [Classifier API \(p. 513\)](#)
- [Crawler API \(p. 522\)](#)
- [Crawler Scheduler API \(p. 533\)](#)

Classifier API

The Classifier API describes AWS Glue classifier data types, and includes the API for creating, deleting, updating, and listing classifiers.

Data Types

- [Classifier Structure \(p. 513\)](#)
- [GrokClassifier Structure \(p. 513\)](#)
- [XMLClassifier Structure \(p. 514\)](#)
- [JsonClassifier Structure \(p. 515\)](#)
- [CsvClassifier Structure \(p. 515\)](#)
- [CreateGrokClassifierRequest Structure \(p. 516\)](#)
- [UpdateGrokClassifierRequest Structure \(p. 516\)](#)
- [CreateXMLClassifierRequest Structure \(p. 517\)](#)
- [UpdateXMLClassifierRequest Structure \(p. 517\)](#)
- [CreateJsonClassifierRequest Structure \(p. 517\)](#)
- [UpdateJsonClassifierRequest Structure \(p. 518\)](#)
- [CreateCsvClassifierRequest Structure \(p. 518\)](#)
- [UpdateCsvClassifierRequest Structure \(p. 518\)](#)

Classifier Structure

Classifiers are triggered during a crawl task. A classifier checks whether a given file is in a format it can handle. If it is, the classifier creates a schema in the form of a `StructType` object that matches that data format.

You can use the standard classifiers that AWS Glue provides, or you can write your own classifiers to best categorize your data sources and specify the appropriate schemas to use for them. A classifier can be a `grok` classifier, an `XML` classifier, a `JSON` classifier, or a custom `csv` classifier, as specified in one of the fields in the `Classifier` object.

Fields

- `GrokClassifier` – A [GrokClassifier \(p. 513\)](#) object.
A classifier that uses `grok`.
- `XMLClassifier` – A [XMLClassifier \(p. 514\)](#) object.
A classifier for XML content.
- `JsonClassifier` – A [JsonClassifier \(p. 515\)](#) object.
A classifier for JSON content.
- `CsvClassifier` – A [CsvClassifier \(p. 515\)](#) object.
A classifier for comma-separated values (CSV).

GrokClassifier Structure

A classifier that uses `grok` patterns.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the classifier.
- **Classification** – *Required*: UTF-8 string.
An identifier of the data format that the classifier matches, such as Twitter, JSON, Omniture logs, and so on.
- **CreationTime** – Timestamp.
The time that this classifier was registered.
- **LastUpdated** – Timestamp.
The time that this classifier was last updated.
- **Version** – Number (long).
The version of this classifier.
- **GrokPattern** – *Required*: UTF-8 string, not less than 1 or more than 2048 bytes long, matching the [A Logstash Grok string pattern \(p. 618\)](#).
The grok pattern applied to a data store by this classifier. For more information, see built-in patterns in [Writing Custom Classifiers](#).
- **CustomPatterns** – UTF-8 string, not more than 16000 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
Optional custom grok patterns defined by this classifier. For more information, see custom patterns in [Writing Custom Classifiers](#).

XMLClassifier Structure

A classifier for XML content.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the classifier.
- **Classification** – *Required*: UTF-8 string.
An identifier of the data format that the classifier matches.
- **CreationTime** – Timestamp.
The time that this classifier was registered.
- **LastUpdated** – Timestamp.
The time that this classifier was last updated.
- **Version** – Number (long).
The version of this classifier.
- **RowTag** – UTF-8 string.
The XML tag designating the element that contains each record in an XML document being parsed. This can't identify a self-closing element (closed by />). An empty row element that contains only

attributes can be parsed as long as it ends with a closing tag (for example, <row item_a="A" item_b="B"></row> is okay, but <row item_a="A" item_b="B" /> is not).

JsonClassifier Structure

A classifier for JSON content.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- **CreationTime** – Timestamp.

The time that this classifier was registered.

- **LastUpdated** – Timestamp.

The time that this classifier was last updated.

- **Version** – Number (long).

The version of this classifier.

- **JsonPath** – *Required*: UTF-8 string.

A JsonPath string defining the JSON data for the classifier to classify. AWS Glue supports a subset of JsonPath, as described in [Writing JsonPath Custom Classifiers](#).

CsvClassifier Structure

A classifier for custom CSV content.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- **CreationTime** – Timestamp.

The time that this classifier was registered.

- **LastUpdated** – Timestamp.

The time that this classifier was last updated.

- **Version** – Number (long).

The version of this classifier.

- **Delimiter** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what separates each column entry in the row.

- **QuoteSymbol** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what combines content into a single column value. It must be different from the column delimiter.

- **ContainsHeader** – UTF-8 string (valid values: UNKNOWN | PRESENT | ABSENT).
Indicates whether the CSV file contains a header.
- **Header** – An array of UTF-8 strings.
A list of strings representing column names.
- **DisableValueTrimming** – Boolean.
Specifies not to trim values before identifying the type of column values. The default value is true.
- **AllowSingleColumn** – Boolean.
Enables the processing of files that contain only one column.

CreateGrokClassifierRequest Structure

Specifies a grok classifier for `CreateClassifier` to create.

Fields

- **Classification** – *Required*: UTF-8 string.
An identifier of the data format that the classifier matches, such as Twitter, JSON, Omniture logs, Amazon CloudWatch Logs, and so on.
- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the new classifier.
- **GrokPattern** – *Required*: UTF-8 string, not less than 1 or more than 2048 bytes long, matching the [A Logstash Grok string pattern \(p. 618\)](#).
The grok pattern used by this classifier.
- **CustomPatterns** – UTF-8 string, not more than 16000 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
Optional custom grok patterns used by this classifier.

UpdateGrokClassifierRequest Structure

Specifies a grok classifier to update when passed to `UpdateClassifier`.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the GrokClassifier.
- **Classification** – UTF-8 string.
An identifier of the data format that the classifier matches, such as Twitter, JSON, Omniture logs, Amazon CloudWatch Logs, and so on.
- **GrokPattern** – UTF-8 string, not less than 1 or more than 2048 bytes long, matching the [A Logstash Grok string pattern \(p. 618\)](#).
The grok pattern used by this classifier.

- `CustomPatterns` – UTF-8 string, not more than 16000 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

Optional custom grok patterns used by this classifier.

CreateXMLClassifierRequest Structure

Specifies an XML classifier for `CreateClassifier` to create.

Fields

- `Classification` – *Required*: UTF-8 string.
An identifier of the data format that the classifier matches.
- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- `RowTag` – UTF-8 string.

The XML tag designating the element that contains each record in an XML document being parsed. This can't identify a self-closing element (closed by `/>`). An empty row element that contains only attributes can be parsed as long as it ends with a closing tag (for example, `<row item_a="A" item_b="B"></row>` is okay, but `<row item_a="A" item_b="B" />` is not).

UpdateXMLClassifierRequest Structure

Specifies an XML classifier to be updated.

Fields

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the classifier.
- `Classification` – UTF-8 string.
An identifier of the data format that the classifier matches.
- `RowTag` – UTF-8 string.

The XML tag designating the element that contains each record in an XML document being parsed. This cannot identify a self-closing element (closed by `/>`). An empty row element that contains only attributes can be parsed as long as it ends with a closing tag (for example, `<row item_a="A" item_b="B"></row>` is okay, but `<row item_a="A" item_b="B" />` is not).

CreateJsonClassifierRequest Structure

Specifies a JSON classifier for `CreateClassifier` to create.

Fields

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- **JsonPath** – *Required*: UTF-8 string.

A JsonPath string defining the JSON data for the classifier to classify. AWS Glue supports a subset of JsonPath, as described in [Writing JsonPath Custom Classifiers](#).

UpdateJsonClassifierRequest Structure

Specifies a JSON classifier to be updated.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- **JsonPath** – UTF-8 string.

A JsonPath string defining the JSON data for the classifier to classify. AWS Glue supports a subset of JsonPath, as described in [Writing JsonPath Custom Classifiers](#).

CreateCsvClassifierRequest Structure

Specifies a custom CSV classifier for `CreateClassifier` to create.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.

- **Delimiter** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what separates each column entry in the row.

- **QuoteSymbol** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what combines content into a single column value. Must be different from the column delimiter.

- **ContainsHeader** – UTF-8 string (valid values: UNKNOWN | PRESENT | ABSENT).

Indicates whether the CSV file contains a header.

- **Header** – An array of UTF-8 strings.

A list of strings representing column names.

- **DisableValueTrimming** – Boolean.

Specifies not to trim values before identifying the type of column values. The default value is true.

- **AllowSingleColumn** – Boolean.

Enables the processing of files that contain only one column.

UpdateCsvClassifierRequest Structure

Specifies a custom CSV classifier to be updated.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the classifier.
- **Delimiter** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what separates each column entry in the row.
- **QuoteSymbol** – UTF-8 string, not less than 1 or more than 1 bytes long, matching the [Custom string pattern #10 \(p. 618\)](#).

A custom symbol to denote what combines content into a single column value. It must be different from the column delimiter.
- **ContainsHeader** – UTF-8 string (valid values: UNKNOWN | PRESENT | ABSENT).

Indicates whether the CSV file contains a header.
- **Header** – An array of UTF-8 strings.

A list of strings representing column names.
- **DisableValueTrimming** – Boolean.

Specifies not to trim values before identifying the type of column values. The default value is true.
- **AllowSingleColumn** – Boolean.

Enables the processing of files that contain only one column.

Operations

- [CreateClassifier Action \(Python: create_classifier\) \(p. 519\)](#)
- [DeleteClassifier Action \(Python: delete_classifier\) \(p. 520\)](#)
- [GetClassifier Action \(Python: get_classifier\) \(p. 520\)](#)
- [GetClassifiers Action \(Python: get_classifiers\) \(p. 521\)](#)
- [UpdateClassifier Action \(Python: update_classifier\) \(p. 521\)](#)

CreateClassifier Action (Python: create_classifier)

Creates a classifier in the user's account. This can be a `GrokClassifier`, an `XMLClassifier`, a `JsonClassifier`, or a `CsvClassifier`, depending on which field of the request is present.

Request

- **GrokClassifier** – A [CreateGrokClassifierRequest \(p. 516\)](#) object.

A `GrokClassifier` object specifying the classifier to create.
- **XMLClassifier** – A [CreateXMLClassifierRequest \(p. 517\)](#) object.

An `XMLClassifier` object specifying the classifier to create.
- **JsonClassifier** – A [CreateJsonClassifierRequest \(p. 517\)](#) object.

A `JsonClassifier` object specifying the classifier to create.
- **CsvClassifier** – A [CreateCsvClassifierRequest \(p. 518\)](#) object.

A CsvClassifier object specifying the classifier to create.

Response

- *No Response parameters.*

Errors

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException

DeleteClassifier Action (Python: delete_classifier)

Removes a classifier from the Data Catalog.

Request

- Name – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the classifier to remove.

Response

- *No Response parameters.*

Errors

- EntityNotFoundException
- OperationTimeoutException

GetClassifier Action (Python: get_classifier)

Retrieve a classifier by name.

Request

- Name – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the classifier to retrieve.

Response

- Classifier – A [Classifier \(p. 513\)](#) object.

The requested classifier.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifiers Action (Python: `get_classifiers`)

Lists all classifier objects in the Data Catalog.

Request

- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The size of the list to return (optional).
- `NextToken` – UTF-8 string.
An optional continuation token.

Response

- `Classifiers` – An array of [Classifier \(p. 513\)](#) objects.
The requested list of classifier objects.
- `NextToken` – UTF-8 string.
A continuation token.

Errors

- `OperationTimeoutException`

UpdateClassifier Action (Python: `update_classifier`)

Modifies an existing classifier (a `GrokClassifier`, an `XMLClassifier`, a `JsonClassifier`, or a `CsvClassifier`, depending on which field is present).

Request

- `GrokClassifier` – An [UpdateGrokClassifierRequest \(p. 516\)](#) object.
A `GrokClassifier` object with updated fields.
- `XMLClassifier` – An [UpdateXMLClassifierRequest \(p. 517\)](#) object.
An `XMLClassifier` object with updated fields.
- `JsonClassifier` – An [UpdateJsonClassifierRequest \(p. 518\)](#) object.
A `JsonClassifier` object with updated fields.
- `CsvClassifier` – An [UpdateCsvClassifierRequest \(p. 518\)](#) object.
A `CsvClassifier` object with updated fields.

Response

- *No Response parameters.*

Errors

- `InvalidArgumentException`
- `VersionMismatchException`
- `EntityNotFoundException`
- `OperationTimeoutException`

Crawler API

The Crawler API describes AWS Glue crawler data types, along with the API for creating, deleting, updating, and listing crawlers.

Data Types

- [Crawler Structure \(p. 522\)](#)
- [Schedule Structure \(p. 523\)](#)
- [CrawlerTargets Structure \(p. 523\)](#)
- [S3Target Structure \(p. 524\)](#)
- [JdbcTarget Structure \(p. 524\)](#)
- [DynamoDBTarget Structure \(p. 524\)](#)
- [CatalogTarget Structure \(p. 525\)](#)
- [CrawlerMetrics Structure \(p. 525\)](#)
- [SchemaChangePolicy Structure \(p. 525\)](#)
- [LastCrawlInfo Structure \(p. 526\)](#)

Crawler Structure

Specifies a crawler program that examines a data source and uses classifiers to try to determine its schema. If successful, the crawler records metadata concerning the data source in the AWS Glue Data Catalog.

Fields

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler.

- `Role` – UTF-8 string.

The Amazon Resource Name (ARN) of an IAM role that's used to access customer resources, such as Amazon Simple Storage Service (Amazon S3) data.

- `Targets` – A [CrawlerTargets \(p. 523\)](#) object.

A collection of targets to crawl.

- `DatabaseName` – UTF-8 string.

The name of the database in which the crawler's output is stored.

- `Description` – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the crawler.

- **Classifiers** – An array of UTF-8 strings.
A list of UTF-8 strings that specify the custom classifiers that are associated with the crawler.
- **SchemaChangePolicy** – A [SchemaChangePolicy \(p. 525\)](#) object.
The policy that specifies update and delete behaviors for the crawler.
- **State** – UTF-8 string (valid values: READY | RUNNING | STOPPING).
Indicates whether the crawler is running, or whether a run is pending.
- **TablePrefix** – UTF-8 string, not more than 128 bytes long.
The prefix added to the names of tables that are created.
- **Schedule** – A [Schedule \(p. 533\)](#) object.
For scheduled crawlers, the schedule when the crawler runs.
- **CrawlElapsedTime** – Number (long).
If the crawler is running, contains the total time elapsed since the last crawl began.
- **CreationTime** – Timestamp.
The time that the crawler was created.
- **LastUpdated** – Timestamp.
The time that the crawler was last updated.
- **LastCrawl** – A [LastCrawlInfo \(p. 526\)](#) object.
The status of the last crawl, and potentially error information if an error occurred.
- **Version** – Number (long).
The version of the crawler.
- **Configuration** – UTF-8 string.
Crawler configuration information. This versioned JSON string allows users to specify aspects of a crawler's behavior. For more information, see [Configuring a Crawler](#).
- **CrawlerSecurityConfiguration** – UTF-8 string, not more than 128 bytes long.
The name of the `SecurityConfiguration` structure to be used by this crawler.

Schedule Structure

A scheduling object using a `cron` statement to schedule an event.

Fields

- **ScheduleExpression** – UTF-8 string.
A `cron` expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`).
- **State** – UTF-8 string (valid values: SCHEDULED | NOT_SCHEDULED | TRANSITIONING).
The state of the schedule.

CrawlerTargets Structure

Specifies data stores to crawl.

Fields

- **S3Targets** – An array of [S3Target \(p. 524\)](#) objects.
Specifies Amazon Simple Storage Service (Amazon S3) targets.
- **JdbcTargets** – An array of [JdbcTarget \(p. 524\)](#) objects.
Specifies JDBC targets.
- **DynamoDBTargets** – An array of [DynamoDBTarget \(p. 524\)](#) objects.
Specifies Amazon DynamoDB targets.
- **CatalogTargets** – An array of [CatalogTarget \(p. 525\)](#) objects.
Specifies AWS Glue Data Catalog targets.

S3Target Structure

Specifies a data store in Amazon Simple Storage Service (Amazon S3).

Fields

- **Path** – UTF-8 string.
The path to the Amazon S3 target.
- **Exclusions** – An array of UTF-8 strings.
A list of glob patterns used to exclude from the crawl. For more information, see [Catalog Tables with a Crawler](#).

JdbcTarget Structure

Specifies a JDBC data store to crawl.

Fields

- **ConnectionName** – UTF-8 string.
The name of the connection to use to connect to the JDBC target.
- **Path** – UTF-8 string.
The path of the JDBC target.
- **Exclusions** – An array of UTF-8 strings.
A list of glob patterns used to exclude from the crawl. For more information, see [Catalog Tables with a Crawler](#).

DynamoDBTarget Structure

Specifies an Amazon DynamoDB table to crawl.

Fields

- **Path** – UTF-8 string.

The name of the DynamoDB table to crawl.

CatalogTarget Structure

Specifies an AWS Glue Data Catalog target.

Fields

- **DatabaseName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the database to be synchronized.

- **Tables** – *Required*: An array of UTF-8 strings, at least 1 string.

A list of the tables to be synchronized.

CrawlerMetrics Structure

Metrics for a specified crawler.

Fields

- **CrawlerName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler.

- **TimeLeftSeconds** – Number (double), not more than None.

The estimated time left to complete a running crawl.

- **StillEstimating** – Boolean.

True if the crawler is still estimating how long it will take to complete this run.

- **LastRuntimeSeconds** – Number (double), not more than None.

The duration of the crawler's most recent run, in seconds.

- **MedianRuntimeSeconds** – Number (double), not more than None.

The median duration of this crawler's runs, in seconds.

- **TablesCreated** – Number (integer), not more than None.

The number of tables created by this crawler.

- **TablesUpdated** – Number (integer), not more than None.

The number of tables updated by this crawler.

- **TablesDeleted** – Number (integer), not more than None.

The number of tables deleted by this crawler.

SchemaChangePolicy Structure

A policy that specifies update and deletion behaviors for the crawler.

Fields

- **UpdateBehavior** – UTF-8 string (valid values: `LOG` | `UPDATE_IN_DATABASE`).
The update behavior when the crawler finds a changed schema.
- **DeleteBehavior** – UTF-8 string (valid values: `LOG` | `DELETE_FROM_DATABASE` | `DEPRECATE_IN_DATABASE`).
The deletion behavior when the crawler finds a deleted object.

LastCrawlInfo Structure

Status and error information about the most recent crawl.

Fields

- **Status** – UTF-8 string (valid values: `SUCCEEDED` | `CANCELLED` | `FAILED`).
Status of the last crawl.
- **ErrorMessage** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
If an error occurred, the error information about the last crawl.
- **LogGroup** – UTF-8 string, not less than 1 or more than 512 bytes long, matching the [Log group string pattern \(p. 618\)](#).
The log group for the last crawl.
- **LogStream** – UTF-8 string, not less than 1 or more than 512 bytes long, matching the [Log-stream string pattern \(p. 618\)](#).
The log stream for the last crawl.
- **MessagePrefix** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The prefix for a message about this crawl.
- **StartTime** – Timestamp.
The time at which the crawl started.

Operations

- [CreateCrawler Action \(Python: create_crawler\) \(p. 527\)](#)
- [DeleteCrawler Action \(Python: delete_crawler\) \(p. 528\)](#)
- [GetCrawler Action \(Python: get_crawler\) \(p. 528\)](#)
- [GetCrawlers Action \(Python: get_crawlers\) \(p. 529\)](#)
- [GetCrawlerMetrics Action \(Python: get_crawler_metrics\) \(p. 529\)](#)
- [UpdateCrawler Action \(Python: update_crawler\) \(p. 530\)](#)
- [StartCrawler Action \(Python: start_crawler\) \(p. 531\)](#)
- [StopCrawler Action \(Python: stop_crawler\) \(p. 531\)](#)
- [BatchGetCrawlers Action \(Python: batch_get_crawlers\) \(p. 532\)](#)
- [ListCrawlers Action \(Python: list_crawlers\) \(p. 532\)](#)

CreateCrawler Action (Python: `create_crawler`)

Creates a new crawler with specified targets, role, configuration, and optional schedule. At least one crawl target must be specified, in the `s3Targets` field, the `jdbcTargets` field, or the `DynamoDBTargets` field.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the new crawler.

- **Role** – *Required*: UTF-8 string.

The IAM role or Amazon Resource Name (ARN) of an IAM role used by the new crawler to access customer resources.

- **DatabaseName** – UTF-8 string.

The AWS Glue database where results are written, such as: `arn:aws:daylight:us-east-1::database/sometable/*`.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the new crawler.

- **Targets** – *Required*: A [CrawlerTargets \(p. 523\)](#) object.

A list of collection of targets to crawl.

- **Schedule** – UTF-8 string.

A cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#)). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`.

- **Classifiers** – An array of UTF-8 strings.

A list of custom classifiers that the user has registered. By default, all built-in classifiers are included in a crawl, but these custom classifiers always override the default classifiers for a given classification.

- **TablePrefix** – UTF-8 string, not more than 128 bytes long.

The table prefix used for catalog tables that are created.

- **SchemaChangePolicy** – A [SchemaChangePolicy \(p. 525\)](#) object.

The policy for the crawler's update and deletion behavior.

- **Configuration** – UTF-8 string.

Crawler configuration information. This versioned JSON string allows users to specify aspects of a crawler's behavior. For more information, see [Configuring a Crawler](#).

- **CrawlerSecurityConfiguration** – UTF-8 string, not more than 128 bytes long.

The name of the `SecurityConfiguration` structure to be used by this crawler.

- **Tags** – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The tags to use with this crawler request. You may use tags to limit access to the crawler. For more information about tags in AWS Glue, see [AWS Tags in AWS Glue](#) in the developer guide.

Response

- *No Response parameters.*

Errors

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCrawler Action (Python: `delete_crawler`)

Removes a specified crawler from the AWS Glue Data Catalog, unless the crawler state is `RUNNING`.

Request

- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler to remove.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `CrawlerRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

GetCrawler Action (Python: `get_crawler`)

Retrieves metadata for a specified crawler.

Request

- `Name` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler to retrieve metadata for.

Response

- `Crawler` – A [Crawler \(p. 522\)](#) object.

The metadata for the specified crawler.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`

GetCrawlers Action (Python: get_crawlers)

Retrieves metadata for all crawlers defined in the customer account.

Request

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The number of crawlers to return on each call.

- `NextToken` – UTF-8 string.

A continuation token, if this is a continuation request.

Response

- `Crawlers` – An array of [Crawler \(p. 522\)](#) objects.

A list of crawler metadata.

- `NextToken` – UTF-8 string.

A continuation token, if the returned list has not reached the end of those defined in this customer account.

Errors

- `OperationTimeoutException`

GetCrawlerMetrics Action (Python: get_crawler_metrics)

Retrieves metrics about specified crawlers.

Request

- `CrawlerNameList` – An array of UTF-8 strings, not more than 100 strings.

A list of the names of crawlers about which to retrieve metrics.

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum size of a list to return.

- `NextToken` – UTF-8 string.

A continuation token, if this is a continuation call.

Response

- `CrawlerMetricsList` – An array of [CrawlerMetrics \(p. 525\)](#) objects.

A list of metrics for the specified crawler.

- **NextToken** – UTF-8 string.

A continuation token, if the returned list does not contain the last metric available.

Errors

- `OperationTimeoutException`

UpdateCrawler Action (Python: update_crawler)

Updates a crawler. If a crawler is running, you must stop it using `StopCrawler` before updating it.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the new crawler.

- **Role** – UTF-8 string.

The IAM role or Amazon Resource Name (ARN) of an IAM role that is used by the new crawler to access customer resources.

- **DatabaseName** – UTF-8 string.

The AWS Glue database where results are stored, such as: `arn:aws:daylight:us-east-1::database/sometable/*`.

- **Description** – UTF-8 string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the new crawler.

- **Targets** – A [CrawlerTargets \(p. 523\)](#) object.

A list of targets to crawl.

- **Schedule** – UTF-8 string.

A cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`).

- **Classifiers** – An array of UTF-8 strings.

A list of custom classifiers that the user has registered. By default, all built-in classifiers are included in a crawl, but these custom classifiers always override the default classifiers for a given classification.

- **TablePrefix** – UTF-8 string, not more than 128 bytes long.

The table prefix used for catalog tables that are created.

- **SchemaChangePolicy** – A [SchemaChangePolicy \(p. 525\)](#) object.

The policy for the crawler's update and deletion behavior.

- **Configuration** – UTF-8 string.

Crawler configuration information. This versioned JSON string allows users to specify aspects of a crawler's behavior. For more information, see [Configuring a Crawler](#).

- **CrawlerSecurityConfiguration** – UTF-8 string, not more than 128 bytes long.

The name of the `SecurityConfiguration` structure to be used by this crawler.

Response

- *No Response parameters.*

Errors

- `InvalidInputException`
- `VersionMismatchException`
- `EntityNotFoundException`
- `CrawlerRunningException`
- `OperationTimeoutException`

StartCrawler Action (Python: start_crawler)

Starts a crawl using the specified crawler, regardless of what is scheduled. If the crawler is already running, returns a [CrawlerRunningException](#).

Request

- Name – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the crawler to start.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `CrawlerRunningException`
- `OperationTimeoutException`

StopCrawler Action (Python: stop_crawler)

If the specified crawler is running, stops the crawl.

Request

- Name – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the crawler to stop.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`

- `CrawlerNotRunningException`
- `CrawlerStoppingException`
- `OperationTimeoutException`

BatchGetCrawlers Action (Python: `batch_get_crawlers`)

Returns a list of resource metadata for a given list of crawler names. After calling the `ListCrawlers` operation, you can call this operation to access the data to which you have been granted permissions. This operation supports all IAM permissions, including permission conditions that uses tags.

Request

- `CrawlerNames` – *Required:* An array of UTF-8 strings, not more than 100 strings.
A list of crawler names, which might be the names returned from the `ListCrawlers` operation.

Response

- `Crawlers` – An array of [Crawler \(p. 522\)](#) objects.
A list of crawler definitions.
- `CrawlersNotFound` – An array of UTF-8 strings, not more than 100 strings.
A list of names of crawlers that were not found.

Errors

- `InvalidArgumentException`
- `OperationTimeoutException`

ListCrawlers Action (Python: `list_crawlers`)

Retrieves the names of all crawler resources in this AWS account, or the resources with the specified tag. This operation allows you to see which resources are available in your account, and their names.

This operation takes the optional `Tags` field, which you can use as a filter on the response so that tagged resources can be retrieved as a group. If you choose to use tags filtering, only resources with the tag are retrieved.

Request

- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum size of a list to return.
- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation request.
- `Tags` – A map array of key-value pairs, not more than 50 pairs.
Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.
Each value is a UTF-8 string, not more than 256 bytes long.
Specifies to return only these tagged resources.

Response

- **CrawlerNames** – An array of UTF-8 strings, not more than 100 strings.
The names of all crawlers in the account, or the crawlers with the specified tags.
- **NextToken** – UTF-8 string.
A continuation token, if the returned list does not contain the last metric available.

Errors

- `OperationTimeoutException`

Crawler Scheduler API

The Crawler Scheduler API describes AWS Glue crawler data types, along with the API for creating, deleting, updating, and listing crawlers.

Data Types

- [Schedule Structure \(p. 533\)](#)

Schedule Structure

A scheduling object using a `cron` statement to schedule an event.

Fields

- **ScheduleExpression** – UTF-8 string.
A `cron` expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`.
- **State** – UTF-8 string (valid values: `SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`).
The state of the schedule.

Operations

- [UpdateCrawlerSchedule Action \(Python: update_crawler_schedule\) \(p. 533\)](#)
- [StartCrawlerSchedule Action \(Python: start_crawler_schedule\) \(p. 534\)](#)
- [StopCrawlerSchedule Action \(Python: stop_crawler_schedule\) \(p. 534\)](#)

UpdateCrawlerSchedule Action (Python: update_crawler_schedule)

Updates the schedule of a crawler using a `cron` expression.

Request

- **CrawlerName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler whose schedule to update.

- **Schedule** – UTF-8 string.

The updated cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: cron(15 12 * * ? *).

Response

- *No Response parameters.*

Errors

- EntityNotFoundException
- InvalidInputException
- VersionMismatchException
- SchedulerTransitioningException
- OperationTimeoutException

StartCrawlerSchedule Action (Python: start_crawler_schedule)

Changes the schedule state of the specified crawler to SCHEDULED, unless the crawler is already running or the schedule state is already SCHEDULED.

Request

- **CrawlerName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the crawler to schedule.

Response

- *No Response parameters.*

Errors

- EntityNotFoundException
- SchedulerRunningException
- SchedulerTransitioningException
- NoScheduleException
- OperationTimeoutException

StopCrawlerSchedule Action (Python: stop_crawler_schedule)

Sets the schedule state of the specified crawler to NOT_SCHEDULED, but does not stop the crawler if it is already running.

Request

- **CrawlerName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the crawler whose schedule state to set.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `SchedulerNotRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

Autogenerating ETL Scripts API

The ETL script-generation API describes the datatypes and API for generating ETL scripts in AWS Glue.

Data Types

- [CodeGenNode Structure \(p. 535\)](#)
- [CodeGenNodeArg Structure \(p. 536\)](#)
- [CodeGenEdge Structure \(p. 536\)](#)
- [Location Structure \(p. 536\)](#)
- [CatalogEntry Structure \(p. 537\)](#)
- [MappingEntry Structure \(p. 537\)](#)

CodeGenNode Structure

Represents a node in a directed acyclic graph (DAG)

Fields

- **Id** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Identifier string pattern \(p. 618\)](#).

A node identifier that is unique within the node's graph.

- **NodeType** – *Required:* UTF-8 string.

The type of node that this is.

- **Args** – *Required:* An array of [CodeGenNodeArg \(p. 536\)](#) objects, not more than 50 structures.

Properties of the node, in the form of name-value pairs.

- **LineNumber** – Number (integer).

The line number of the node.

CodeGenNodeArg Structure

An argument or property of a node.

Fields

- **Name** – *Required*: UTF-8 string.

The name of the argument or property.

- **Value** – *Required*: UTF-8 string.

The value of the argument or property.

- **Param** – Boolean.

True if the value is used as a parameter.

CodeGenEdge Structure

Represents a directional edge in a directed acyclic graph (DAG).

Fields

- **Source** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Identifier string pattern \(p. 618\)](#).

The ID of the node at which the edge starts.

- **Target** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Identifier string pattern \(p. 618\)](#).

The ID of the node at which the edge ends.

- **TargetParameter** – UTF-8 string.

The target of the edge.

Location Structure

The location of resources.

Fields

- **Jdbc** – An array of [CodeGenNodeArg \(p. 536\)](#) objects, not more than 50 structures.

A JDBC location.

- **S3** – An array of [CodeGenNodeArg \(p. 536\)](#) objects, not more than 50 structures.

An Amazon Simple Storage Service (Amazon S3) location.

- **DynamoDB** – An array of [CodeGenNodeArg \(p. 536\)](#) objects, not more than 50 structures.

An Amazon DynamoDB table location.

CatalogEntry Structure

Specifies a table definition in the AWS Glue Data Catalog.

Fields

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The database in which the table metadata resides.

- `TableName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the table in question.

MappingEntry Structure

Defines a mapping.

Fields

- `SourceTable` – UTF-8 string.

The name of the source table.

- `SourcePath` – UTF-8 string.

The source path.

- `SourceType` – UTF-8 string.

The source type.

- `TargetTable` – UTF-8 string.

The target table.

- `TargetPath` – UTF-8 string.

The target path.

- `TargetType` – UTF-8 string.

The target type.

Operations

- [CreateScript Action \(Python: create_script\) \(p. 537\)](#)
- [GetDataflowGraph Action \(Python: get_dataflow_graph\) \(p. 538\)](#)
- [GetMapping Action \(Python: get_mapping\) \(p. 539\)](#)
- [GetPlan Action \(Python: get_plan\) \(p. 539\)](#)

CreateScript Action (Python: create_script)

Transforms a directed acyclic graph (DAG) into code.

Request

- **DagNodes** – An array of [CodeGenNode \(p. 535\)](#) objects.
A list of the nodes in the DAG.
- **DagEdges** – An array of [CodeGenEdge \(p. 536\)](#) objects.
A list of the edges in the DAG.
- **Language** – UTF-8 string (valid values: **PYTHON** | **SCALA**).
The programming language of the resulting code from the DAG.

Response

- **PythonScript** – UTF-8 string.
The Python script generated from the DAG.
- **ScalaCode** – UTF-8 string.
The Scala code generated from the DAG.

Errors

- **InvalidArgumentException**
- **InternalServiceException**
- **OperationTimeoutException**

GetDataflowGraph Action (Python: get_dataflow_graph)

Transforms a Python script into a directed acyclic graph (DAG).

Request

- **PythonScript** – UTF-8 string.
The Python script to transform.

Response

- **DagNodes** – An array of [CodeGenNode \(p. 535\)](#) objects.
A list of the nodes in the resulting DAG.
- **DagEdges** – An array of [CodeGenEdge \(p. 536\)](#) objects.
A list of the edges in the resulting DAG.

Errors

- **InvalidArgumentException**
- **InternalServiceException**

- `OperationTimeoutException`

GetMapping Action (Python: get_mapping)

Creates mappings.

Request

- **Source** – *Required*: A [CatalogEntry \(p. 537\)](#) object.
Specifies the source table.
- **Sinks** – An array of [CatalogEntry \(p. 537\)](#) objects.
A list of target tables.
- **Location** – A [Location \(p. 536\)](#) object.
Parameters for the mapping.

Response

- **Mapping** – *Required*: An array of [MappingEntry \(p. 537\)](#) objects.
A list of mappings to the specified targets.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetPlan Action (Python: get_plan)

Gets code to perform a specified mapping.

Request

- **Mapping** – *Required*: An array of [MappingEntry \(p. 537\)](#) objects.
The list of mappings from a source table to target tables.
- **Source** – *Required*: A [CatalogEntry \(p. 537\)](#) object.
The source table.
- **Sinks** – An array of [CatalogEntry \(p. 537\)](#) objects.
The target tables.
- **Location** – A [Location \(p. 536\)](#) object.
The parameters for the mapping.
- **Language** – UTF-8 string (valid values: `PYTHON` | `SCALA`).
The programming language of the code to perform the mapping.

Response

- **PythonScript** – UTF-8 string.
A Python script to perform the mapping.
- **ScalaCode** – UTF-8 string.
The Scala code to perform the mapping.

Errors

- **InvalidInputException**
- **InternalServiceException**
- **OperationTimeoutException**

Jobs API

The Jobs API describes jobs data types and contains APIs for working with jobs, job runs, and triggers in AWS Glue.

Topics

- [Jobs \(p. 540\)](#)
- [Job Runs \(p. 551\)](#)
- [Triggers \(p. 561\)](#)

Jobs

The Jobs API describes the data types and API related to creating, updating, deleting, or viewing jobs in AWS Glue.

Data Types

- [Job Structure \(p. 540\)](#)
- [ExecutionProperty Structure \(p. 542\)](#)
- [NotificationProperty Structure \(p. 543\)](#)
- [JobCommand Structure \(p. 543\)](#)
- [ConnectionsList Structure \(p. 543\)](#)
- [JobUpdate Structure \(p. 543\)](#)

Job Structure

Specifies a job definition.

Fields

- [Name](#) – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name you assign to this job definition.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the job.

- **LogUri** – UTF-8 string.

This field is reserved for future use.

- **Role** – UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role associated with this job.

- **CreatedOn** – Timestamp.

The time and date that this job definition was created.

- **LastModifiedOn** – Timestamp.

The last point in time when this job definition was modified.

- **ExecutionProperty** – An [ExecutionProperty \(p. 542\)](#) object.

An [ExecutionProperty](#) specifying the maximum number of concurrent runs allowed for this job.

- **Command** – A [JobCommand \(p. 543\)](#) object.

The [JobCommand](#) that executes this job.

- **DefaultArguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The default arguments for this job, specified as name-value pairs.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own Job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- **Connections** – A [ConnectionsList \(p. 543\)](#) object.

The connections used for this job.

- **MaxRetries** – Number (integer).

The maximum number of times to retry this job after a JobRun fails.

- **AllocatedCapacity** – Number (integer).

This field is deprecated. Use [MaxCapacity](#) instead.

The number of AWS Glue data processing units (DPUs) allocated to runs of this job. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

- **Timeout** – Number (integer), at least 1.

The job timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters [TIMEOUT](#) status. The default is 2,880 minutes (48 hours).

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

Do not set `MaxCapacity` if using `WorkerType` and `NumberOfWorkers`.

The value that can be allocated for `MaxCapacity` depends on whether you are running a Python shell job or an Apache Spark ETL job:

- When you specify a Python shell job (`JobCommand.Name="pythonshell"`), you can allocate either 0.0625 or 1 DPU. The default is 0.0625 DPU.
- When you specify an Apache Spark ETL job (`JobCommand.Name="glueetl"`), you can allocate from 2 to 100 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.
- `WorkerType` – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when a job runs. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- `NumberOfWorkers` – Number (integer).

The number of workers of a defined `workerType` that are allocated when a job runs.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this job.

- `NotificationProperty` – A [NotificationProperty \(p. 543\)](#) object.

Specifies configuration properties of a job notification.

- `GlueVersion` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

Jobs that are created without specifying a Glue version default to Glue 0.9.

ExecutionProperty Structure

An execution property of a job.

Fields

- `MaxConcurrentRuns` – Number (integer).

The maximum number of concurrent runs allowed for the job. The default is 1. An error is returned when this threshold is reached. The maximum value you can specify is controlled by a service limit.

NotificationProperty Structure

Specifies configuration properties of a notification.

Fields

- **NotifyDelayAfter** – Number (integer), at least 1.

After a job run starts, the number of minutes to wait before sending a job run delay notification.

JobCommand Structure

Specifies code executed when a job is run.

Fields

- **Name** – UTF-8 string.

The name of the job command. For an Apache Spark ETL job, this must be `glueetl`. For a Python shell job, it must be `pythonshell`.

- **ScriptLocation** – UTF-8 string.

Specifies the Amazon Simple Storage Service (Amazon S3) path to a script that executes a job.

- **PythonVersion** – UTF-8 string, matching the [Custom string pattern #11 \(p. 618\)](#).

The Python version being used to execute a Python shell job. Allowed values are 2 or 3.

ConnectionsList Structure

Specifies the connections used by a job.

Fields

- **Connections** – An array of UTF-8 strings.

A list of connections used by the job.

JobUpdate Structure

Specifies information used to update an existing job definition. The previous job definition is completely overwritten by this information.

Fields

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

Description of the job being defined.

- **LogUri** – UTF-8 string.

This field is reserved for future use.

- **Role** – UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role associated with this job (required).

- **ExecutionProperty** – An [ExecutionProperty \(p. 542\)](#) object.

An `ExecutionProperty` specifying the maximum number of concurrent runs allowed for this job.

- `Command` – A [JobCommand \(p. 543\)](#) object.
The `JobCommand` that executes this job (required).
- `DefaultArguments` – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The default arguments for this job.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own Job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- `Connections` – A [ConnectionsList \(p. 543\)](#) object.

The connections used for this job.

- `MaxRetries` – Number (integer).

The maximum number of times to retry this job if it fails.

- `AllocatedCapacity` – Number (integer).

This field is deprecated. Use `MaxCapacity` instead.

The number of AWS Glue data processing units (DPUs) to allocate to this job. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

- `Timeout` – Number (integer), at least 1.

The job timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours).

- `MaxCapacity` – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

Do not set `Max Capacity` if using `WorkerType` and `NumberOfWorkers`.

The value that can be allocated for `MaxCapacity` depends on whether you are running a Python shell job or an Apache Spark ETL job:

- When you specify a Python shell job (`JobCommand.Name="pythonshell"`), you can allocate either 0.0625 or 1 DPU. The default is 0.0625 DPU.
- When you specify an Apache Spark ETL job (`JobCommand.Name="glueetl"`), you can allocate from 2 to 100 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.
- `WorkerType` – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when a job runs. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- **NumberOfWorkers** – Number (integer).

The number of workers of a defined `workerType` that are allocated when a job runs.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this job.

- **NotificationProperty** – A [NotificationProperty \(p. 543\)](#) object.

Specifies the configuration properties of a job notification.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

Operations

- [CreateJob Action \(Python: create_job\) \(p. 545\)](#)
- [UpdateJob Action \(Python: update_job\) \(p. 548\)](#)
- [GetJob Action \(Python: get_job\) \(p. 548\)](#)
- [GetJobs Action \(Python: get_jobs\) \(p. 549\)](#)
- [DeleteJob Action \(Python: delete_job\) \(p. 549\)](#)
- [ListJobs Action \(Python: list_jobs\) \(p. 550\)](#)
- [BatchGetJobs Action \(Python: batch_get_jobs\) \(p. 550\)](#)

CreateJob Action (Python: create_job)

Creates a new job definition.

Request

- **Name** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name you assign to this job definition. It must be unique in your account.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

Description of the job being defined.

- **LogUri** – UTF-8 string.

This field is reserved for future use.

- **Role** – *Required*: UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role associated with this job.

- **ExecutionProperty** – An [ExecutionProperty \(p. 542\)](#) object.

An [ExecutionProperty](#) specifying the maximum number of concurrent runs allowed for this job.

- **Command** – *Required*: A [JobCommand \(p. 543\)](#) object.

The [JobCommand](#) that executes this job.

- **DefaultArguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The default arguments for this job.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own Job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- **Connections** – A [ConnectionsList \(p. 543\)](#) object.

The connections used for this job.

- **MaxRetries** – Number (integer).

The maximum number of times to retry this job if it fails.

- **AllocatedCapacity** – Number (integer).

This parameter is deprecated. Use `MaxCapacity` instead.

The number of AWS Glue data processing units (DPUs) to allocate to this Job. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

- **Timeout** – Number (integer), at least 1.

The job timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours).

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

Do not set `Max Capacity` if using `WorkerType` and `NumberOfWorkers`.

The value that can be allocated for `MaxCapacity` depends on whether you are running a Python shell job or an Apache Spark ETL job:

- When you specify a Python shell job (`JobCommand.Name="pythonshell"`), you can allocate either 0.0625 or 1 DPU. The default is 0.0625 DPU.

- When you specify an Apache Spark ETL job (`JobCommand.Name="glueetl"`), you can allocate from 2 to 100 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.
- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this job.

- `Tags` – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The tags to use with this job. You may use tags to limit access to the job. For more information about tags in AWS Glue, see [AWS Tags in AWS Glue](#) in the developer guide.

- `NotificationProperty` – A [NotificationProperty \(p. 543\)](#) object.

Specifies configuration properties of a job notification.

- `GlueVersion` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

Jobs that are created without specifying a Glue version default to Glue 0.9.

- `NumberOfWorkers` – Number (integer).

The number of workers of a defined `workerType` that are allocated when a job runs.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- `WorkerType` – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when a job runs. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique name that was provided for this job definition.

Errors

- `InvalidArgumentException`
- `IdempotentParameterMismatchException`
- `AlreadyExistsException`

- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

UpdateJob Action (Python: update_job)

Updates an existing job definition.

Request

- `JobName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition to update.

- `JobUpdate` – *Required*: A [JobUpdate \(p. 543\)](#) object.

Specifies the values with which to update the job definition.

Response

- `JobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Returns the name of the updated job definition.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

GetJob Action (Python: get_job)

Retrieves an existing job definition.

Request

- `JobName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition to retrieve.

Response

- `Job` – A [Job \(p. 540\)](#) object.

The requested job definition.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobs Action (Python: get_jobs)

Retrieves all current job definitions.

Request

- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation call.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum size of the response.

Response

- `Jobs` – An array of [Job \(p. 540\)](#) objects.
A list of job definitions.
- `NextToken` – UTF-8 string.
A continuation token, if not all job definitions have yet been returned.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

DeleteJob Action (Python: delete_job)

Deletes a specified job definition. If the job definition is not found, no exception is thrown.

Request

- `JobName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the job definition to delete.

Response

- `JobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the job definition that was deleted.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

ListJobs Action (Python: list_jobs)

Retrieves the names of all job resources in this AWS account, or the resources with the specified tag. This operation allows you to see which resources are available in your account, and their names.

This operation takes the optional `Tags` field, which you can use as a filter on the response so that tagged resources can be retrieved as a group. If you choose to use tags filtering, only resources with the tag are retrieved.

Request

- `NextToken` – UTF-8 string.
 - A continuation token, if this is a continuation request.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.
 - The maximum size of a list to return.
- `Tags` – A map array of key-value pairs, not more than 50 pairs.
 - Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.
 - Each value is a UTF-8 string, not more than 256 bytes long.
 - Specifies to return only these tagged resources.

Response

- `JobNames` – An array of UTF-8 strings.
 - The names of all jobs in the account, or the jobs with the specified tags.
- `NextToken` – UTF-8 string.
 - A continuation token, if the returned list does not contain the last metric available.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetJobs Action (Python: batch_get_jobs)

Returns a list of resource metadata for a given list of job names. After calling the `ListJobs` operation, you can call this operation to access the data to which you have been granted permissions. This operation supports all IAM permissions, including permission conditions that uses tags.

Request

- **JobNames** – *Required:* An array of UTF-8 strings.

A list of job names, which might be the names returned from the `ListJobs` operation.

Response

- **Jobs** – An array of [Job \(p. 540\)](#) objects.

A list of job definitions.

- **JobsNotFound** – An array of UTF-8 strings.

A list of names of jobs not found.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`

Job Runs

The Jobs Runs API describes the data types and API related to starting, stopping, or viewing job runs, and resetting job bookmarks, in AWS Glue.

Data Types

- [JobRun Structure \(p. 551\)](#)
- [Predecessor Structure \(p. 554\)](#)
- [JobBookmarkEntry Structure \(p. 554\)](#)
- [BatchStopJobRunSuccessfulSubmission Structure \(p. 554\)](#)
- [BatchStopJobRunError Structure \(p. 555\)](#)

JobRun Structure

Contains information about a job run.

Fields

- **Id** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of this job run.
- **Attempt** – Number (integer).

The number of the attempt to run this job.
- **PreviousRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the previous run of this job. For example, the `JobRunId` specified in the `StartJobRun` action.

- `TriggerName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger that started this job run.

- `JobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition being used in this run.

- `StartedOn` – Timestamp.

The date and time at which this job run was started.

- `LastModifiedOn` – Timestamp.

The last time that this job run was modified.

- `CompletedOn` – Timestamp.

The date and time that this job run completed.

- `JobRunState` – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The current state of the job run.

- `Arguments` – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The job arguments associated with this run. For this job run, they replace the default arguments set in the job definition itself.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- `ErrorMessage` – UTF-8 string.

An error message associated with this job run.

- `PredecessorRuns` – An array of [Predecessor \(p. 554\)](#) objects.

A list of predecessors to this job run.

- `AllocatedCapacity` – Number (integer).

This field is deprecated. Use `MaxCapacity` instead.

The number of AWS Glue data processing units (DPUs) allocated to this JobRun. From 2 to 100 DPUs can be allocated; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

- `ExecutionTime` – Number (integer).

The amount of time (in seconds) that the job run consumed resources.

- `Timeout` – Number (integer), at least 1.

The `JobRun` timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours). This overrides the timeout value set in the parent job.

- `MaxCapacity` – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

Do not set `Max Capacity` if using `WorkerType` and `NumberOfWorkers`.

The value that can be allocated for `MaxCapacity` depends on whether you are running a Python shell job or an Apache Spark ETL job:

- When you specify a Python shell job (`JobCommand.Name="pythonshell"`), you can allocate either 0.0625 or 1 DPU. The default is 0.0625 DPU.
- When you specify an Apache Spark ETL job (`JobCommand.Name="glueetl"`), you can allocate from 2 to 100 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.
- `WorkerType` – UTF-8 string (valid values: `Standard="" | G.1X="" | G.2X=""`).

The type of predefined worker that is allocated when a job runs. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.
- For the G.2X worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.
- `NumberOfWorkers` – Number (integer).

The number of workers of a defined `workerType` that are allocated when a job runs.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this job run.

- `LogGroupName` – UTF-8 string.

The name of the log group for secure logging that can be server-side encrypted in Amazon CloudWatch using AWS KMS. This name can be `/aws-glue/jobs/`, in which case the default encryption is `NONE`. If you add a role name and `SecurityConfiguration` name (in other words, `/aws-glue/jobs-yourRoleName-yourSecurityConfigurationName/`), then that security configuration is used to encrypt the log group.

- `NotificationProperty` – A [NotificationProperty \(p. 543\)](#) object.

Specifies configuration properties of a job run notification.

- `GlueVersion` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

Jobs that are created without specifying a Glue version default to Glue 0.9.

Predecessor Structure

A job run that was used in the predicate of a conditional trigger that triggered this job run.

Fields

- **JobName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition used by the predecessor job run.

- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The job-run ID of the predecessor job run.

JobBookmarkEntry Structure

Defines a point that a job can resume processing.

Fields

- **JobName** – UTF-8 string.

The name of the job in question.

- **Version** – Number (integer).

The version of the job.

- **Run** – Number (integer).

The run ID number.

- **Attempt** – Number (integer).

The attempt ID number.

- **PreviousRunId** – UTF-8 string.

The unique run identifier associated with the previous job run.

- **RunId** – UTF-8 string.

The run ID number.

- **JobBookmark** – UTF-8 string.

The bookmark itself.

BatchStopJobRunSuccessfulSubmission Structure

Records a successful request to stop a specified JobRun.

Fields

- **JobName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition used in the job run that was stopped.

- **JobRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The `JobRunId` of the job run that was stopped.

BatchStopJobRunError Structure

Records an error that occurred when attempting to stop a specified job run.

Fields

- **JobName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition that is used in the job run in question.

- **JobRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The `JobRunId` of the job run in question.

- **ErrorDetail** – An [ErrorDetail \(p. 617\)](#) object.

Specifies details about the error that was encountered.

Operations

- [StartJobRun Action \(Python: start_job_run\) \(p. 555\)](#)
- [BatchStopJobRun Action \(Python: batch_stop_job_run\) \(p. 557\)](#)
- [GetJobRun Action \(Python: get_job_run\) \(p. 558\)](#)
- [GetJobRuns Action \(Python: get_job_runs\) \(p. 558\)](#)
- [GetJobBookmark Action \(Python: get_job_bookmark\) \(p. 559\)](#)
- [GetJobBookmarks Action \(Python: get_job_bookmarks\) \(p. 559\)](#)
- [ResetJobBookmark Action \(Python: reset_job_bookmark\) \(p. 560\)](#)

StartJobRun Action (Python: start_job_run)

Starts a job run using a job definition.

Request

- **JobName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition to use.

- **JobRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of a previous `JobRun` to retry.

- **Arguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The job arguments specifically for this run. For this job run, they replace the default arguments set in the job definition itself.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own Job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- AllocatedCapacity – Number (integer).

This field is deprecated. Use MaxCapacity instead.

The number of AWS Glue data processing units (DPUs) to allocate to this JobRun. From 2 to 100 DPUs can be allocated; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

- Timeout – Number (integer), at least 1.

The JobRun timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters TIMEOUT status. The default is 2,880 minutes (48 hours). This overrides the timeout value set in the parent job.

- MaxCapacity – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

Do not set Max Capacity if using WorkerType and NumberOfWorkers.

The value that can be allocated for MaxCapacity depends on whether you are running a Python shell job, or an Apache Spark ETL job:

- When you specify a Python shell job (`JobCommand.Name="pythonshell"`), you can allocate either 0.0625 or 1 DPU. The default is 0.0625 DPU.
- When you specify an Apache Spark ETL job (`JobCommand.Name="glueetl"`), you can allocate from 2 to 100 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.
- SecurityConfiguration – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the SecurityConfiguration structure to be used with this job run.

- NotificationProperty – A [NotificationProperty \(p. 543\)](#) object.

Specifies configuration properties of a job run notification.

- WorkerType – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when a job runs. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.

- For the G.2X worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.
- **NumberOfWorkers** – Number (integer).

The number of workers of a defined `workerType` that are allocated when a job runs.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

Response

- **JobRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID assigned to this job run.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentRunsExceededException`

BatchStopJobRun Action (Python: batch_stop_job_run)

Stops one or more job runs for a specified job definition.

Request

- **JobName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job definition for which to stop job runs.

- **JobRunIds** – *Required:* An array of UTF-8 strings, not less than 1 or more than 25 strings.

A list of the `JobRunIds` that should be stopped for that job definition.

Response

- **SuccessfulSubmissions** – An array of [BatchStopJobRunSuccessfulSubmission \(p. 554\)](#) objects.

A list of the `JobRuns` that were successfully submitted for stopping.

- **Errors** – An array of [BatchStopJobRunError \(p. 555\)](#) objects.

A list of the errors that were encountered in trying to stop `JobRuns`, including the `JobRunId` for which each error was encountered and details about the error.

Errors

- `InvalidArgumentException`
- `InternalServiceException`

- `OperationTimeoutException`

GetJobRun Action (Python: `get_job_run`)

Retrieves the metadata for a given job run.

Request

- `JobName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
Name of the job definition being run.
- `RunId` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the job run.
- `PredecessorsIncluded` – Boolean.
True if a list of predecessor runs should be returned.

Response

- `JobRun` – A [JobRun \(p. 551\)](#) object.

The requested job-run metadata.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobRuns Action (Python: `get_job_runs`)

Retrieves metadata for all runs of a given job definition.

Request

- `JobName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the job definition for which to retrieve all job runs.
- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation call.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum size of the response.

Response

- `JobRuns` – An array of [JobRun \(p. 551\)](#) objects.

A list of job-run metadata objects.

- **NextToken** – UTF-8 string.

A continuation token, if not all requested job runs have been returned.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobBookmark Action (Python: `get_job_bookmark`)

Returns information on a job bookmark entry.

Request

- **JobName** – *Required*: UTF-8 string.
The name of the job in question.
- **Version** – Number (integer).
The version of the job.
- **RunId** – UTF-8 string.
The unique run identifier associated with this job run.

Response

- **JobBookmarkEntry** – A [JobBookmarkEntry \(p. 554\)](#) object.
A structure that defines a point that a job can resume processing.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ValidationException`

GetJobBookmarks Action (Python: `get_job_bookmarks`)

Returns information on the job bookmark entries. The list is ordered on decreasing version numbers.

Request

- **JobName** – *Required*: UTF-8 string.

The name of the job in question.

- **MaxResults** – Number (integer).

The maximum size of the response.

- **NextToken** – Number (integer).

A continuation token, if this is a continuation call.

Response

- **JobBookmarkEntries** – An array of [JobBookmarkEntry \(p. 554\)](#) objects.

A list of job bookmark entries that defines a point that a job can resume processing.

- **NextToken** – Number (integer).

A continuation token, which has a value of 1 if all the entries are returned, or > 1 if not all requested job runs have been returned.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

ResetJobBookmark Action (Python: `reset_job_bookmark`)

Resets a bookmark entry.

Request

- **JobName** – *Required:* UTF-8 string.

The name of the job in question.

- **RunId** – UTF-8 string.

The unique run identifier associated with this job run.

Response

- **JobBookmarkEntry** – A [JobBookmarkEntry \(p. 554\)](#) object.

The reset bookmark entry.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

Triggers

The Triggers API describes the data types and API related to creating, updating, or deleting, and starting and stopping job triggers in AWS Glue.

Data Types

- [Trigger Structure \(p. 561\)](#)
- [TriggerUpdate Structure \(p. 562\)](#)
- [Predicate Structure \(p. 562\)](#)
- [Condition Structure \(p. 562\)](#)
- [Action Structure \(p. 563\)](#)

Trigger Structure

Information about a specific trigger.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger.

- **WorkflowName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the workflow associated with the trigger.

- **Id** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Reserved for future use.

- **Type** – UTF-8 string (valid values: `SCHEDULED` | `CONDITIONAL` | `ON_DEMAND`).

The type of trigger that this is.

- **State** – UTF-8 string (valid values: `CREATING` | `CREATED` | `ACTIVATING` | `ACTIVATED` | `DEACTIVATING` | `DEACTIVATED` | `DELETING` | `UPDATING`).

The current state of the trigger.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of this trigger.

- **Schedule** – UTF-8 string.

A cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`).

- **Actions** – An array of [Action \(p. 563\)](#) objects.

The actions initiated by this trigger.

- **Predicate** – A [Predicate \(p. 562\)](#) object.

The predicate of this trigger, which defines when it will fire.

TriggerUpdate Structure

A structure used to provide information used to update a trigger. This object updates the previous trigger definition by overwriting it completely.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Reserved for future use.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of this trigger.

- **Schedule** – UTF-8 string.

A cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#)). For example, to run something every day at 12:15 UTC, you would specify: `cron(15 12 * * ? *)`.

- **Actions** – An array of [Action \(p. 563\)](#) objects.

The actions initiated by this trigger.

- **Predicate** – A [Predicate \(p. 562\)](#) object.

The predicate of this trigger, which defines when it will fire.

Predicate Structure

Defines the predicate of the trigger, which determines when it fires.

Fields

- **Logical** – UTF-8 string (valid values: AND | ANY).

An optional field if only one condition is listed. If multiple conditions are listed, then this field is required.

- **Conditions** – An array of [Condition \(p. 562\)](#) objects.

A list of the conditions that determine when the trigger will fire.

Condition Structure

Defines a condition under which a trigger fires.

Fields

- **LogicalOperator** – UTF-8 string (valid values: EQUALS).

A logical operator.

- **JobName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job whose JobRuns this condition applies to, and on which this trigger waits.

- **State** – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The condition state. Currently, the values supported are SUCCEEDED, STOPPED, TIMEOUT, and FAILED.

- CrawlerName – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler to which this condition applies.

- CrawlState – UTF-8 string (valid values: RUNNING | SUCCEEDED | CANCELLED | FAILED).

The state of the crawler to which this condition applies.

Action Structure

Defines an action to be initiated by a trigger.

Fields

- JobName – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of a job to be executed.

- Arguments – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The job arguments used when this trigger fires. For this job run, they replace the default arguments set in the job definition itself.

You can specify arguments here that your own job-execution script consumes, as well as arguments that AWS Glue itself consumes.

For information about how to specify and consume your own Job arguments, see the [Calling AWS Glue APIs in Python](#) topic in the developer guide.

For information about the key-value pairs that AWS Glue consumes to set up your job, see the [Special Parameters Used by AWS Glue](#) topic in the developer guide.

- Timeout – Number (integer), at least 1.

The JobRun timeout in minutes. This is the maximum time that a job run can consume resources before it is terminated and enters TIMEOUT status. The default is 2,880 minutes (48 hours). This overrides the timeout value set in the parent job.

- SecurityConfiguration – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the SecurityConfiguration structure to be used with this action.

- NotificationProperty – A [NotificationProperty \(p. 543\)](#) object.

Specifies configuration properties of a job run notification.

- CrawlerName – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the crawler to be used with this action.

Operations

- [CreateTrigger Action \(Python: create_trigger\) \(p. 564\)](#)
- [StartTrigger Action \(Python: start_trigger\) \(p. 565\)](#)
- [GetTrigger Action \(Python: get_trigger\) \(p. 566\)](#)
- [GetTriggers Action \(Python: get_triggers\) \(p. 566\)](#)
- [UpdateTrigger Action \(Python: update_trigger\) \(p. 567\)](#)
- [StopTrigger Action \(Python: stop_trigger\) \(p. 567\)](#)
- [DeleteTrigger Action \(Python: delete_trigger\) \(p. 568\)](#)
- [ListTriggers Action \(Python: list_triggers\) \(p. 568\)](#)
- [BatchGetTriggers Action \(Python: batch_get_triggers\) \(p. 569\)](#)

CreateTrigger Action (Python: create_trigger)

Creates a new trigger.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger.

- **WorkflowName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the workflow associated with the trigger.

- **Type** – *Required*: UTF-8 string (valid values: SCHEDULED | CONDITIONAL | ON_DEMAND).

The type of the new trigger.

- **Schedule** – UTF-8 string.

A cron expression used to specify the schedule (see [Time-Based Schedules for Jobs and Crawlers](#). For example, to run something every day at 12:15 UTC, you would specify: cron(15 12 * * ? *).

This field is required when the trigger type is SCHEDULED.

- **Predicate** – A [Predicate \(p. 562\)](#) object.

A predicate to specify when the new trigger should fire.

This field is required when the trigger type is CONDITIONAL.

- **Actions** – *Required*: An array of [Action \(p. 563\)](#) objects.

The actions initiated by this trigger when it fires.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the new trigger.

- **StartOnCreation** – Boolean.

Set to true to start SCHEDULED and CONDITIONAL triggers when created. True is not supported for ON_DEMAND triggers.

- **Tags** – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The tags to use with this trigger. You may use tags to limit access to the trigger. For more information about tags in AWS Glue, see [AWS Tags in AWS Glue](#) in the developer guide.

Response

- Name – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger.

Errors

- `AlreadyExistsException`
- `EntityNotFoundException`
- `InvalidArgumentException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

StartTrigger Action (Python: start_trigger)

Starts an existing trigger. See [Triggering Jobs](#) for information about how different types of trigger are started.

Request

- Name – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger to start.

Response

- Name – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger that was started.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`

- `ResourceNumberLimitExceeded`
- `ConcurrentRunsExceeded`

GetTrigger Action (Python: get_trigger)

Retrieves the definition of a trigger.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger to retrieve.

Response

- `Trigger` – A [Trigger \(p. 561\)](#) object.

The requested trigger definition.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

GetTriggers Action (Python: get_triggers)

Gets all the triggers associated with a job.

Request

- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation call.
- `DependentJobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job to retrieve triggers for. The trigger that can start this job is returned, and if there is no such trigger, all triggers are returned.

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum size of the response.

Response

- `Triggers` – An array of [Trigger \(p. 561\)](#) objects.

A list of triggers for the specified job.

- `NextToken` – UTF-8 string.

A continuation token, if not all the requested triggers have yet been returned.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

UpdateTrigger Action (Python: update_trigger)

Updates a trigger definition.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger to update.

- `TriggerUpdate` – *Required*: A [TriggerUpdate \(p. 562\)](#) object.

The new values with which to update the trigger.

Response

- `Trigger` – A [Trigger \(p. 561\)](#) object.

The resulting trigger definition.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

StopTrigger Action (Python: stop_trigger)

Stops a specified trigger.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger to stop.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger that was stopped.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteTrigger Action (Python: `delete_trigger`)

Deletes a specified trigger. If the trigger is not found, no exception is thrown.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger to delete.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the trigger that was deleted.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

ListTriggers Action (Python: `list_triggers`)

Retrieves the names of all trigger resources in this AWS account, or the resources with the specified tag. This operation allows you to see which resources are available in your account, and their names.

This operation takes the optional `Tags` field, which you can use as a filter on the response so that tagged resources can be retrieved as a group. If you choose to use tags filtering, only resources with the tag are retrieved.

Request

- `NextToken` – UTF-8 string.
 - A continuation token, if this is a continuation request.
- `DependentJobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the job for which to retrieve triggers. The trigger that can start this job is returned. If there is no such trigger, all triggers are returned.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.
The maximum size of a list to return.
- **Tags** – A map array of key-value pairs, not more than 50 pairs.
Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.
Each value is a UTF-8 string, not more than 256 bytes long.
Specifies to return only these tagged resources.

Response

- **TriggerNames** – An array of UTF-8 strings.
The names of all triggers in the account, or the triggers with the specified tags.
- **NextToken** – UTF-8 string.
A continuation token, if the returned list does not contain the last metric available.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetTriggers Action (Python: `batch_get_triggers`)

Returns a list of resource metadata for a given list of trigger names. After calling the `ListTriggers` operation, you can call this operation to access the data to which you have been granted permissions. This operation supports all IAM permissions, including permission conditions that uses tags.

Request

- **TriggerNames** – *Required:* An array of UTF-8 strings.
A list of trigger names, which may be the names returned from the `ListTriggers` operation.

Response

- **Triggers** – An array of [Trigger \(p. 561\)](#) objects.
A list of trigger definitions.
- **TriggersNotFound** – An array of UTF-8 strings.
A list of names of triggers not found.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`

Workflows

The Workflows API describes the data types and API related to creating, updating, or viewing workflows in AWS Glue.

Data Types

- [JobNodeDetails Structure \(p. 570\)](#)
- [CrawlerNodeDetails Structure \(p. 570\)](#)
- [TriggerNodeDetails Structure \(p. 570\)](#)
- [Crawl Structure \(p. 571\)](#)
- [Node Structure \(p. 571\)](#)
- [Edge Structure \(p. 572\)](#)
- [WorkflowGraph Structure \(p. 572\)](#)
- [WorkflowRun Structure \(p. 572\)](#)
- [WorkflowRunStatistics Structure \(p. 573\)](#)
- [Workflow Structure \(p. 573\)](#)

JobNodeDetails Structure

The details of a Job node present in the workflow.

Fields

- [JobRuns – An array of \[JobRun \\(p. 551\\)\]\(#\) objects.](#)

The information for the job runs represented by the job node.

CrawlerNodeDetails Structure

The details of a Crawler node present in the workflow.

Fields

- [Crawls – An array of \[Crawl \\(p. 571\\)\]\(#\) objects.](#)

A list of crawls represented by the crawl node.

TriggerNodeDetails Structure

The details of a Trigger node present in the workflow.

Fields

- [Trigger – A \[Trigger \\(p. 561\\)\]\(#\) object.](#)

The information of the trigger represented by the trigger node.

Crawl Structure

The details of a crawl in the workflow.

Fields

- **State** – UTF-8 string (valid values: RUNNING | SUCCEEDED | CANCELLED | FAILED).
The state of the crawler.
- **StartedOn** – Timestamp.
The date and time on which the crawl started.
- **CompletedOn** – Timestamp.
The date and time on which the crawl completed.
- **ErrorMessage** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).
The error message associated with the crawl.
- **LogGroup** – UTF-8 string, not less than 1 or more than 512 bytes long, matching the [Log group string pattern \(p. 618\)](#).
The log group associated with the crawl.
- **LogStream** – UTF-8 string, not less than 1 or more than 512 bytes long, matching the [Log-stream string pattern \(p. 618\)](#).
The log stream associated with the crawl.

Node Structure

A node represents an AWS Glue component like Trigger, Job etc. which is part of a workflow.

Fields

- **Type** – UTF-8 string (valid values: CRAWLER | JOB | TRIGGER).
The type of AWS Glue component represented by the node.
- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the AWS Glue component represented by the node.
- **UniqueId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The unique Id assigned to the node within the workflow.
- **TriggerDetails** – A [TriggerNodeDetails \(p. 570\)](#) object.
Details of the Trigger when the node represents a Trigger.
- **JobDetails** – A [JobNodeDetails \(p. 570\)](#) object.
Details of the Job when the node represents a Job.
- **CrawlerDetails** – A [CrawlerNodeDetails \(p. 570\)](#) object.
Details of the crawler when the node represents a crawler.

Edge Structure

An edge represents a directed connection between two AWS Glue components which are part of the workflow the edge belongs to.

Fields

- **SourceId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique of the node within the workflow where the edge starts.

- **DestinationId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique of the node within the workflow where the edge ends.

WorkflowGraph Structure

A workflow graph represents the complete workflow containing all the AWS Glue components present in the workflow and all the directed connections between them.

Fields

- **Nodes** – An array of [Node \(p. 571\)](#) objects.

A list of the the AWS Glue components belong to the workflow represented as nodes.

- **Edges** – An array of [Edge \(p. 572\)](#) objects.

A list of all the directed connections between the nodes belonging to the workflow.

WorkflowRun Structure

A workflow run is an execution of a workflow providing all the runtime information.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow which was executed.

- **WorkflowRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of this workflow run.

- **WorkflowRunProperties** – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

The workflow run properties which were set during the run.

- **StartedOn** – Timestamp.

The date and time when the workflow run was started.

- **CompletedOn** – Timestamp.
The date and time when the workflow run completed.
- **Status** – UTF-8 string (valid values: RUNNING | COMPLETED).
The status of the workflow run.
- **Statistics** – A [WorkflowRunStatistics \(p. 573\)](#) object.
The statistics of the run.
- **Graph** – A [WorkflowGraph \(p. 572\)](#) object.
The graph representing all the AWS Glue components that belong to the workflow as nodes and directed connections between them as edges.

WorkflowRunStatistics Structure

Workflow run statistics provides statistics about the workflow run.

Fields

- **TotalActions** – Number (integer).
Total number of Actions in the workflow run.
- **TimeoutActions** – Number (integer).
Total number of Actions which timed out.
- **FailedActions** – Number (integer).
Total number of Actions which have failed.
- **StoppedActions** – Number (integer).
Total number of Actions which have stopped.
- **SucceededActions** – Number (integer).
Total number of Actions which have succeeded.
- **RunningActions** – Number (integer).
Total number Actions in running state.

Workflow Structure

A workflow represents a flow in which AWS Glue components should be executed to complete a logical task.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The name of the workflow representing the flow.
- **Description** – UTF-8 string.
A description of the workflow.
- **DefaultRunProperties** – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

A collection of properties to be used as part of each execution of the workflow.

- `CreatedOn` – Timestamp.

The date and time when the workflow was created.

- `LastModifiedOn` – Timestamp.

The date and time when the workflow was last modified.

- `LastRun` – A [WorkflowRun \(p. 572\)](#) object.

The information about the last execution of the workflow.

- `Graph` – A [WorkflowGraph \(p. 572\)](#) object.

The graph representing all the AWS Glue components that belong to the workflow as nodes and directed connections between them as edges.

- `CreationStatus` – UTF-8 string (valid values: CREATING | CREATED | CREATION_FAILED).

The creation status of the workflow.

Operations

- [CreateWorkflow Action \(Python: create_workflow\) \(p. 574\)](#)
- [UpdateWorkflow Action \(Python: update_workflow\) \(p. 575\)](#)
- [DeleteWorkflow Action \(Python: delete_workflow\) \(p. 576\)](#)
- [ListWorkflows Action \(Python: list_workflows\) \(p. 576\)](#)
- [BatchGetWorkflows Action \(Python: batch_get_workflows\) \(p. 577\)](#)
- [GetWorkflowRun Action \(Python: get_workflow_run\) \(p. 577\)](#)
- [GetWorkflowRuns Action \(Python: get_workflow_runs\) \(p. 578\)](#)
- [GetWorkflowRunProperties Action \(Python: get_workflow_run_properties\) \(p. 579\)](#)
- [PutWorkflowRunProperties Action \(Python: put_workflow_run_properties\) \(p. 579\)](#)

CreateWorkflow Action (Python: create_workflow)

Creates a new workflow.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name to be assigned to the workflow. It should be unique within your account.

- `Description` – UTF-8 string.

A description of the workflow.

- `DefaultRunProperties` – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

A collection of properties to be used as part of each execution of the workflow.

- **Tags** – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The tags to be used with this workflow.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the workflow which was provided as part of the request.

Errors

- `AlreadyExistsException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

UpdateWorkflow Action (Python: update_workflow)

Updates an existing workflow.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow to be updated.

- **Description** – UTF-8 string.

The description of the workflow.

- **DefaultRunProperties** – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

A collection of properties to be used as part of each execution of the workflow.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the workflow which was specified in input.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteWorkflow Action (Python: delete_workflow)

Deletes a workflow.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow to be deleted.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow specified in input.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

ListWorkflows Action (Python: list_workflows)

Lists names of workflows created in the account.

Request

- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation request.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum size of a list to return.

Response

- `Workflows` – An array of UTF-8 strings, not less than 1 or more than 25 strings.

List of names of workflows in the account.

- **NextToken** – UTF-8 string.

A continuation token, if not all workflow names have been returned.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetWorkflows Action (Python: batch_get_workflows)

Returns a list of resource metadata for a given list of workflow names. After calling the `ListWorkflows` operation, you can call this operation to access the data to which you have been granted permissions. This operation supports all IAM permissions, including permission conditions that uses tags.

Request

- **Names** – *Required*: An array of UTF-8 strings, not less than 1 or more than 25 strings.
A list of workflow names, which may be the names returned from the `ListWorkflows` operation.
- **IncludeGraph** – Boolean.
Specifies whether to include a graph when returning the workflow resource metadata.

Response

- **Workflows** – An array of [Workflow \(p. 573\)](#) objects, not less than 1 or more than 25 structures.
A list of workflow resource metadata.
- **MissingWorkflows** – An array of UTF-8 strings, not less than 1 or more than 25 strings.
A list of names of workflows not found.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetWorkflowRun Action (Python: get_workflow_run)

Retrieves the metadata for a given workflow run.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow being run.

- **RunId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the workflow run.

- **IncludeGraph** – Boolean.

Specifies whether to include the workflow graph in response or not.

Response

- **Run** – A [WorkflowRun \(p. 572\)](#) object.

The requested workflow run metadata.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetWorkflowRuns Action (Python: `get_workflow_runs`)

Retrieves metadata for all runs of a given workflow.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow whose metadata of runs should be returned.

- **IncludeGraph** – Boolean.

Specifies whether to include the workflow graph in response or not.

- **NextToken** – UTF-8 string.

The maximum size of the response.

- **MaxResults** – Number (integer), not less than 1 or more than 1000.

The maximum number of workflow runs to be included in the response.

Response

- **Runs** – An array of [WorkflowRun \(p. 572\)](#) objects, not less than 1 or more than 1000 structures.

A list of workflow run metadata objects.

- **NextToken** – UTF-8 string.

A continuation token, if not all requested workflow runs have been returned.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetWorkflowRunProperties Action (Python: get_workflow_run_properties)

Retrieves the workflow run properties which were set during the run.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
Name of the workflow which was run.
- `RunId` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the workflow run whose run properties should be returned.

Response

- `RunProperties` – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

The workflow run properties which were set during the specified run.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

PutWorkflowRunProperties Action (Python: put_workflow_run_properties)

Puts the specified workflow run properties for the given workflow run. If a property already exists for the specified run, then it overrides the value otherwise adds the property to existing properties.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Name of the workflow which was run.

- **RunId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The ID of the workflow run for which the run properties should be updated.

- **RunProperties** – *Required*: A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

Each value is a UTF-8 string.

The properties to put for the specified run.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

Development Endpoints API

The Development Endpoints API describes the AWS Glue API related to testing using a custom DevEndpoint.

Data Types

- [DevEndpoint Structure \(p. 580\)](#)
- [DevEndpointCustomLibraries Structure \(p. 583\)](#)

DevEndpoint Structure

A development endpoint where a developer can remotely debug extract, transform, and load (ETL) scripts.

Fields

- **EndpointName** – UTF-8 string.
The name of the DevEndpoint.
- **RoleArn** – UTF-8 string, matching the [AWS IAM ARN string pattern \(p. 618\)](#).
The Amazon Resource Name (ARN) of the IAM role used in this DevEndpoint.

- **SecurityGroupIds** – An array of UTF-8 strings.
A list of security group identifiers used in this DevEndpoint.
 - **SubnetId** – UTF-8 string.
The subnet ID for this DevEndpoint.
 - **YarnEndpointAddress** – UTF-8 string.
The YARN endpoint address used by this DevEndpoint.
 - **PrivateAddress** – UTF-8 string.
A private IP address to access the DevEndpoint within a VPC if the DevEndpoint is created within one. The PrivateAddress field is present only when you create the DevEndpoint within your VPC.
 - **ZeppelinRemoteSparkInterpreterPort** – Number (integer).
The Apache Zeppelin port for the remote Apache Spark interpreter.
 - **PublicAddress** – UTF-8 string.
The public IP address used by this DevEndpoint. The PublicAddress field is present only when you create a non-virtual private cloud (VPC) DevEndpoint.
 - **Status** – UTF-8 string.
The current status of this DevEndpoint.
 - **WorkerType** – UTF-8 string (valid values: Standard="" | G.1X="" | G.2X="").
The type of predefined worker that is allocated to the development endpoint. Accepts a value of Standard, G.1X, or G.2X.
 - For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
 - For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
 - For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- Known issue: when a development endpoint is created with the G.2X WorkerType configuration, the Spark drivers for the development endpoint will run on 4 vCPU, 16 GB of memory, and a 64 GB disk.
- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).
- Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for running your ETL scripts on development endpoints.
- For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.
- Development endpoints that are created without specifying a Glue version default to Glue 0.9.
- You can specify a version of Python support for development endpoints by using the Arguments parameter in the `CreateDevEndpoint` or `UpdateDevEndpoint` APIs. If no arguments are provided, the version defaults to Python 2.
- **NumberOfWorkers** – Number (integer).
The number of workers of a defined workerType that are allocated to the development endpoint.
- The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- **NumberOfNodes** – Number (integer).

The number of AWS Glue Data Processing Units (DPUs) allocated to this DevEndpoint.

- **AvailabilityZone** – UTF-8 string.

The AWS Availability Zone where this DevEndpoint is located.

- **VpcId** – UTF-8 string.

The ID of the virtual private cloud (VPC) used by this DevEndpoint.

- **ExtraPythonLibsS3Path** – UTF-8 string.

The paths to one or more Python libraries in an Amazon S3 bucket that should be loaded in your DevEndpoint. Multiple values must be complete paths separated by a comma.

Note

You can only use pure Python libraries with a DevEndpoint. Libraries that rely on C extensions, such as the [pandas](#) Python data analysis library, are not currently supported.

- **ExtraJarsS3Path** – UTF-8 string.

The path to one or more Java .jar files in an S3 bucket that should be loaded in your DevEndpoint.

Note

You can only use pure Java/Scala libraries with a DevEndpoint.

- **FailureReason** – UTF-8 string.

The reason for a current failure in this DevEndpoint.

- **LastUpdateStatus** – UTF-8 string.

The status of the last update.

- **CreatedTimestamp** – Timestamp.

The point in time at which this DevEndpoint was created.

- **LastModifiedTimestamp** – Timestamp.

The point in time at which this DevEndpoint was last modified.

- **PublicKey** – UTF-8 string.

The public key to be used by this DevEndpoint for authentication. This attribute is provided for backward compatibility because the recommended attribute to use is public keys.

- **PublicKeys** – An array of UTF-8 strings, not more than 5 strings.

A list of public keys to be used by the DevEndpoints for authentication. Using this attribute is preferred over a single public key because the public keys allow you to have a different private key per client.

Note

If you previously created an endpoint with a public key, you must remove that key to be able to set a list of public keys. Call the `UpdateDevEndpoint` API operation with the public key content in the `deletePublicKeys` attribute, and the list of new keys in the `addPublicKeys` attribute.

- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this DevEndpoint.

- **Arguments** – A map array of key-value pairs, not more than 100 pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

A map of arguments used to configure the DevEndpoint.

Valid arguments are:

- `--enable-glue-datacatalog": ""`

You can specify a version of Python support for development endpoints by using the `Arguments` parameter in the `CreateDevEndpoint` or `UpdateDevEndpoint` APIs. If no arguments are provided, the version defaults to Python 2.

DevEndpointCustomLibraries Structure

Custom libraries to be loaded into a development endpoint.

Fields

- `ExtraPythonLibsS3Path` – UTF-8 string.

The paths to one or more Python libraries in an Amazon Simple Storage Service (Amazon S3) bucket that should be loaded in your DevEndpoint. Multiple values must be complete paths separated by a comma.

Note

You can only use pure Python libraries with a DevEndpoint. Libraries that rely on C extensions, such as the `pandas` Python data analysis library, are not currently supported.

- `ExtraJarsS3Path` – UTF-8 string.

The path to one or more Java `.jar` files in an S3 bucket that should be loaded in your DevEndpoint.

Note

You can only use pure Java/Scala libraries with a DevEndpoint.

Operations

- [CreateDevEndpoint Action \(Python: create_dev_endpoint\) \(p. 583\)](#)
- [UpdateDevEndpoint Action \(Python: update_dev_endpoint\) \(p. 587\)](#)
- [DeleteDevEndpoint Action \(Python: delete_dev_endpoint\) \(p. 588\)](#)
- [GetDevEndpoint Action \(Python: get_dev_endpoint\) \(p. 588\)](#)
- [GetDevEndpoints Action \(Python: get_dev_endpoints\) \(p. 589\)](#)
- [BatchGetEndpoints Action \(Python: batch_get_dev_endpoints\) \(p. 590\)](#)
- [ListEndpoints Action \(Python: list_endpoints\) \(p. 590\)](#)

CreateDevEndpoint Action (Python: create_dev_endpoint)

Creates a new development endpoint.

Request

- `EndpointName` – *Required*: UTF-8 string.

The name to be assigned to the new DevEndpoint.

- **RoleArn** – *Required:* UTF-8 string, matching the [AWS IAM ARN string pattern \(p. 618\)](#).

The IAM role for the DevEndpoint.

- **SecurityGroupIds** – An array of UTF-8 strings.

Security group IDs for the security groups to be used by the new DevEndpoint.

- **SubnetId** – UTF-8 string.

The subnet ID for the new DevEndpoint to use.

- **PublicKey** – UTF-8 string.

The public key to be used by this DevEndpoint for authentication. This attribute is provided for backward compatibility because the recommended attribute to use is publicKeys.

- **PublicKeys** – An array of UTF-8 strings, not more than 5 strings.

A list of public keys to be used by the development endpoints for authentication. The use of this attribute is preferred over a single public key because the public keys allow you to have a different private key per client.

Note

If you previously created an endpoint with a public key, you must remove that key to be able to set a list of public keys. Call the `UpdateDevEndpoint` API with the public key content in the `deletePublicKeys` attribute, and the list of new keys in the `addPublicKeys` attribute.

- **NumberOfNodes** – Number (integer).

The number of AWS Glue Data Processing Units (DPUs) to allocate to this DevEndpoint.

- **WorkerType** – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated to the development endpoint. Accepts a value of Standard, G.1X, or G.2X.

- For the Standard worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the G.1X worker type, each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.
- For the G.2X worker type, each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. We recommend this worker type for memory-intensive jobs.

Known issue: when a development endpoint is created with the G.2X WorkerType configuration, the Spark drivers for the development endpoint will run on 4 vCPU, 16 GB of memory, and a 64 GB disk.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for running your ETL scripts on development endpoints.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

Development endpoints that are created without specifying a Glue version default to Glue 0.9.

You can specify a version of Python support for development endpoints by using the `Arguments` parameter in the `CreateDevEndpoint` or `UpdateDevEndpoint` APIs. If no arguments are provided, the version defaults to Python 2.

- **NumberOfWorkers** – Number (integer).

The number of workers of a defined `workerType` that are allocated to the development endpoint.

The maximum number of workers you can define are 299 for G.1X, and 149 for G.2X.

- `ExtraPythonLibsS3Path` – UTF-8 string.

The paths to one or more Python libraries in an Amazon S3 bucket that should be loaded in your `DevEndpoint`. Multiple values must be complete paths separated by a comma.

Note

You can only use pure Python libraries with a `DevEndpoint`. Libraries that rely on C extensions, such as the [pandas](#) Python data analysis library, are not yet supported.

- `ExtraJarsS3Path` – UTF-8 string.

The path to one or more Java .jar files in an S3 bucket that should be loaded in your `DevEndpoint`.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the `SecurityConfiguration` structure to be used with this `DevEndpoint`.

- `Tags` – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The tags to use with this `DevEndpoint`. You may use tags to limit access to the `DevEndpoint`. For more information about tags in AWS Glue, see [AWS Tags in AWS Glue](#) in the developer guide.

- `Arguments` – A map array of key-value pairs, not more than 100 pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

A map of arguments used to configure the `DevEndpoint`.

Response

- `EndpointName` – UTF-8 string.

The name assigned to the new `DevEndpoint`.

- `Status` – UTF-8 string.

The current status of the new `DevEndpoint`.

- `SecurityGroupIds` – An array of UTF-8 strings.

The security groups assigned to the new `DevEndpoint`.

- `SubnetId` – UTF-8 string.

The subnet ID assigned to the new `DevEndpoint`.

- `RoleArn` – UTF-8 string, matching the [AWS IAM ARN string pattern \(p. 618\)](#).

The Amazon Resource Name (ARN) of the role assigned to the new `DevEndpoint`.

- `YarnEndpointAddress` – UTF-8 string.

The address of the YARN endpoint used by this `DevEndpoint`.

- `ZeppelinRemoteSparkInterpreterPort` – Number (integer).

The Apache Zeppelin port for the remote Apache Spark interpreter.

- **NumberOfNodes** – Number (integer).

The number of AWS Glue Data Processing Units (DPUs) allocated to this DevEndpoint.

- **WorkerType** – UTF-8 string (valid values: Standard="" | G.1X="" | G.2X="").

The type of predefined worker that is allocated to the development endpoint. May be a value of Standard, G.1X, or G.2X.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for running your ETL scripts on development endpoints.

For more information about the available AWS Glue versions and corresponding Spark and Python versions, see [Glue version](#) in the developer guide.

- **NumberOfWorkers** – Number (integer).

The number of workers of a defined **workerType** that are allocated to the development endpoint.

- **AvailabilityZone** – UTF-8 string.

The AWS Availability Zone where this DevEndpoint is located.

- **VpcId** – UTF-8 string.

The ID of the virtual private cloud (VPC) used by this DevEndpoint.

- **ExtraPythonLibsS3Path** – UTF-8 string.

The paths to one or more Python libraries in an S3 bucket that will be loaded in your DevEndpoint.

- **ExtraJarsS3Path** – UTF-8 string.

Path to one or more Java .jar files in an S3 bucket that will be loaded in your DevEndpoint.

- **FailureReason** – UTF-8 string.

The reason for a current failure in this DevEndpoint.

- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the **SecurityConfiguration** structure being used with this DevEndpoint.

- **CreatedTimestamp** – Timestamp.

The point in time at which this DevEndpoint was created.

- **Arguments** – A map array of key-value pairs, not more than 100 pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

The map of arguments used to configure this DevEndpoint.

Valid arguments are:

- "--enable-glue-datacatalog": ""

You can specify a version of Python support for development endpoints by using the **Arguments** parameter in the `CreateDevEndpoint` or `UpdateDevEndpoint` APIs. If no arguments are provided, the version defaults to Python 2.

Errors

- `AccessDeniedException`
- `AlreadyExistsException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ValidationException`
- `ResourceNumberLimitExceededException`

UpdateDevEndpoint Action (Python: update_dev_endpoint)

Updates a specified development endpoint.

Request

- `EndpointName` – *Required:* UTF-8 string.
The name of the DevEndpoint to be updated.
- `PublicKey` – UTF-8 string.
The public key for the DevEndpoint to use.
- `AddPublicKeys` – An array of UTF-8 strings, not more than 5 strings.
The list of public keys for the DevEndpoint to use.
- `DeletePublicKeys` – An array of UTF-8 strings, not more than 5 strings.
The list of public keys to be deleted from the DevEndpoint.
- `CustomLibraries` – A [DevEndpointCustomLibraries \(p. 583\)](#) object.
Custom Python or Java libraries to be loaded in the DevEndpoint.
- `UpdateEtlLibraries` – Boolean.
True if the list of custom libraries to be loaded in the development endpoint needs to be updated, or False if otherwise.
- `DeleteArguments` – An array of UTF-8 strings.
The list of argument keys to be deleted from the map of arguments used to configure the DevEndpoint.
- `AddArguments` – A map array of key-value pairs, not more than 100 pairs.
Each key is a UTF-8 string.
Each value is a UTF-8 string.
The map of arguments to add the map of arguments used to configure the DevEndpoint.
Valid arguments are:
 - `--enable-glue-datacatalog": "`

You can specify a version of Python support for development endpoints by using the `Arguments` parameter in the `CreateDevEndpoint` or `UpdateDevEndpoint` APIs. If no arguments are provided, the version defaults to Python 2.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ValidationException`

DeleteDevEndpoint Action (Python: delete_dev_endpoint)

Deletes a specified development endpoint.

Request

- `EndpointName` – *Required:* UTF-8 string.
The name of the DevEndpoint.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoint Action (Python: get_dev_endpoint)

Retrieves information about a specified development endpoint.

Note

When you create a development endpoint in a virtual private cloud (VPC), AWS Glue returns only a private IP address, and the public IP address field is not populated. When you create a non-VPC development endpoint, AWS Glue returns only a public IP address.

Request

- `EndpointName` – *Required:* UTF-8 string.

Name of the DevEndpoint to retrieve information for.

Response

- `DevEndpoint` – A [DevEndpoint \(p. 580\)](#) object.

A DevEndpoint definition.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`

GetDevEndpoints Action (Python: get_dev_endpoints)

Retrieves all the development endpoints in this AWS account.

Note

When you create a development endpoint in a virtual private cloud (VPC), AWS Glue returns only a private IP address and the public IP address field is not populated. When you create a non-VPC development endpoint, AWS Glue returns only a public IP address.

Request

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum size of information to return.

- `NextToken` – UTF-8 string.

A continuation token, if this is a continuation call.

Response

- `DevEndpoints` – An array of [DevEndpoint \(p. 580\)](#) objects.

A list of DevEndpoint definitions.

- `NextToken` – UTF-8 string.

A continuation token, if not all DevEndpoint definitions have yet been returned.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`

BatchGetDevEndpoints Action (Python: batch_get_dev_endpoints)

Returns a list of resource metadata for a given list of development endpoint names. After calling the `ListDevEndpoints` operation, you can call this operation to access the data to which you have been granted permissions. This operation supports all IAM permissions, including permission conditions that uses tags.

Request

- `customerAccountId` – UTF-8 string.
The AWS account ID.
- `DevEndpointNames` – *Required*: An array of UTF-8 strings, not less than 1 or more than 25 strings.
The list of `DevEndpoint` names, which might be the names returned from the `ListDevEndpoint` operation.

Response

- `DevEndpoints` – An array of `DevEndpoint` (p. 580) objects.
A list of `DevEndpoint` definitions.
- `DevEndpointsNotFound` – An array of UTF-8 strings, not less than 1 or more than 25 strings.
A list of `DevEndpoints` not found.

Errors

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`

ListDevEndpoints Action (Python: list_dev_endpoints)

Retrieves the names of all `DevEndpoint` resources in this AWS account, or the resources with the specified tag. This operation allows you to see which resources are available in your account, and their names.

This operation takes the optional `Tags` field, which you can use as a filter on the response so that tagged resources can be retrieved as a group. If you choose to use tags filtering, only resources with the tag are retrieved.

Request

- `NextToken` – UTF-8 string.
A continuation token, if this is a continuation request.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.
The maximum size of a list to return.

- **Tags** – A map array of key-value pairs, not more than 50 pairs.
 - Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.
 - Each value is a UTF-8 string, not more than 256 bytes long.
- Specifies to return only these tagged resources.

Response

- **DevEndpointNames** – An array of UTF-8 strings.

The names of all the DevEndpoints in the account, or the DevEndpoints with the specified tags.
- **NextToken** – UTF-8 string.

A continuation token, if the returned list does not contain the last metric available.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

AWS Glue Machine Learning API

The Machine Learning API describes the machine learning data types, and includes the API for creating, deleting, or updating a transform, or starting a machine learning task run.

Data Types

- [TransformParameters Structure \(p. 592\)](#)
- [EvaluationMetrics Structure \(p. 592\)](#)
- [MLTransform Structure \(p. 592\)](#)
- [FindMatchesParameters Structure \(p. 594\)](#)
- [FindMatchesMetrics Structure \(p. 595\)](#)
- [ConfusionMatrix Structure \(p. 596\)](#)
- [GlueTable Structure \(p. 596\)](#)
- [TaskRun Structure \(p. 597\)](#)
- [TransformFilterCriteria Structure \(p. 597\)](#)
- [TransformSortCriteria Structure \(p. 598\)](#)
- [TaskRunFilterCriteria Structure \(p. 598\)](#)
- [TaskRunSortCriteria Structure \(p. 599\)](#)
- [TaskRunProperties Structure \(p. 599\)](#)
- [FindMatchesTaskRunProperties Structure \(p. 599\)](#)
- [ImportLabelsTaskRunProperties Structure \(p. 600\)](#)
- [ExportLabelsTaskRunProperties Structure \(p. 600\)](#)
- [LabelingSetGenerationTaskRunProperties Structure \(p. 600\)](#)

- SchemaColumn Structure (p. 600)

TransformParameters Structure

The algorithm-specific parameters that are associated with the machine learning transform.

Fields

- **TransformType** – *Required*: UTF-8 string (valid values: FIND_MATCHES).

The type of machine learning transform.

For information about the types of machine learning transforms, see [Creating Machine Learning Transforms](#).

- **FindMatchesParameters** – A [FindMatchesParameters \(p. 594\)](#) object.

The parameters for the find matches algorithm.

EvaluationMetrics Structure

Evaluation metrics provide an estimate of the quality of your machine learning transform.

Fields

- **TransformType** – *Required*: UTF-8 string (valid values: FIND_MATCHES).

The type of machine learning transform.

- **FindMatchesMetrics** – A [FindMatchesMetrics \(p. 595\)](#) object.

The evaluation metrics for the find matches algorithm.

MLTransform Structure

A structure for a machine learning transform.

Fields

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique transform ID that is generated for the machine learning transform. The ID is guaranteed to be unique and does not change.

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A user-defined name for the machine learning transform. Names are not guaranteed unique and can be changed at any time.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A user-defined, long-form description text for the machine learning transform. Descriptions are not guaranteed to be unique and can be changed at any time.

- **Status** – UTF-8 string (valid values: NOT_READY | READY | DELETING).

The current status of the machine learning transform.

- **CreatedOn** – Timestamp.

A timestamp. The time and date that this machine learning transform was created.

- **LastModifiedOn** – Timestamp.

A timestamp. The last point in time when this machine learning transform was modified.

- **InputRecordTables** – An array of [GlueTable \(p. 596\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- **Parameters** – A [TransformParameters \(p. 592\)](#) object.

A [TransformParameters](#) object. You can use parameters to tune (customize) the behavior of the machine learning transform by specifying what data it learns from and your preference on various tradeoffs (such as precision vs. recall, or accuracy vs. cost).

- **EvaluationMetrics** – An [EvaluationMetrics \(p. 592\)](#) object.

An [EvaluationMetrics](#) object. Evaluation metrics provide an estimate of the quality of your machine learning transform.

- **LabelCount** – Number (integer).

A count identifier for the labeling files generated by AWS Glue for this transform. As you create a better transform, you can iteratively download, label, and upload the labeling file.

- **Schema** – An array of [SchemaColumn \(p. 600\)](#) objects, not more than 100 structures.

A map of key-value pairs representing the columns and data types that this transform can run against. Has an upper bound of 100 columns.

- **Role** – UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role with the required permissions. The required permissions include both AWS Glue service role permissions to AWS Glue resources, and Amazon S3 permissions required by the transform.

- This role needs AWS Glue service role permissions to allow access to resources in AWS Glue. See [Attach a Policy to IAM Users That Access AWS Glue](#).
- This role needs permission to your Amazon Simple Storage Service (Amazon S3) sources, targets, temporary directory, scripts, and any libraries used by the task run for this transform.
- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#) in the developer guide.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that are allocated to task runs for this transform. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

MaxCapacity is a mutually exclusive option with **NumberOfWorkers** and **WorkerType**.

- If either **NumberOfWorkers** or **WorkerType** is set, then **MaxCapacity** cannot be set.
- If **MaxCapacity** is set then neither **NumberOfWorkers** or **WorkerType** can be set.
- If **WorkerType** is set, then **NumberOfWorkers** is required (and vice versa).
- **MaxCapacity** and **NumberOfWorkers** must both be at least 1.

When the `WorkerType` field is set to a value other than `Standard`, the `MaxCapacity` field is set automatically and becomes read-only.

- `WorkerType` – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when a task of this transform runs. Accepts a value of `Standard`, `G.1X`, or `G.2X`.

- For the `Standard` worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the `G.1X` worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.
- For the `G.2X` worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.

`MaxCapacity` is a mutually exclusive option with `NumberOfWorkers` and `WorkerType`.

- If either `NumberOfWorkers` or `WorkerType` is set, then `MaxCapacity` cannot be set.
- If `MaxCapacity` is set then neither `NumberOfWorkers` or `WorkerType` can be set.
- If `WorkerType` is set, then `NumberOfWorkers` is required (and vice versa).
- `MaxCapacity` and `NumberOfWorkers` must both be at least 1.

- `NumberOfWorkers` – Number (integer).

The number of workers of a defined `workerType` that are allocated when a task of the transform runs.

If `WorkerType` is set, then `NumberOfWorkers` is required (and vice versa).

- `Timeout` – Number (integer), at least 1.

The timeout in minutes of the machine learning transform.

- `MaxRetries` – Number (integer).

The maximum number of times to retry after an `MLTaskRun` of the machine learning transform fails.

FindMatchesParameters Structure

The parameters to configure the find matches transform.

Fields

- `PrimaryKeyColumnName` – UTF-8 string, not less than 1 or more than 1024 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of a column that uniquely identifies rows in the source table. Used to help identify matching records.

- `PrecisionRecallTradeoff` – Number (double), not more than 1.0.

The value selected when tuning your transform for a balance between precision and recall. A value of 0.5 means no preference; a value of 1.0 means a bias purely for precision, and a value of 0.0 means a bias for recall. Because this is a tradeoff, choosing values close to 1.0 means very low recall, and choosing values close to 0.0 results in very low precision.

The precision metric indicates how often your model is correct when it predicts a match.

The recall metric indicates that for an actual match, how often your model predicts the match.

- `AccuracyCostTradeoff` – Number (double), not more than 1.0.

The value that is selected when tuning your transform for a balance between accuracy and cost. A value of 0.5 means that the system balances accuracy and cost concerns. A value of 1.0 means a bias purely for accuracy, which typically results in a higher cost, sometimes substantially higher. A value of 0.0 means a bias purely for cost, which results in a less accurate `FindMatches` transform, sometimes with unacceptable accuracy.

Accuracy measures how well the transform finds true positives and true negatives. Increasing accuracy requires more machine resources and cost. But it also results in increased recall.

Cost measures how many compute resources, and thus money, are consumed to run the transform.

- `EnforceProvidedLabels` – Boolean.

The value to switch on or off to force the output to match the provided labels from users. If the value is `True`, the `find matches` transform forces the output to match the provided labels. The results override the normal conflation results. If the value is `False`, the `find matches` transform does not ensure all the labels provided are respected, and the results rely on the trained model.

Note that setting this value to true may increase the conflation execution time.

FindMatchesMetrics Structure

The evaluation metrics for the find matches algorithm. The quality of your machine learning transform is measured by getting your transform to predict some matches and comparing the results to known matches from the same dataset. The quality metrics are based on a subset of your data, so they are not precise.

Fields

- `AreaUnderPRCurve` – Number (double), not more than 1.0.

The area under the precision/recall curve (AUPRC) is a single number measuring the overall quality of the transform, that is independent of the choice made for precision vs. recall. Higher values indicate that you have a more attractive precision vs. recall tradeoff.

For more information, see [Precision and recall](#) in Wikipedia.

- `Precision` – Number (double), not more than 1.0.

The precision metric indicates when often your transform is correct when it predicts a match. Specifically, it measures how well the transform finds true positives from the total true positives possible.

For more information, see [Precision and recall](#) in Wikipedia.

- `Recall` – Number (double), not more than 1.0.

The recall metric indicates that for an actual match, how often your transform predicts the match. Specifically, it measures how well the transform finds true positives from the total records in the source data.

For more information, see [Precision and recall](#) in Wikipedia.

- `F1` – Number (double), not more than 1.0.

The maximum F1 metric indicates the transform's accuracy between 0 and 1, where 1 is the best accuracy.

For more information, see [F1 score](#) in Wikipedia.

- `ConfusionMatrix` – A [ConfusionMatrix \(p. 596\)](#) object.

The confusion matrix shows you what your transform is predicting accurately and what types of errors it is making.

For more information, see [Confusion matrix](#) in Wikipedia.

ConfusionMatrix Structure

The confusion matrix shows you what your transform is predicting accurately and what types of errors it is making.

For more information, see [Confusion matrix](#) in Wikipedia.

Fields

- `NumTruePositives` – Number (long).

The number of matches in the data that the transform correctly found, in the confusion matrix for your transform.

- `NumFalsePositives` – Number (long).

The number of nonmatches in the data that the transform incorrectly classified as a match, in the confusion matrix for your transform.

- `NumTrueNegatives` – Number (long).

The number of nonmatches in the data that the transform correctly rejected, in the confusion matrix for your transform.

- `NumFalseNegatives` – Number (long).

The number of matches in the data that the transform didn't find, in the confusion matrix for your transform.

GlueTable Structure

The database and table in the AWS Glue Data Catalog that is used for input or output data.

Fields

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A database name in the AWS Glue Data Catalog.

- `TableName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A table name in the AWS Glue Data Catalog.

- `CatalogId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A unique identifier for the AWS Glue Data Catalog.

- `ConnectionName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the connection to the AWS Glue Data Catalog.

TaskRun Structure

The sampling parameters that are associated with the machine learning transform.

Fields

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The unique identifier for the transform.
- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The unique identifier for this task run.
- **Status** – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).
The current status of the requested task run.
- **LogGroupName** – UTF-8 string.
The names of the log group for secure logging, associated with this task run.
- **Properties** – A [TaskRunProperties \(p. 599\)](#) object.
Specifies configuration properties associated with this task run.
- **ErrorMessage** – UTF-8 string.
The list of error strings associated with this task run.
- **StartedOn** – Timestamp.
The date and time that this task run started.
- **LastModifiedOn** – Timestamp.
The last point in time that the requested task run was updated.
- **CompletedOn** – Timestamp.
The last point in time that the requested task run was completed.
- **ExecutionTime** – Number (integer).
The amount of time (in seconds) that the task run consumed resources.

TransformFilterCriteria Structure

The criteria used to filter the machine learning transforms.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
A unique transform name that is used to filter the machine learning transforms.
- **TransformType** – UTF-8 string (valid values: FIND_MATCHES).
The type of machine learning transform that is used to filter the machine learning transforms.
- **Status** – UTF-8 string (valid values: NOT_READY | READY | DELETING).

Filters the list of machine learning transforms by the last known status of the transforms (to indicate whether a transform can be used or not). One of "NOT_READY", "READY", or "DELETING".

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#) in the developer guide.

- **CreatedBefore** – Timestamp.

The time and date before which the transforms were created.

- **CreatedAfter** – Timestamp.

The time and date after which the transforms were created.

- **LastModifiedBefore** – Timestamp.

Filter on transforms last modified before this date.

- **LastModifiedAfter** – Timestamp.

Filter on transforms last modified after this date.

- **Schema** – An array of [SchemaColumn \(p. 600\)](#) objects, not more than 100 structures.

Filters on datasets with a specific schema. The `Map<Column, Type>` object is an array of key-value pairs representing the schema this transform accepts, where `Column` is the name of a column, and `Type` is the type of the data such as an integer or string. Has an upper bound of 100 columns.

TransformSortCriteria Structure

The sorting criteria that are associated with the machine learning transform.

Fields

- **Column** – *Required*: UTF-8 string (valid values: NAME | TRANSFORM_TYPE | STATUS | CREATED | LAST_MODIFIED).

The column to be used in the sorting criteria that are associated with the machine learning transform.

- **SortDirection** – *Required*: UTF-8 string (valid values: DESCENDING | ASCENDING).

The sort direction to be used in the sorting criteria that are associated with the machine learning transform.

TaskRunFilterCriteria Structure

The criteria that are used to filter the task runs for the machine learning transform.

Fields

- **TaskRunType** – UTF-8 string (valid values: EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES).

The type of task run.

- **Status** – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The current status of the task run.

- `StartedBefore` – Timestamp.

Filter on task runs started before this date.

- `StartedAfter` – Timestamp.

Filter on task runs started after this date.

TaskRunSortCriteria Structure

The sorting criteria that are used to sort the list of task runs for the machine learning transform.

Fields

- `Column` – *Required*: UTF-8 string (valid values: `TASK_RUN_TYPE` | `STATUS` | `STARTED`).

The column to be used to sort the list of task runs for the machine learning transform.

- `SortDirection` – *Required*: UTF-8 string (valid values: `DESCENDING` | `ASCENDING`).

The sort direction to be used to sort the list of task runs for the machine learning transform.

TaskRunProperties Structure

The configuration properties for the task run.

Fields

- `TaskType` – UTF-8 string (valid values: `EVALUATION` | `LABELING_SET_GENERATION` | `IMPORT_LABELS` | `EXPORT_LABELS` | `FIND_MATCHES`).

The type of task run.

- `ImportLabelsTaskRunProperties` – An [ImportLabelsTaskRunProperties \(p. 600\)](#) object.

The configuration properties for an importing labels task run.

- `ExportLabelsTaskRunProperties` – An [ExportLabelsTaskRunProperties \(p. 600\)](#) object.

The configuration properties for an exporting labels task run.

- `LabelingSetGenerationTaskRunProperties` – A [LabelingSetGenerationTaskRunProperties \(p. 600\)](#) object.

The configuration properties for a labeling set generation task run.

- `FindMatchesTaskRunProperties` – A [FindMatchesTaskRunProperties \(p. 599\)](#) object.

The configuration properties for a find matches task run.

FindMatchesTaskRunProperties Structure

Specifies configuration properties for a Find Matches task run.

Fields

- `JobId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The job ID for the Find Matches task run.

- `JobName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name assigned to the job for the Find Matches task run.

- `JobRunId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The job run ID for the Find Matches task run.

ImportLabelsTaskRunProperties Structure

Specifies configuration properties for an importing labels task run.

Fields

- `InputS3Path` – UTF-8 string.

The Amazon Simple Storage Service (Amazon S3) path from where you will import the labels.

- `Replace` – Boolean.

Indicates whether to overwrite your existing labels.

ExportLabelsTaskRunProperties Structure

Specifies configuration properties for an exporting labels task run.

Fields

- `OutputS3Path` – UTF-8 string.

The Amazon Simple Storage Service (Amazon S3) path where you will export the labels.

LabelingSetGenerationTaskRunProperties Structure

Specifies configuration properties for a labeling set generation task run.

Fields

- `OutputS3Path` – UTF-8 string.

The Amazon Simple Storage Service (Amazon S3) path where you will generate the labeling set.

SchemaColumn Structure

A key-value pair representing a column and data type that this transform can run against. The `Schema` parameter of the `MLTransform` may contain up to 100 of these structures.

Fields

- `Name` – UTF-8 string, not less than 1 or more than 1024 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The name of the column.

- **DataType** – UTF-8 string, not more than 131072 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The type of data in the column.

Operations

- [CreateMLTransform Action \(Python: create_ml_transform\) \(p. 601\)](#)
- [UpdateMLTransform Action \(Python: update_ml_transform\) \(p. 603\)](#)
- [DeleteMLTransform Action \(Python: delete_ml_transform\) \(p. 605\)](#)
- [GetMLTransform Action \(Python: get_ml_transform\) \(p. 605\)](#)
- [GetMLTransforms Action \(Python: get_ml_transforms\) \(p. 607\)](#)
- [StartMLEvaluationTaskRun Action \(Python: start_ml_evaluation_task_run\) \(p. 608\)](#)
- [StartMLLabelingSetGenerationTaskRun Action \(Python: start_ml_labeling_set_generation_task_run\) \(p. 609\)](#)
- [GetMLTaskRun Action \(Python: get_ml_task_run\) \(p. 609\)](#)
- [GetMLTaskRuns Action \(Python: get_ml_task_runs\) \(p. 611\)](#)
- [CancelMLTaskRun Action \(Python: cancel_ml_task_run\) \(p. 611\)](#)
- [StartExportLabelsTaskRun Action \(Python: start_export_labels_task_run\) \(p. 612\)](#)
- [StartImportLabelsTaskRun Action \(Python: start_import_labels_task_run\) \(p. 613\)](#)

CreateMLTransform Action (Python: create_ml_transform)

Creates an AWS Glue machine learning transform. This operation creates the transform and all the necessary parameters to train it.

Call this operation as the first step in the process of using a machine learning transform (such as the [FindMatches transform](#)) for deduplicating data. You can provide an optional [Description](#), in addition to the parameters that you want to use for your algorithm.

You must also specify certain parameters for the tasks that AWS Glue runs on your behalf as part of learning from your data and creating a high-quality machine learning transform. These parameters include [Role](#), and optionally, [AllocatedCapacity](#), [Timeout](#), and [MaxRetries](#). For more information, see [Jobs](#).

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique name that you give the transform when you create it.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the machine learning transform that is being defined. The default is an empty string.

- **InputRecordTables** – *Required*: An array of [GlueTable \(p. 596\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- **Parameters** – *Required:* A [TransformParameters \(p. 592\)](#) object.

The algorithmic parameters that are specific to the transform type used. Conditionally dependent on the transform type.

- **Role** – *Required:* UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role with the required permissions. The required permissions include both AWS Glue service role permissions to AWS Glue resources, and Amazon S3 permissions required by the transform.

- This role needs AWS Glue service role permissions to allow access to resources in AWS Glue. See [Attach a Policy to IAM Users That Access AWS Glue](#).
- This role needs permission to your Amazon Simple Storage Service (Amazon S3) sources, targets, temporary directory, scripts, and any libraries used by the task run for this transform.
- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#) in the developer guide.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that are allocated to task runs for this transform. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

MaxCapacity is a mutually exclusive option with **NumberOfWorkers** and **WorkerType**.

- If either **NumberOfWorkers** or **WorkerType** is set, then **MaxCapacity** cannot be set.
- If **MaxCapacity** is set then neither **NumberOfWorkers** or **WorkerType** can be set.
- If **WorkerType** is set, then **NumberOfWorkers** is required (and vice versa).
- **MaxCapacity** and **NumberOfWorkers** must both be at least 1.

When the **WorkerType** field is set to a value other than **Standard**, the **MaxCapacity** field is set automatically and becomes read-only.

When the **WorkerType** field is set to a value other than **Standard**, the **MaxCapacity** field is set automatically and becomes read-only.

- **WorkerType** – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when this task runs. Accepts a value of **Standard**, **G.1X**, or **G.2X**.

- For the **Standard** worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the **G.1X** worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.
- For the **G.2X** worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.

MaxCapacity is a mutually exclusive option with **NumberOfWorkers** and **WorkerType**.

- If either **NumberOfWorkers** or **WorkerType** is set, then **MaxCapacity** cannot be set.
- If **MaxCapacity** is set then neither **NumberOfWorkers** or **WorkerType** can be set.
- If **WorkerType** is set, then **NumberOfWorkers** is required (and vice versa).
- **MaxCapacity** and **NumberOfWorkers** must both be at least 1.

- **NumberOfWorkers** – Number (integer).

The number of workers of a defined `workerType` that are allocated when this task runs.

If `WorkerType` is set, then `NumberOfWorkers` is required (and vice versa).

- `Timeout` – Number (integer), at least 1.

The timeout of the task run for this transform in minutes. This is the maximum time that a task run for this transform can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours).

- `MaxRetries` – Number (integer).

The maximum number of times to retry a task for this transform after a task run fails.

Response

- `TransformId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A unique identifier that is generated for the transform.

Errors

- `AlreadyExistsException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`
- `AccessDeniedException`
- `ResourceNumberLimitExceededException`
- `IdempotentParameterMismatchException`

UpdateMLTransform Action (Python: update_ml_transform)

Updates an existing machine learning transform. Call this operation to tune the algorithm parameters to achieve better results.

After calling this operation, you can call the `StartMLEvaluationTaskRun` operation to assess how well your new parameters achieved your goals (such as improving the quality of your machine learning transform, or making it more cost-effective).

Request

- `TransformId` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A unique identifier that was generated when the transform was created.

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique name that you gave the transform when you created it.

- `Description` – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the transform. The default is an empty string.

- **Parameters** – A [TransformParameters \(p. 592\)](#) object.

The configuration parameters that are specific to the transform type (algorithm) used. Conditionally dependent on the transform type.

- **Role** – UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role with the required permissions.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#) in the developer guide.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that are allocated to task runs for this transform. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

When the `WorkerType` field is set to a value other than `Standard`, the `MaxCapacity` field is set automatically and becomes read-only.

- **WorkerType** – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when this task runs. Accepts a value of `Standard`, `G.1X`, or `G.2X`.

- For the `Standard` worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the `G.1X` worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.
- For the `G.2X` worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.
- **NumberOfWorkers** – Number (integer).

The number of workers of a defined `workerType` that are allocated when this task runs.

- **Timeout** – Number (integer), at least 1.

The timeout for a task run for this transform in minutes. This is the maximum time that a task run for this transform can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours).

- **MaxRetries** – Number (integer).

The maximum number of times to retry a task for this transform after a task run fails.

Response

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier for the transform that was updated.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `AccessDeniedException`

DeleteMLTransform Action (Python: delete_ml_transform)

Deletes an AWS Glue machine learning transform. Machine learning transforms are a special type of transform that use machine learning to learn the details of the transformation to be performed by learning from examples provided by humans. These transformations are then saved by AWS Glue. If you no longer need a transform, you can delete it by calling `DeleteMLTransforms`. However, any AWS Glue jobs that still reference the deleted transform will no longer succeed.

Request

- `TransformId` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the transform to delete.

Response

- `TransformId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the transform that was deleted.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransform Action (Python: get_ml_transform)

Gets an AWS Glue machine learning transform artifact and all its corresponding metadata. Machine learning transforms are a special type of transform that use machine learning to learn the details of the transformation to be performed by learning from examples provided by humans. These transformations are then saved by AWS Glue. You can retrieve their metadata by calling `GetMLTransform`.

Request

- `TransformId` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the transform, generated at the time that the transform was created.

Response

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the transform, generated at the time that the transform was created.

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique name given to the transform when it was created.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A description of the transform.

- **Status** – UTF-8 string (valid values: NOT_READY | READY | DELETING).

The last known status of the transform (to indicate whether it can be used or not). One of "NOT_READY", "READY", or "DELETING".

- **CreatedOn** – Timestamp.

The date and time when the transform was created.

- **LastModifiedOn** – Timestamp.

The date and time when the transform was last modified.

- **InputRecordTables** – An array of [GlueTable \(p. 596\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- **Parameters** – A [TransformParameters \(p. 592\)](#) object.

The configuration parameters that are specific to the algorithm used.

- **EvaluationMetrics** – An [EvaluationMetrics \(p. 592\)](#) object.

The latest evaluation metrics.

- **LabelCount** – Number (integer).

The number of labels available for this transform.

- **Schema** – An array of [SchemaColumn \(p. 600\)](#) objects, not more than 100 structures.

The Map<Column, Type> object that represents the schema that this transform accepts. Has an upper bound of 100 columns.

- **Role** – UTF-8 string.

The name or Amazon Resource Name (ARN) of the IAM role with the required permissions.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 618\)](#).

This value determines which version of AWS Glue this machine learning transform is compatible with. Glue 1.0 is recommended for most customers. If the value is not set, the Glue compatibility defaults to Glue 0.9. For more information, see [AWS Glue Versions](#) in the developer guide.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that are allocated to task runs for this transform. You can allocate from 2 to 100 DPUs; the default is 10. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory. For more information, see the [AWS Glue pricing page](#).

When the `WorkerType` field is set to a value other than `Standard`, the `MaxCapacity` field is set automatically and becomes read-only.

- `WorkerType` – UTF-8 string (valid values: `Standard=""` | `G.1X=""` | `G.2X=""`).

The type of predefined worker that is allocated when this task runs. Accepts a value of `Standard`, `G.1X`, or `G.2X`.

- For the `Standard` worker type, each worker provides 4 vCPU, 16 GB of memory and a 50GB disk, and 2 executors per worker.
- For the `G.1X` worker type, each worker provides 4 vCPU, 16 GB of memory and a 64GB disk, and 1 executor per worker.
- For the `G.2X` worker type, each worker provides 8 vCPU, 32 GB of memory and a 128GB disk, and 1 executor per worker.

- `NumberOfWorkers` – Number (integer).

The number of workers of a defined `workerType` that are allocated when this task runs.

- `Timeout` – Number (integer), at least 1.

The timeout for a task run for this transform in minutes. This is the maximum time that a task run for this transform can consume resources before it is terminated and enters `TIMEOUT` status. The default is 2,880 minutes (48 hours).

- `MaxRetries` – Number (integer).

The maximum number of times to retry a task for this transform after a task run fails.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransforms Action (Python: get_ml_transforms)

Gets a sortable, filterable list of existing AWS Glue machine learning transforms. Machine learning transforms are a special type of transform that use machine learning to learn the details of the transformation to be performed by learning from examples provided by humans. These transformations are then saved by AWS Glue, and you can retrieve their metadata by calling `GetMLTransforms`.

Request

- `NextToken` – UTF-8 string.

A paginated token to offset the results.
- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum number of results to return.
- `Filter` – A [TransformFilterCriteria \(p. 597\)](#) object.

The filter transformation criteria.
- `Sort` – A [TransformSortCriteria \(p. 598\)](#) object.

The sorting criteria.

Response

- **Transforms** – *Required:* An array of [MLTransform \(p. 592\)](#) objects.
A list of machine learning transforms.
- **NextToken** – UTF-8 string.
A pagination token, if more results are available.

Errors

- [EntityNotFoundException](#)
- [InvalidInputException](#)
- [OperationTimeoutException](#)
- [InternalServiceException](#)

StartMLEvaluationTaskRun Action (Python: start_ml_evaluation_task_run)

Starts a task to estimate the quality of the transform.

When you provide label sets as examples of truth, AWS Glue machine learning uses some of those examples to learn from them. The rest of the labels are used as a test to estimate quality.

Returns a unique identifier for the run. You can call [GetMLTaskRun](#) to get more information about the stats of the [EvaluationTaskRun](#).

Request

- **TransformId** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

Response

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier associated with this run.

Errors

- [EntityNotFoundException](#)
- [InvalidInputException](#)
- [OperationTimeoutException](#)
- [InternalServiceException](#)
- [ConcurrentRunsExceededException](#)
- [MLTransformNotReadyException](#)

StartMLLabelingSetGenerationTaskRun Action (Python: start_ml_labeling_set_generation_task_run)

Starts the active learning workflow for your machine learning transform to improve the transform's quality by generating label sets and adding labels.

When the StartMLLabelingSetGenerationTaskRun finishes, AWS Glue will have generated a "labeling set" or a set of questions for humans to answer.

In the case of the FindMatches transform, these questions are of the form, "What is the correct way to group these rows together into groups composed entirely of matching records?"

After the labeling process is finished, you can upload your labels with a call to StartImportLabelsTaskRun. After StartImportLabelsTaskRun finishes, all future runs of the machine learning transform will use the new and improved labels and perform a higher-quality transformation.

Request

- **TransformId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **OutputS3Path** – *Required*: UTF-8 string.

The Amazon Simple Storage Service (Amazon S3) path where you generate the labeling set.

Response

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique run identifier that is associated with this task run.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConcurrentRunsExceededException`

GetMLTaskRun Action (Python: get_ml_task_run)

Gets details for a specific task run on a machine learning transform. Machine learning task runs are asynchronous tasks that AWS Glue runs on your behalf as part of various machine learning workflows. You can check the stats of any task run by calling GetMLTaskRun with the TaskRunID and its parent transform's TransformID.

Request

- **TransformId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **TaskRunId** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the task run.

Response

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the task run.

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique run identifier associated with this run.

- **Status** – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The status for this task run.

- **LogGroupName** – UTF-8 string.

The names of the log groups that are associated with the task run.

- **Properties** – A [TaskRunProperties \(p. 599\)](#) object.

The list of properties that are associated with the task run.

- **ErrorString** – UTF-8 string.

The error strings that are associated with the task run.

- **StartedOn** – Timestamp.

The date and time when this task run started.

- **LastModifiedOn** – Timestamp.

The date and time when this task run was last modified.

- **CompletedOn** – Timestamp.

The date and time when this task run was completed.

- **ExecutionTime** – Number (integer).

The amount of time (in seconds) that the task run consumed resources.

Errors

- **EntityNotFoundException**
- **InvalidInputException**
- **OperationTimeoutException**
- **InternalServiceException**

GetMLTaskRuns Action (Python: get_ml_task_runs)

Gets a list of runs for a machine learning transform. Machine learning task runs are asynchronous tasks that AWS Glue runs on your behalf as part of various machine learning workflows. You can get a sortable, filterable list of machine learning task runs by calling `GetMLTaskRuns` with their parent transform's `TransformID` and other optional parameters as documented in this section.

This operation returns a list of historic runs and must be paginated.

Request

- `TransformID` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- `NextToken` – UTF-8 string.

A token for pagination of the results. The default is empty.

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum number of results to return.

- `Filter` – A [TaskRunFilterCriteria \(p. 598\)](#) object.

The filter criteria, in the `TaskRunFilterCriteria` structure, for the task run.

- `Sort` – A [TaskRunSortCriteria \(p. 599\)](#) object.

The sorting criteria, in the `TaskRunSortCriteria` structure, for the task run.

Response

- `TaskRuns` – An array of [TaskRun \(p. 597\)](#) objects.

A list of task runs that are associated with the transform.

- `NextToken` – UTF-8 string.

A pagination token, if more results are available.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`

CancelMLTaskRun Action (Python: cancel_ml_task_run)

Cancels (stops) a task run. Machine learning task runs are asynchronous tasks that AWS Glue runs on your behalf as part of various machine learning workflows. You can cancel a machine learning task run at any time by calling `CancelMLTaskRun` with a task run's parent transform's `TransformID` and the task run's `TaskRunId`.

Request

- **TransformId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **TaskRunId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

A unique identifier for the task run.

Response

- **TransformId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier for the task run.

- **Status** – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The status for this run.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

StartExportLabelsTaskRun Action (Python: `start_export_labels_task_run`)

Begins an asynchronous task to export all labeled data for a particular transform. This task is the only label-related API call that is not part of the typical active learning workflow. You typically use `StartExportLabelsTaskRun` when you want to work with all of your existing labels at the same time, such as when you want to remove or change labels that were previously submitted as truth. This API operation accepts the `TransformId` whose labels you want to export and an Amazon Simple Storage Service (Amazon S3) path to export the labels to. The operation returns a `TaskRunId`. You can check on the status of your task run by calling the `GetMLTaskRun` API.

Request

- **TransformId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **OutputS3Path** – *Required*: UTF-8 string.

The Amazon S3 path where you export the labels.

Response

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier for the task run.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`

StartImportLabelsTaskRun Action (Python: start_import_labels_task_run)

Enables you to provide additional labels (examples of truth) to be used to teach the machine learning transform and improve its quality. This API operation is generally used as part of the active learning workflow that starts with the `StartMLLabelingSetGenerationTaskRun` call and that ultimately results in improving the quality of your machine learning transform.

After the `StartMLLabelingSetGenerationTaskRun` finishes, AWS Glue machine learning will have generated a series of questions for humans to answer. (Answering these questions is often called 'labeling' in the machine learning workflows). In the case of the `FindMatches` transform, these questions are of the form, "What is the correct way to group these rows together into groups composed entirely of matching records?" After the labeling process is finished, users upload their answers/labels with a call to `StartImportLabelsTaskRun`. After `StartImportLabelsTaskRun` finishes, all future runs of the machine learning transform use the new and improved labels and perform a higher-quality transformation.

By default, `StartMLLabelingSetGenerationTaskRun` continually learns from and combines all labels that you upload unless you set `Replace` to true. If you set `Replace` to true, `StartImportLabelsTaskRun` deletes and forgets all previously uploaded labels and learns only from the exact set that you upload. Replacing labels can be helpful if you realize that you previously uploaded incorrect labels, and you believe that they are having a negative effect on your transform quality.

You can check on the status of your task run by calling the `GetMLTaskRun` operation.

Request

- **TransformId** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier of the machine learning transform.

- **InputS3Path** – *Required:* UTF-8 string.

The Amazon Simple Storage Service (Amazon S3) path from where you import the labels.

- **ReplaceAllLabels** – Boolean.

Indicates whether to overwrite your existing labels.

Response

- **TaskRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The unique identifier for the task run.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`

Tagging APIs in AWS Glue

Data Types

- [Tag Structure \(p. 614\)](#)

Tag Structure

The `Tag` object represents a label that you can assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define.

For more information about tags, and controlling access to resources in AWS Glue, see [AWS Tags in AWS Glue](#) and [Specifying AWS Glue Resource ARNs](#) in the developer guide.

Fields

- **key** – UTF-8 string, not less than 1 or more than 128 bytes long.

The tag key. The key is required when you create a tag on an object. The key is case-sensitive, and must not contain the prefix `aws`.

- **value** – UTF-8 string, not more than 256 bytes long.

The tag value. The value is optional when you create a tag on an object. The value is case-sensitive, and must not contain the prefix `aws`.

Operations

- [TagResource Action \(Python: tag_resource\) \(p. 614\)](#)
- [UntagResource Action \(Python: untag_resource\) \(p. 615\)](#)
- [GetTags Action \(Python: get_tags\) \(p. 616\)](#)

TagResource Action (Python: tag_resource)

Adds tags to a resource. A tag is a label you can assign to an AWS resource. In AWS Glue, you can tag only certain resources. For information about what resources you can tag, see [AWS Tags in AWS Glue](#).

Request

- **ResourceArn** – *Required:* UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [AWS Glue ARN string pattern \(p. 618\)](#).

The ARN of the AWS Glue resource to which to add the tags. For more information about AWS Glue resource ARNs, see the [AWS Glue ARN string pattern](#).

- **TagsToAdd** – *Required:* A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

Tags to add to this resource.

Response

- *No Response parameters.*

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

UntagResource Action (Python: untag_resource)

Removes tags from a resource.

Request

- **ResourceArn** – *Required:* UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [AWS Glue ARN string pattern \(p. 618\)](#).

The Amazon Resource Name (ARN) of the resource from which to remove the tags.

- **TagsToRemove** – *Required:* An array of UTF-8 strings, not more than 50 strings.

Tags to remove from this resource.

Response

- *No Response parameters.*

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetTags Action (Python: get_tags)

Retrieves a list of tags associated with a resource.

Request

- **ResourceArn** – *Required:* UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [AWS Glue ARN string pattern \(p. 618\)](#).

The Amazon Resource Name (ARN) of the resource for which to retrieve tags.

Response

- **Tags** – A map array of key-value pairs, not more than 50 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long.

Each value is a UTF-8 string, not more than 256 bytes long.

The requested tags.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

Common Data Types

The Common Data Types describes miscellaneous common data types in AWS Glue.

Tag Structure

The `Tag` object represents a label that you can assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define.

For more information about tags, and controlling access to resources in AWS Glue, see [AWS Tags in AWS Glue](#) and [Specifying AWS Glue Resource ARNs](#) in the developer guide.

Fields

- **key** – UTF-8 string, not less than 1 or more than 128 bytes long.

The tag key. The key is required when you create a tag on an object. The key is case-sensitive, and must not contain the prefix aws.

- **value** – UTF-8 string, not more than 256 bytes long.

The tag value. The value is optional when you create a tag on an object. The value is case-sensitive, and must not contain the prefix aws.

DecimalNumber Structure

Contains a numeric value in decimal format.

Fields

- **UnscaledValue** – Blob.

The unscaled numeric value.

- **Scale** – Number (integer).

The scale that determines where the decimal point falls in the unscaled value.

ErrorDetail Structure

Contains details about an error.

Fields

- **ErrorCode** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).

The code associated with this error.

- **ErrorMessage** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

A message describing the error.

PropertyPredicate Structure

Defines a property predicate.

Fields

- **Key** – Value string, not more than 1024 bytes long.

The key of the property.

- **Value** – Value string, not more than 1024 bytes long.

The value of the property.

- **Comparator** – UTF-8 string (valid values: EQUALS | GREATER_THAN | LESS_THAN | GREATER_THAN_EQUALS | LESS_THAN_EQUALS).

The comparator used to compare this property to others.

ResourceUri Structure

The URIs for function resources.

Fields

- **ResourceType** – UTF-8 string (valid values: JAR | FILE | ARCHIVE).

The type of the resource.

- **Uri** – Uniform resource identifier (uri), not less than 1 or more than 1024 bytes long, matching the [URI address multi-line string pattern \(p. 618\)](#).

The URI for accessing the resource.

String Patterns

The API uses the following regular expressions to define what is valid content for various string parameters and members:

- Single-line string pattern – "[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF \t]*"
- URI address multi-line string pattern – "[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF \uDFFF\r\n\t]*"
- A Logstash Grok string pattern – "[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF \r\t]*"
- Identifier string pattern – "[A-Za-z_][A-Za-z0-9_]*"
- AWS Glue ARN string pattern – "arn:aws:glue:.*"
- AWS IAM ARN string pattern – "arn:aws:iam::\d{12}:role/.+"
- Version string pattern – "^[a-zA-Z0-9_-]+\$"
- Log group string pattern – "[\._/_/#A-Za-z0-9]+"
- Log-stream string pattern – "[^:]*"
- Custom string pattern #10 – "[^\r\n]"
- Custom string pattern #11 – "^[2-3]\$"
- Custom string pattern #12 – "^\w+\.\w+\.\w+\$"
- Custom string pattern #13 – "^\w+\.\w+\$"
- Custom string pattern #14 – "arn:aws:kms:.*"

Exceptions

This section describes AWS Glue exceptions that you can use to find the source of problems and fix them.

AccessDeniedException Structure

Access to a resource was denied.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

AlreadyExistsException Structure

A resource to be created or added already exists.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

ConcurrentModificationException Structure

Two processes are trying to modify a resource simultaneously.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

ConcurrentRunsExceededException Structure

Too many jobs are being run concurrently.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

CrawlerNotRunningException Structure

The specified crawler is not running.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

CrawlerRunningException Structure

The operation cannot be performed because the crawler is already running.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

CrawlerStoppingException Structure

The specified crawler is stopping.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

EntityNotFoundException Structure

A specified entity does not exist

Fields

- **Message** – UTF-8 string.

A message describing the problem.

GlueEncryptionException Structure

An encryption operation failed.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

IdempotentParameterMismatchException Structure

The same unique identifier was associated with two different records.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

InternalServiceException Structure

An internal service error occurred.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

InvalidExecutionEngineException Structure

An unknown or invalid execution engine was specified.

Fields

- **message** – UTF-8 string.

A message describing the problem.

InvalidInputException Structure

The input provided was not valid.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

InvalidTaskStatusTransitionException Structure

Proper transition from one task to the next failed.

Fields

- **message** – UTF-8 string.
A message describing the problem.

JobDefinitionErrorException Structure

A job definition is not valid.

Fields

- **message** – UTF-8 string.
A message describing the problem.

JobRunInTerminalStateException Structure

The terminal state of a job run signals a failure.

Fields

- **message** – UTF-8 string.
A message describing the problem.

JobRunInvalidStateTransitionException Structure

A job run encountered an invalid transition from source state to target state.

Fields

- **jobRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 618\)](#).
The ID of the job run in question.
- **message** – UTF-8 string.
A message describing the problem.

- `sourceState` – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The source state.

- `targetState` – UTF-8 string (valid values: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

The target state.

JobRunNotInTerminalStateException Structure

A job run is not in a terminal state.

Fields

- `message` – UTF-8 string.

A message describing the problem.

LateRunnerException Structure

A job runner is late.

Fields

- `Message` – UTF-8 string.

A message describing the problem.

NoScheduleException Structure

There is no applicable schedule.

Fields

- `Message` – UTF-8 string.

A message describing the problem.

OperationTimeoutException Structure

The operation timed out.

Fields

- `Message` – UTF-8 string.

A message describing the problem.

ResourceNumberLimitExceededException Structure

A resource numerical limit was exceeded.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

SchedulerNotRunningException Structure

The specified scheduler is not running.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

SchedulerRunningException Structure

The specified scheduler is already running.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

SchedulerTransitioningException Structure

The specified scheduler is transitioning.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

UnrecognizedRunnerException Structure

The job runner was not recognized.

Fields

- **Message** – UTF-8 string.
A message describing the problem.

ValidationException Structure

A value could not be validated.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

VersionMismatchException Structure

There was a version conflict.

Fields

- **Message** – UTF-8 string.

A message describing the problem.

AWS Glue Troubleshooting

Topics

- [Gathering AWS Glue Troubleshooting Information \(p. 625\)](#)
- [Troubleshooting Connection Issues in AWS Glue \(p. 625\)](#)
- [Troubleshooting Errors in AWS Glue \(p. 626\)](#)
- [AWS Glue Quotas \(p. 633\)](#)

Gathering AWS Glue Troubleshooting Information

If you encounter errors or unexpected behavior in AWS Glue and need to contact AWS Support, you should first gather information about names, IDs, and logs that are associated with the failed action. Having this information available enables AWS Support to help you resolve the problems you're experiencing.

Along with your *account ID*, gather the following information for each of these types of failures:

When a crawler fails, gather the following information:

- Crawler name

Logs from crawler runs are located in CloudWatch Logs under `/aws-glue/crawlers`.

When a test connection fails, gather the following information:

- Connection name
- Connection ID
- JDBC connection string in the form `jdbc:protocol://host:port/database-name`.

Logs from test connections are located in CloudWatch Logs under `/aws-glue/testconnection`.

When a job fails, gather the following information:

- Job name
- Job run ID in the form `jr_xxxxx`.

Logs from job runs are located in CloudWatch Logs under `/aws-glue/jobs`.

Troubleshooting Connection Issues in AWS Glue

When an AWS Glue crawler or a job uses connection properties to access a data store, you might encounter errors when you try to connect. AWS Glue uses private IP addresses in the subnet when it creates elastic network interfaces in your specified virtual private cloud (VPC) and subnet. Security groups specified in the connection are applied on each of the elastic network interfaces. Check to see whether security groups allow outbound access and if they allow connectivity to the database cluster.

In addition, Apache Spark requires bi-directional connectivity among driver and executor nodes. One of the security groups needs to allow ingress rules on all TCP ports. You can prevent it from being open to the world by restricting the source of the security group to itself with a self-referencing security group.

Here are some typical actions you can take to troubleshoot connection problems:

- Check the port address of your connection.
- Check the user name and password string in your connection.
- For a JDBC data store, verify that it allows incoming connections.
- Verify that your data store can be accessed within your VPC.

Troubleshooting Errors in AWS Glue

If you encounter errors in AWS Glue, use the following solutions to help you find the source of the problems and fix them.

Note

The AWS Glue GitHub repository contains additional troubleshooting guidance in [AWS Glue Frequently Asked Questions](#).

Topics

- [Error: Resource Unavailable \(p. 626\)](#)
- [Error: Could Not Find S3 Endpoint or NAT Gateway for subnetId in VPC \(p. 627\)](#)
- [Error: Inbound Rule in Security Group Required \(p. 627\)](#)
- [Error: Outbound Rule in Security Group Required \(p. 627\)](#)
- [Error: Job Run Failed Because the Role Passed Should Be Given Assume Role Permissions for the AWS Glue Service \(p. 627\)](#)
- [Error: DescribeVpcEndpoints Action Is Unauthorized. Unable to Validate VPC ID vpc-id \(p. 627\)](#)
- [Error: DescribeRouteTables Action Is Unauthorized. Unable to Validate Subnet Id: subnet-id in VPC id: vpc-id \(p. 628\)](#)
- [Error: Failed to Call ec2:DescribeSubnets \(p. 628\)](#)
- [Error: Failed to Call ec2:DescribeSecurityGroups \(p. 628\)](#)
- [Error: Could Not Find Subnet for AZ \(p. 628\)](#)
- [Error: Job Run Exception When Writing to a JDBC Target \(p. 628\)](#)
- [Error: Amazon S3 Timeout \(p. 629\)](#)
- [Error: Amazon S3 Access Denied \(p. 629\)](#)
- [Error: Amazon S3 Access Key ID Does Not Exist \(p. 629\)](#)
- [Error: Job Run Fails When Accessing Amazon S3 with an s3a:// URI \(p. 629\)](#)
- [Error: Amazon S3 Service Token Expired \(p. 630\)](#)
- [Error: No Private DNS for Network Interface Found \(p. 631\)](#)
- [Error: Development Endpoint Provisioning Failed \(p. 631\)](#)
- [Error: Notebook Server CREATE_FAILED \(p. 631\)](#)
- [Error: Local Notebook Fails to Start \(p. 631\)](#)
- [Error: Notebook Usage Errors \(p. 631\)](#)
- [Error: Running Crawler Failed \(p. 632\)](#)
- [Error: Upgrading Athena Data Catalog \(p. 632\)](#)
- [Error: A Job is Reprocessing Data When Job Bookmarks Are Enabled \(p. 632\)](#)

Error: Resource Unavailable

If AWS Glue returns a resource unavailable message, you can view error messages or logs to help you learn more about the issue. The following tasks describe general methods for troubleshooting.

- For any connections and development endpoints that you use, check that your cluster has not run out of elastic network interfaces.

Error: Could Not Find S3 Endpoint or NAT Gateway for subnetId in VPC

Check the subnet ID and VPC ID in the message to help you diagnose the issue.

- Check that you have an Amazon S3 VPC endpoint set up, which is required with AWS Glue. In addition, check your NAT gateway if that's part of your configuration. For more information, see [Amazon VPC Endpoints for Amazon S3 \(p. 30\)](#).

Error: Inbound Rule in Security Group Required

At least one security group must open all ingress ports. To limit traffic, the source security group in your inbound rule can be restricted to the same security group.

- For any connections that you use, check your security group for an inbound rule that is self-referencing. For more information, see [Setting Up Your Environment to Access Data Stores \(p. 30\)](#).
- When you are using a development endpoint, check your security group for an inbound rule that is self-referencing. For more information, see [Setting Up Your Environment to Access Data Stores \(p. 30\)](#).

Error: Outbound Rule in Security Group Required

At least one security group must open all egress ports. To limit traffic, the source security group in your outbound rule can be restricted to the same security group.

- For any connections that you use, check your security group for an outbound rule that is self-referencing. For more information, see [Setting Up Your Environment to Access Data Stores \(p. 30\)](#).
- When you are using a development endpoint, check your security group for an outbound rule that is self-referencing. For more information, see [Setting Up Your Environment to Access Data Stores \(p. 30\)](#).

Error: Job Run Failed Because the Role Passed Should Be Given Assume Role Permissions for the AWS Glue Service

The user who defines a job must have permission for `iam:PassRole` for AWS Glue.

- When a user creates an AWS Glue job, confirm that the user's role contains a policy that contains `iam:PassRole` for AWS Glue. For more information, see [Step 3: Attach a Policy to IAM Users That Access AWS Glue \(p. 15\)](#).

Error: DescribeVpcEndpoints Action Is Unauthorized. Unable to Validate VPC ID `vpc-id`

- Check the policy passed to AWS Glue for the `ec2:DescribeVpcEndpoints` permission.

Error: DescribeRouteTables Action Is Unauthorized. Unable to Validate Subnet Id: subnet-id in VPC id: vpc-id

- Check the policy passed to AWS Glue for the `ec2:DescribeRouteTables` permission.

Error: Failed to Call ec2:DescribeSubnets

- Check the policy passed to AWS Glue for the `ec2:DescribeSubnets` permission.

Error: Failed to Call ec2:DescribeSecurityGroups

- Check the policy passed to AWS Glue for the `ec2:DescribeSecurityGroups` permission.

Error: Could Not Find Subnet for AZ

- The Availability Zone might not be available to AWS Glue. Create and use a new subnet in a different Availability Zone from the one specified in the message.

Error: Job Run Exception When Writing to a JDBC Target

When you are running a job that writes to a JDBC target, the job might encounter errors in the following scenarios:

- If your job writes to a Microsoft SQL Server table, and the table has columns defined as type `Boolean`, then the table must be predefined in the SQL Server database. When you define the job on the AWS Glue console using a SQL Server target with the option **Create tables in your data target**, don't map any source columns to a target column with data type `Boolean`. You might encounter an error when the job runs.

You can avoid the error by doing the following:

- Choose an existing table with the `Boolean` column.
- Edit the `ApplyMapping` transform and map the `Boolean` column in the source to a number or string in the target.
- Edit the `ApplyMapping` transform to remove the `Boolean` column from the source.
- If your job writes to an Oracle table, you might need to adjust the length of names of Oracle objects. In some versions of Oracle, the maximum identifier length is limited to 30 bytes or 128 bytes. This limit affects the table names and column names of Oracle target data stores.

You can avoid the error by doing the following:

- Name Oracle target tables within the limit for your version.
- The default column names are generated from the field names in the data. To handle the case when the column names are longer than the limit, use `ApplyMapping` or `RenameField` transforms to change the name of the column to be within the limit.

Error: Amazon S3 Timeout

If AWS Glue returns a connect timed out error, it might be because it is trying to access an Amazon S3 bucket in another AWS Region.

- An Amazon S3 VPC endpoint can only route traffic to buckets within an AWS Region. If you need to connect to buckets in other Regions, a possible workaround is to use a NAT gateway. For more information, see [NAT Gateways](#).

Error: Amazon S3 Access Denied

If AWS Glue returns an access denied error to an Amazon S3 bucket or object, it might be because the IAM role provided does not have a policy with permission to your data store.

- An ETL job must have access to an Amazon S3 data store used as a source or target. A crawler must have access to an Amazon S3 data store that it crawls. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 14\)](#).

Error: Amazon S3 Access Key ID Does Not Exist

If AWS Glue returns an access key ID does not exist error when running a job, it might be because of one of the following reasons:

- An ETL job uses an IAM role to access data stores, confirm that the IAM role for your job was not deleted before the job started.
- An IAM role contains permissions to access your data stores, confirm that any attached Amazon S3 policy containing s3>ListBucket is correct.

Error: Job Run Fails When Accessing Amazon S3 with an s3a:// URI

If a job run returns an error like *Failed to parse XML document with handler class*, it might be because of a failure trying to list hundreds of files using an s3a:// URI. Access your data store using an s3:// URI instead. The following exception trace highlights the errors to look for:

```
1. com.amazonaws.SdkClientException: Failed to parse XML document with handler class
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser$ListBucketHandler
2. at
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseXmlInputStream(XmlResponsesSaxPar
3. at
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseListBucketObjectsResponse(XmlRespon
4. at com.amazonaws.services.s3.model.transform.Unmarshallers
   $ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:70)
5. at com.amazonaws.services.s3.model.transform.Unmarshallers
   $ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:59)
6. at
   com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:62)
7. at
   com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:31)
8. at
   com.amazonaws.http.response.AwsResponseHandlerAdapter.handle(AwsResponseHandlerAdapter.java:70)
9. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.handleResponse(AmazonHttpClient.java:1554)
```

```
10. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeOneRequest(AmazonHttpClient.java:1272)
11. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeHelper(AmazonHttpClient.java:1056)
12. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.doExecute(AmazonHttpClient.java:743)
13. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeWithTimer(AmazonHttpClient.java:717)
14. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.execute(AmazonHttpClient.java:699)
15. at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access
$500(AmazonHttpClient.java:667)
16. at com.amazonaws.http.AmazonHttpClient
$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:649)
17. at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:513)
18. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4325)
19. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4272)
20. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4266)
21. at com.amazonaws.services.s3.AmazonS3Client.listObjects(AmazonS3Client.java:834)
22. at org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:971)
23. at
org.apache.hadoop.fs.s3a.S3AFileSystem.deleteUnnecessaryFakeDirectories(S3AFileSystem.java:1155)
24. at org.apache.hadoop.fs.s3a.S3AFileSystem.finishedWrite(S3AFileSystem.java:1144)
25. at org.apache.hadoop.fs.s3a.S3AOutputStream.close(S3AOutputStream.java:142)
26. at org.apache.hadoop.fs.FSDataOutputStream
$PositionCache.close(FSDataOutputStream.java:74)
27. at org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:108)
28. at org.apache.parquet.hadoop.ParquetFileWriter.end(ParquetFileWriter.java:467)
29. at
org.apache.parquet.hadoop.InternalParquetRecordWriter.close(InternalParquetRecordWriter.java:117)
30. at org.apache.parquet.hadoop.ParquetRecordWriter.close(ParquetRecordWriter.java:112)
31. at
org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.close(ParquetOutputWriter.scala)
32. at org.apache.spark.sql.execution.datasources.FileFormatWriter
$SingleDirectoryWriteTask.releaseResources(FileFormatWriter.scala:252)
33. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:191)
34. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:188)
35. at org.apache.spark.util.Utils$.tryWithSafeFinallyAndFailureCallbacks(Utils.scala:1341)
36. at org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark$sql
$execution$datasources$FileFormatWriter$$executeTask(FileFormatWriter.scala:193)
37. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:129)
38. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:128)
39. at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
40. at org.apache.spark.scheduler.Task.run(Task.scala:99)
41. at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:282)
42. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
43. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
44. at java.lang.Thread.run(Thread.java:748)
```

Error: Amazon S3 Service Token Expired

When moving data to and from Amazon Redshift, temporary Amazon S3 credentials, which expire after 1 hour, are used. If you have a long running job, it might fail. For information about how to set up your long running jobs to move data to and from Amazon Redshift, see [Moving Data to and from Amazon Redshift \(p. 305\)](#).

Error: No Private DNS for Network Interface Found

If a job fails or a development endpoint fails to provision, it might be because of a problem in the network setup.

- If you are using the Amazon provided DNS, the value of `enableDnsHostnames` must be set to true. For more information, see [DNS](#).

Error: Development Endpoint Provisioning Failed

If AWS Glue fails to successfully provision a development endpoint, it might be because of a problem in the network setup.

- When you define a development endpoint, the VPC, subnet, and security groups are validated to confirm that they meet certain requirements.
- If you provided the optional SSH public key, check that it is a valid SSH public key.
- Check in the VPC console that your VPC uses a valid **DHCP option set**. For more information, see [DHCP option sets](#).
- If the cluster remains in the PROVISIONING state, contact AWS Support.

Error: Notebook Server CREATE_FAILED

If AWS Glue fails to create the notebook server for a development endpoint, it might be because of one of the following problems:

- AWS Glue passes an IAM role to Amazon EC2 when it is setting up the notebook server. The IAM role must have a trust relationship to Amazon EC2.
- The IAM role must have an instance profile of the same name. When you create the role for Amazon EC2 with the IAM console, the instance profile with the same name is automatically created. Check for an error in the log regarding an invalid instance profile name `iamInstanceProfile.name`. For more information, see [Using Instance Profiles](#).
- Check that your role has permission to access `aws-glue*` buckets in the policy that you pass to create the notebook server.

Error: Local Notebook Fails to Start

If your local notebook fails to start and reports errors that a directory or folder cannot be found, it might be because of one of the following problems:

- If you are running on Microsoft Windows, make sure that the `JAVA_HOME` environment variable points to the correct Java directory. It's possible to update Java without updating this variable, and if it points to a folder that no longer exists, Zeppelin notebooks fail to start.

Error: Notebook Usage Errors

When using an Apache Zeppelin notebook, you might encounter errors due to your setup or environment.

- You provide an IAM role with an attached policy when you created the notebook server. If the policy does not include all the required permissions, you might get an error such as `assumed-`

role/*name-of-role*/i-0bf0fa9d038087062 is not authorized to perform *some-action*. AccessDeniedException. Check the policy that is passed to your notebook server in the IAM console.

- If the Zeppelin notebook does not render correctly in your web browser, check the Zeppelin requirements for browser support. For example, there might be specific versions and setup required for the Safari browser. You might need to update your browser or use a different browser.

Error: Running Crawler Failed

If AWS Glue fails to successfully run a crawler to catalog your data, it might be because of one of the following reasons. First check if an error is listed in the AWS Glue console crawlers list. Check if there is an exclamation icon next to the crawler name and hover over the icon to see any associated messages.

- Check the logs for the crawler run in CloudWatch Logs under /aws-glue/crawlers.

Error: Upgrading Athena Data Catalog

If you encounter errors while upgrading your Athena Data Catalog to the AWS Glue Data Catalog, see the *Amazon Athena User Guide* topic [Upgrading to the AWS Glue Data Catalog Step-by-Step](#).

Error: A Job is Reprocessing Data When Job Bookmarks Are Enabled

There might be cases when you have enabled AWS Glue job bookmarks, but your ETL job is reprocessing data that was already processed in an earlier run. Check for these common causes of this error:

Max Concurrency

Ensure that the maximum number of concurrent runs for the job is 1. For more information, see the discussion of max concurrency in [Adding Jobs in AWS Glue \(p. 164\)](#). When you have multiple concurrent jobs with job bookmarks and the maximum concurrency is set to 1, the job bookmark doesn't work correctly.

Missing Job Object

Ensure that your job run script ends with the following commit:

```
job.commit()
```

When you include this object, AWS Glue records the timestamp and path of the job run. If you run the job again with the same path, AWS Glue processes only the new files. If you don't include this object and job bookmarks are enabled, the job reprocesses the already processed files along with the new files and creates redundancy in the job's target data store.

Missing Transformation Context Parameter

Transformation context is an optional parameter in the `GlueContext` class, but job bookmarks don't work if you don't include it. To resolve this error, add the transformation context parameter when you [create the DynamicFrame](#), as shown following:

```
sample_dynF=create_dynamic_frame_from_catalog(database,
    table_name,transformation_ctx="sample_dynF")
```

Input Source

If you are using a relational database (a JDBC connection) for the input source, job bookmarks work only if the table's primary keys are in sequential order. Job bookmarks work for new rows, but not for updated rows. That is because job bookmarks look for the primary keys, which already exist. This does not apply if your input source is Amazon Simple Storage Service (Amazon S3).

Last Modified Time

For Amazon S3 input sources, job bookmarks check the last modified time of the objects, rather than the file names, to verify which objects need to be reprocessed. If your input source data has been modified since your last job run, the files are reprocessed when you run the job again.

AWS Glue Quotas

You can contact AWS Support to [request a quota increase](#) for the service quotas listed in the *AWS General Reference*. Unless otherwise noted, each quota is Region-specific. For more information, see [AWS Glue Endpoints and Quotas](#).

Known Issues for AWS Glue

Note the following known issues for AWS Glue.

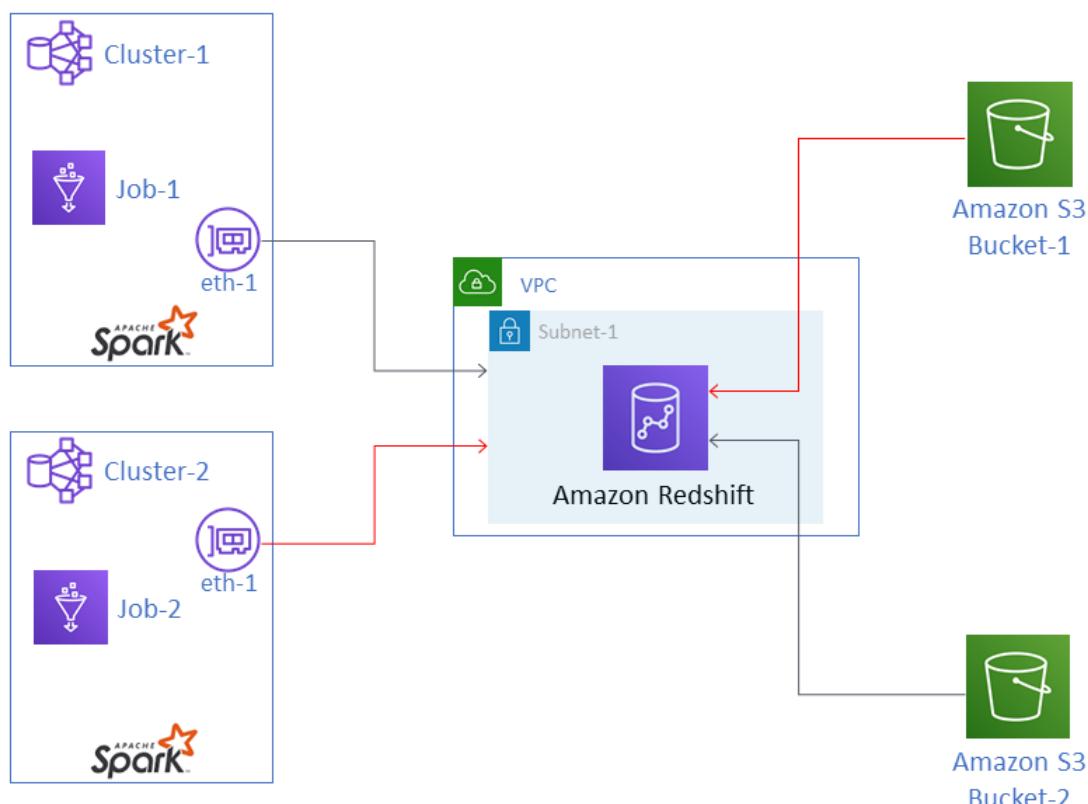
Topics

- Preventing Cross-Job Data Access (p. 634)

Preventing Cross-Job Data Access

Consider the situation where you have two AWS Glue Spark jobs in a single AWS account, each running in a separate AWS Glue Spark cluster. The jobs are using AWS Glue connections to access resources in the same virtual private cloud (VPC). In this situation, a job running in one cluster might be able to access the data from the job running in the other cluster.

The following diagram illustrates an example of this situation.



In the diagram, AWS Glue Job-1 is running in Cluster-1, and Job-2 is running in Cluster-2. Both jobs are working with the same instance of Amazon Redshift, which resides in Subnet-1 of a VPC. Subnet-1 could be a public or private subnet.

Job-1 is transforming data from Amazon Simple Storage Service (Amazon S3) Bucket-1 and writing the data to Amazon Redshift. Job-2 is doing the same with data in Bucket-2. Job-1 uses the AWS

Identity and Access Management (IAM) role Role-1 (not shown), which gives access to Bucket-1. Job-2 uses Role-2 (not shown), which gives access to Bucket-2.

These jobs have network paths that enable them to communicate with each other's clusters and thus access each other's data. For example, Job-2 could access data in Bucket-1. In the diagram, this is shown as the path in red.

To prevent this situation, we recommend that you attach different security configurations to Job-1 and Job-2. By attaching the security configurations, cross-job access to data is blocked by virtue of certificates that AWS Glue creates. The security configurations can be *dummy* configurations. That is, you can create the security configurations without enabling encryption of Amazon S3 data, Amazon CloudWatch data, or job bookmarks. All three encryption options can be disabled.

For information about security configurations, see [the section called "Encrypting Data Written by AWS Glue" \(p. 44\)](#).

To attach a security configuration to a job

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. On the **Configure the job properties** page for the job, expand the **Security configuration, script libraries, and job parameters** section.
3. Select a security configuration in the list.

AWS Glue Release Notes

The Glue version parameter is configured when adding or updating a job. Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The Python version indicates the version supported for jobs of type Spark. The following table lists the available Glue versions, the corresponding Spark and Python versions, and other changes in functionality.

AWS Glue Versions

Glue version	Supported Spark and Python versions	Changes in Functionality
Glue 0.9	<ul style="list-style-type: none">• Spark 2.2.1• Python 2.7	Jobs that were created without specifying a Glue version default to Glue 0.9.
Glue 1.0	<ul style="list-style-type: none">• Spark 2.4.3• Python 2.7• Python 3.6	You can maintain job bookmarks for Parquet and ORC formats in Glue ETL jobs (using Glue Version 1.0). Previously, you were only able to bookmark common Amazon S3 source formats such as JSON, CSV, Apache Avro and XML in AWS Glue ETL jobs.

Document History for AWS Glue

The following table describes important changes to the documentation for AWS Glue.

- **Latest API version:** 2019-01-15
- **Latest documentation update:** January 16, 2020

update-history-change	update-history-description	update-history-date
Support for new transforms to work with datasets in Amazon S3 (p. 637)	Added information about new transforms (Merge, Purge, and Transition) and Amazon S3 storage class exclusions for Apache Spark applications to work with datasets in Amazon S3. For more information on support for these transforms for Python, see mergeDynamicFrame and Working with Datasets in Amazon S3 . For Scala, see mergeDynamicFrames and AWS Glue Scala GlueContext APIs .	January 16, 2020
Support for updating the Data Catalog with new partition information from an ETL job (p. 637)	Added information about how to code an extract, transform, and load (ETL) script to update the AWS Glue Data Catalog with new partition information. With this capability, you no longer have to rerun the crawler after job completion to view the new partitions. For more information see Updating the Data Catalog with New Partitions .	January 15, 2020
New tutorial: Using an Amazon SageMaker notebook (p. 637)	Added a tutorial that demonstrates how to use an Amazon SageMaker notebook to help develop your ETL and machine learning scripts. See Tutorial: Use an Amazon SageMaker Notebook with Your Development Endpoint .	January 3, 2020
Various corrections and clarifications (p. 637)	Added corrections and clarifications throughout. Removed entries from the Known Issues chapter. Added warnings that AWS Glue supports only symmetrical customer master keys (CMKs) when specifying Data Catalog encryption settings and creating	December 9, 2019

	security configurations. Added a note that AWS Glue does not support writing to Amazon DynamoDB.	
Support for custom JDBC drivers (p. 637)	Added information about connecting to data sources and targets with JDBC drivers that AWS Glue does not natively support, such as MySQL version 8 and Oracle Database version 18. For more information see JDBC connectionType Values .	November 25, 2019
Support for connecting Amazon SageMaker notebooks to different development endpoints (p. 637)	Added information about how you can connect an Amazon SageMaker notebook to different development endpoints. Updates to describe the new console action for switching to a new development endpoint, and the new Amazon SageMaker IAM policy. For more information, see Working with Notebooks on the AWS Glue Console and Create an IAM Policy for Amazon SageMaker Notebooks .	November 21, 2019
Support for Glue version in machine learning transforms (p. 637)	Added information about defining the Glue version in a machine learning transform to indicate the which version of AWS Glue a machine learning transform is compatible with. For more information see Working with Machine Learning Transforms on the AWS Glue Console .	November 21, 2019
Support for rewinding your job bookmarks (p. 637)	Added information about rewinding your job bookmarks to any previous job run, resulting in the subsequent job run reprocessing data only from the bookmarked job run. Described two new suboptions for the job-bookmark-pause option that allow you to run a job between two bookmarks. For more information, see Tracking Processed Data Using Job Bookmarks and Special Parameters Used by AWS Glue .	October 22, 2019

Support for custom JDBC certificates for connecting to a data store (p. 637)	Added information about AWS Glue support of custom JDBC certificates for SSL connections to AWS Glue data sources or targets. For more information, see Working with Connections on the AWS Glue Console .	October 10, 2019
Support for Python wheel (p. 637)	Added information about AWS Glue support of wheel files (along with egg files) as dependencies for Python shell jobs. For more information, see Providing Your Own Python Library .	September 26, 2019
Support for versioning of development endpoints in AWS Glue (p. 637)	Added information about defining the <code>Glue_version</code> in development endpoints. <code>Glue_version</code> determines the versions of Apache Spark and Python that AWS Glue supports. For more information, see Adding a Development Endpoint .	September 19, 2019
Support for monitoring AWS Glue using Spark UI (p. 637)	Added information about using Apache Spark UI to monitor and debug AWS Glue ETL jobs running on the AWS Glue job system, and Spark applications on AWS Glue development endpoints. For more information, see Monitoring AWS Glue Using Spark UI .	September 19, 2019
Enhancement of support for local ETL script development using the public AWS Glue ETL library (p. 637)	Updated the AWS Glue ETL library content to reflect that AWS Glue version 1.0 is now supported. For more information, see Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library .	September 18, 2019
Support for excluding Amazon S3 storage classes when running jobs (p. 637)	Added information about excluding Amazon S3 storage classes when running AWS Glue ETL jobs that read files or partitions from Amazon S3. For more information, see Excluding Amazon S3 Storage Classes .	August 29, 2019

Support for local ETL script development using the public AWS Glue ETL library (p. 637)	Added information about how to develop and test Python and Scala ETL scripts locally without the need for a network connection. For more information, see Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library .	August 28, 2019
Known Issues (p. 637)	Added information about known issues in AWS Glue. For more information, see Known Issues for AWS Glue .	August 28, 2019
Support for machine learning transforms in AWS Glue (p. 637)	Added information about machine learning capabilities provided by AWS Glue to create custom transforms. You can create these transforms when you create a job. For more information, see Machine Learning Transforms in AWS Glue .	August 8, 2019
Support for shared Amazon Virtual Private Cloud (p. 637)	Added information about AWS Glue support for shared Amazon Virtual Private Cloud. For more information, see Shared Amazon VPCs .	August 6, 2019
Support for versioning in AWS Glue (p. 637)	Added information about defining the <code>Glue version</code> in job properties. Glue version determines the versions of Apache Spark and Python that AWS Glue supports. For more information, see Adding Jobs in AWS Glue .	July 24, 2019
Support for additional configuration options for development endpoints (p. 637)	Added information about configuration options for development endpoints that have memory-intensive workloads. You can choose from two new configurations that provide more memory per executor. For more information, see Working with Development Endpoints on the AWS Glue Console .	July 24, 2019

Support for performing extract, transfer, and load (ETL) activities using workflows (p. 637)	Added information about using a new construct called a workflow to design a complex multi-job extract, transform, and load (ETL) activity that AWS Glue can execute and track as single entity. For more information, see Performing Complex ETL Activities Using Workflows in AWS Glue .	June 20, 2019
Support for Python 3.6 in Python shell jobs (p. 637)	Added information about support for Python 3.6 in Python shell jobs. You can specify either Python 2.7 or Python 3.6 as a job property. For more information, see Adding Python Shell Jobs in AWS Glue .	June 5, 2019
Support for virtual private cloud (VPC) endpoints (p. 637)	Added information about connecting directly to AWS Glue through an interface endpoint in your VPC. When you use a VPC interface endpoint, communication between your VPC and AWS Glue is conducted entirely and securely within the AWS network. For more information, see Using AWS Glue with VPC Endpoints .	June 4, 2019
Support for real-time, continuous logging for AWS Glue jobs. (p. 637)	Added information about enabling and viewing real-time Apache Spark job logs in CloudWatch including the driver logs, each of the executor logs, and a Spark job progress bar. For more information, see Continuous Logging for AWS Glue Jobs .	May 28, 2019
Support for existing Data Catalog tables as crawler sources (p. 637)	Added information about specifying a list of existing Data Catalog tables as crawler sources. Crawlers can then detect changes to table schemas, update table definitions, and register new partitions as new data becomes available. For more information, see Crawler Properties .	May 10, 2019

Support for additional configuration options for memory-intensive jobs (p. 637)	Added information about configuration options for Apache Spark jobs with memory-intensive workloads. You can choose from two new configurations that provide more memory per executor. For more information, see Adding Jobs in AWS Glue .	April 5, 2019
Support for CSV custom classifiers (p. 637)	Added information about using a custom CSV classifier to infer the schema of various types of CSV data. For more information, see Writing Custom Classifiers .	March 26, 2019
Support for AWS Resource Tags (p. 637)	Added information about using AWS resource tags to help you manage and control access to your AWS Glue resources. You can assign AWS resource tags to jobs, triggers, endpoints, and crawlers in AWS Glue. For more information, see AWS Tags in AWS Glue .	March 20, 2019
Support of AWS Glue Data Catalog for Spark SQL jobs (p. 637)	Added information about configuring your AWS Glue jobs and development endpoints to use the AWS Glue Data Catalog as an external Apache Hive Metastore. This allows jobs and development endpoints to directly run Apache Spark SQL queries against the tables stored in the AWS Glue Data Catalog. For more information, see AWS Glue Data Catalog Support for Spark SQL Jobs .	March 14, 2019
Support for Python shell jobs (p. 637)	Added information about Python shell jobs and the new field Maximum capacity . For more information, see Adding Python Shell Jobs in AWS Glue .	January 18, 2019
Support for notifications when there are changes to databases and tables (p. 637)	Added information about events that are generated for changes to database, table, and partition API calls. You can configure actions in CloudWatch Events to respond to these events. For more information, see Automating AWS Glue with CloudWatch Events .	January 16, 2019

Support for encrypting connection passwords (p. 637)	Added information about encrypting passwords used in connection objects. For more information, see Encrypting Connection Passwords .	December 11, 2018
Support for resource-level permission and resource-based policies (p. 637)	Added information about using resource-level permissions and resource-based policies with AWS Glue. For more information, see the topics within Security in AWS Glue .	October 15, 2018
Support for Amazon SageMaker notebooks (p. 637)	Added information about using Amazon SageMaker notebooks with AWS Glue development endpoints. For more information, see Managing Notebooks .	October 5, 2018
Support for encryption (p. 637)	Added information about using encryption with AWS Glue. For more information, see Encryption at Rest, Encryption in Transit, and Setting Up Encryption in AWS Glue .	August 24, 2018
Support for Apache Spark job metrics (p. 637)	Added information about the use of Apache Spark metrics for better debugging and profiling of ETL jobs. You can easily track runtime metrics such as bytes read and written, memory usage and CPU load of the driver and executors, and data shuffles among executors from the AWS Glue console. For more information, see Monitoring AWS Glue Using CloudWatch Metrics, Job Monitoring and Debugging, and Working with Jobs on the AWS Glue Console .	July 13, 2018
Support of DynamoDB as a data source (p. 637)	Added information about crawling DynamoDB and using it as a data source of ETL jobs. For more information, see Cataloging Tables with a Crawler and Connection Parameters .	July 10, 2018
Updates to create notebook server procedure (p. 637)	Updated information about how to create a notebook server on an Amazon EC2 instance associated with a development endpoint. For more information, see Creating a Notebook Server Associated with a Development Endpoint .	July 9, 2018

Updates now available over RSS (p. 637)	You can now subscribe to an RSS feed to receive notifications about updates to the AWS Glue Developer Guide.	June 25, 2018
Support delay notifications for jobs (p. 637)	Added information about configuring a delay threshold when a job runs. For more information, see Adding Jobs in AWS Glue .	May 25, 2018
Configure a crawler to append new columns (p. 637)	Added information about new configuration option for crawlers, MergeNewColumns. For more information, see Configuring a Crawler .	May 7, 2018
Support timeout of jobs (p. 637)	Added information about setting a timeout threshold when a job runs. For more information, see Adding Jobs in AWS Glue .	April 10, 2018
Support Scala ETL script and trigger jobs based on additional run states (p. 637)	Added information about using Scala as the ETL programming language. In addition, the trigger API now supports firing when any conditions are met (in addition to all conditions). Also, jobs can be triggered based on a "failed" or "stopped" job run (in addition to a "succeeded" job run).	January 12, 2018

Earlier Updates

The following table describes the important changes in each release of the *AWS Glue Developer Guide* before January 2018.

Change	Description	Date
Support XML data sources and new crawler configuration option	Added information about classifying XML data sources and new crawler option for partition changes.	November 16, 2017
New transforms, support for additional Amazon RDS database engines, and development endpoint enhancements	Added information about the map and filter transforms, support for Amazon RDS Microsoft SQL Server, and Amazon RDS Oracle, and new features for development endpoints.	September 29, 2017
AWS Glue initial release	This is the initial release of the <i>AWS Glue Developer Guide</i> .	August 14, 2017

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.