
AWS Glue

Developer Guide



AWS Glue: Developer Guide

Copyright © 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

AWS Glue and AWS Glue Studio preview features	1
What Is AWS Glue?	2
When Should I Use AWS Glue?	2
How It Works	4
Serverless ETL Jobs Run in Isolation	5
Concepts	5
AWS Glue Terminology	7
Components	8
AWS Glue Console	8
AWS Glue Data Catalog	9
AWS Glue Crawlers and Classifiers	9
AWS Glue ETL Operations	10
Streaming ETL in AWS Glue	10
The AWS Glue Jobs System	10
Converting Semi-Structured Schemas to Relational Schemas	10
Setting up	12
Setting up IAM Permissions for AWS Glue	12
Step 1: Create an IAM Policy for the AWS Glue Service	12
Step 2: Create an IAM Role for AWS Glue	16
Step 3: Attach a Policy to IAM Users That Access AWS Glue	17
Step 4: Create an IAM Policy for Notebook Servers	25
Step 5: Create an IAM Role for Notebook Servers	28
Step 6: Create an IAM Policy for SageMaker Notebooks	29
Step 7: Create an IAM Role for SageMaker Notebooks	30
Setting Up DNS in Your VPC	31
Setting Up Your Environment to Access Data Stores	32
Amazon VPC Endpoints for Amazon S3	32
Setting Up a VPC to Connect to JDBC Data Stores	34
Setting Up Encryption	36
Setting Up Your Environment for Development Endpoints	39
Setting Up Your Network for a Development Endpoint	39
Setting Up Amazon EC2 for a Notebook Server	40
Getting Started	42
Console Workflow Overview	42
Getting Started with the AWS Glue Data Catalog	43
Overview	43
Step 1: Create a Database	43
Step 2. Create a Table	44
Security	48
Data Protection	48
Encryption at Rest	48
Encryption in Transit	54
FIPS Compliance	55
Key Management	55
AWS Glue Dependency on Other AWS Services	55
Development Endpoints	55
Identity and access management	56
Authentication	56
Access control overview	57
Console permissions	61
Identity-based policies	61
Resource policies	68
Granting cross-account access	71
Resource ARNs	76

Policy examples	81
API permissions reference	95
Logging and Monitoring	96
Compliance Validation	96
Resilience	97
Infrastructure Security	97
VPC endpoints (AWS PrivateLink)	97
Shared Amazon VPCs	99
Populating the AWS Glue Data Catalog	100
Defining a Database in Your Data Catalog	101
Database Resource Links	101
Working with Databases on the Console	102
Defining Tables in the AWS Glue Data Catalog	102
Table Partitions	103
Table Resource Links	103
Updating Manually Created Tables with Crawlers	104
Working with Tables on the Console	104
Working with Partition Indexes	107
Defining Connections in the AWS Glue Data Catalog	110
AWS Glue Connections	110
AWS Glue Connection Properties	111
Adding an AWS Glue Connection	115
Testing an AWS Glue Connection	115
Configuring AWS calls to go through your VPC	116
Connecting to a JDBC Data Store in a VPC	116
Using a MongoDB Connection	118
Crawling an Amazon S3 Data Store using a VPC Endpoint	119
Defining Crawlers	125
Which Data Stores Can I Crawl?	125
How Crawlers Work	126
Crawler Prerequisites	130
Crawler Properties	131
Setting Crawler Configuration Options	137
Scheduling a Crawler	143
Working with Crawlers on the Console	144
Accelerating Crawls Using Amazon S3 Event Notifications	145
Using Encryption with the Amazon S3 Event Crawler	152
Adding Classifiers to a Crawler	157
When Do I Use a Classifier?	157
Custom Classifiers	157
Built-In Classifiers in AWS Glue	158
Writing Custom Classifiers	160
Working with Classifiers on the Console	171
Working with Data Catalog Settings on the AWS Glue Console	173
Creating Tables, Updating Schema, and Adding New Partitions in the Data Catalog from AWS Glue ETL Jobs	174
New Partitions	174
Updating Table Schema	175
Creating New Tables	176
Restrictions	176
Creating AWS Glue resources using AWS CloudFormation templates	178
Sample Database	179
Sample Database, Table, Partitions	180
Sample Grok Classifier	183
Sample JSON Classifier	183
Sample XML Classifier	184
Sample Amazon S3 Crawler	185

Sample Connection	186
Sample JDBC Crawler	187
Sample Job for Amazon S3 to Amazon S3	189
Sample Job for JDBC to Amazon S3	190
Sample On-Demand Trigger	191
Sample Scheduled Trigger	192
Sample Conditional Trigger	193
Sample Development Endpoint	194
Authoring Jobs	196
Workflow Overview	197
Adding Jobs	197
Defining Job Properties for Spark Jobs	198
Adding Python Shell Jobs	202
Adding Streaming ETL Jobs	206
Built-In Transforms	214
Jobs on the Console	216
Restrictions for Jobs That Access Lake Formation Managed Tables	221
Using Auto Scaling for AWS Glue	221
Editing Scripts	225
Defining a Script	225
Scripts on the Console	227
Providing Your Own Custom Scripts	228
Integrating with MongoDB	229
Interactive applications and script development	230
Overview of AWS Glue Interactive Sessions	230
Getting started with AWS Glue interactive sessions	231
Configuring AWS Glue Interactive Sessions for Jupyter and AWS Glue Studio notebooks	241
Interactive Sessions with IAM	245
Converting a script or notebook into a Glue job	249
AWS Glue Interactive Sessions for streaming	250
Developing Scripts Using Development Endpoints	252
Development Environment	252
Development Endpoint Workflow	252
How Development Endpoints Work with SageMaker Notebooks	253
Adding a Development Endpoint	254
Viewing Development Endpoint Properties	256
Accessing Your Development Endpoint	257
Creating a Notebook Server Hosted on Amazon EC2	258
Tutorial Prerequisites	260
Tutorial: Local Zeppelin Notebook	264
Tutorial: Amazon EC2 Zeppelin Notebook Server	267
Tutorial: Jupyter Notebook in JupyterLab	269
Tutorial: Use a SageMaker Notebook	272
Tutorial: Use a REPL Shell	274
Tutorial: Use PyCharm Professional	275
Advanced Configuration: Sharing Dev Endpoints among Multiple Users	281
Managing Notebooks	285
Notebook Server Considerations	287
Working with Notebooks on the Console	293
Starting Jobs and Crawlers Using Triggers	296
AWS Glue Triggers	296
Adding Triggers	297
Activating and Deactivating Triggers	298
Running and Monitoring	300
Automated Tools	301
Time-Based Schedules for Jobs and Crawlers	301
Cron Expressions	301

Tracking Processed Data Using Job Bookmarks	303
Using Job Bookmarks	304
Using an AWS Glue Script	306
Workload Partitioning with Bounded Execution	309
Enabling Workload Partitioning	309
Setting up an AWS Glue Trigger to Automatically Run the Job	310
AWS Tags	310
Tagging support for AWS Glue connections	311
Examples	312
Automating with CloudWatch Events	314
Monitoring with the Spark UI	315
Enabling the Spark UI for Jobs	319
Enabling the Spark UI for Dev Endpoints	320
Launching the Spark History Server	321
AWS Glue Spark shuffle manager with Amazon S3	325
Using AWS Glue Spark shuffle manager from the AWS Console	326
Using AWS Glue Spark shuffle manager	327
Monitoring with AWS Glue job run insights	327
Requirements	328
Enabling job run insights	328
Access the Job Run Insights Log Streams	328
Example	329
Monitoring with CloudWatch	331
Using CloudWatch Metrics	331
Setting Up Amazon CloudWatch Alarms on AWS Glue Job Profiles	345
Continuous Logging for AWS Glue Jobs	345
Job Monitoring and Debugging	349
Debugging OOM Exceptions and Job Abnormalities	350
Debugging Demanding Stages and Straggler Tasks	357
Monitoring the Progress of Multiple Jobs	361
Monitoring for DPU Capacity Planning	365
Logging Using CloudTrail	370
AWS Glue Information in CloudTrail	370
Understanding AWS Glue Log File Entries	371
Job Run Statuses	372
Performing Complex ETL Activities Using Blueprints and Workflows	373
Overview of Workflows	373
Overview of Blueprints	376
Developing Blueprints	378
Overview of Developing Blueprints	378
Prerequisites for Developing Blueprints	379
Writing the Blueprint Code	382
Sample Blueprint Project	386
Testing a Blueprint	389
Publishing a Blueprint	390
Blueprint Classes Reference	391
Blueprint Samples	394
Registering a Blueprint	394
Viewing Blueprints	396
Updating a Blueprint	397
Creating a Workflow from a Blueprint	398
Viewing Blueprint Runs	400
Creating and Building Out a Workflow Manually	400
Step 1: Create the Workflow	401
Step 2: Add a Start Trigger	401
Step 3: Add More Triggers	402
Starting a Workflow with an EventBridge Event	404

Viewing the EventBridge Events That Started a Workflow	408
Running and Monitoring a Workflow	409
Stopping a Workflow Run	411
Repairing and Resuming a Workflow Run	411
Resuming a Workflow Run: How It Works	412
Resuming a Workflow Run	413
Notes and Limitations for Resuming Workflow Runs	415
Getting and Setting Workflow Run Properties	416
Querying Workflows Using the AWS Glue API	417
Querying Static Views	417
Querying Dynamic Views	417
Blueprint and Workflow Restrictions	420
Blueprint Restrictions	420
Workflow Restrictions	420
Troubleshooting Blueprint errors	420
Error: missing PySpark module	421
Error: missing blueprint config file	421
Error: missing imported file	422
Error: not authorized to perform iamPassRole on resource	422
Error: invalid cron schedule	422
Error: a trigger with the same name already exists	422
Error: Workflow with name: foo already exists.	423
Error: module not found in specified layoutGenerator path	423
Error: validation error in Connections field	423
Permissions for Blueprint Personas and Roles	424
Blueprint Personas	424
Permissions for Blueprint Personas	424
Permissions for Blueprint Roles	426
AWS Glue Schema Registry	428
Schemas	428
Registries	430
Schema Versioning and Compatibility	431
Open Source Serde Libraries	434
Quotas of the Schema Registry	434
How it Works	434
Getting Started	436
Installing SerDe Libraries	436
Using AWS CLI for the AWS Glue Schema Registry APIs	437
Creating a Registry	438
Dealing with a Specific Record (JAVA POJO) for JSON	439
Creating a Schema	440
Updating a Schema or Registry	444
Deleting a Schema or Registry	447
IAM Examples for Serializers	449
IAM Examples for Deserializers	450
Private Connectivity using AWS PrivateLink	450
Accessing Amazon CloudWatch Metrics	450
Integrating with AWS Glue Schema Registry	451
Use Case: Connecting Schema Registry to Amazon MSK or Apache Kafka	451
Use Case: Integrating Amazon Kinesis Data Streams with the AWS Glue Schema Registry	452
Use Case: Amazon Kinesis Data Analytics for Apache Flink	459
Use Case: Integration with AWS Lambda	463
Use Case: AWS Glue Data Catalog	463
Use case: AWS Glue Streaming	464
Use Case: Apache Kafka Streams	465
Use Case: Apache Kafka Connect	466
Migration from a Third-Party Schema Registry to AWS Glue Schema Registry	469

Sample AWS CloudFormation Template for Schema Registry	470
ETL Programming	472
General Information	472
Special Parameters	472
Connection Parameters	475
Examples: Setting Connection Types and Options	501
Format Options	504
Managing Partitions	509
Grouping Input Files	511
Reading from JDBC in Parallel	512
Moving Data to and from Amazon Redshift	513
Data Catalog Support for Spark SQL Jobs	514
Excluding Amazon S3 Storage Classes	517
Developing and testing AWS Glue job scripts	518
Cross-Account Cross-Region Access to DynamoDB Tables	531
ETL Programming in Python	533
Using Python	533
List of Extensions	534
List of Transforms	534
Python Setup	535
Calling APIs	535
Python Libraries	537
Python Samples	540
PySpark Extensions	553
PySpark Transforms	588
ETL Programming in Scala	625
Using Scala	630
Scala Script Example	631
Scala API List	632
Matching Records with FindMatches	682
Types of Machine Learning Transforms	682
Find Matches Transform	683
Tuning Machine Learning Transforms	686
Machine Learning Measurements	686
Deciding Between Precision and Recall	687
Deciding Between Accuracy and Cost	688
Estimating the quality of matches using match confidence scores	688
Teaching the Find Matches Transform	690
Machine Learning Transforms on the Console	691
Transform Properties	691
Adding and Editing Machine Learning Transforms	692
Viewing Transform Details	693
Tutorial: Creating a Machine Learning Transform	695
Step 1: Crawl the Source Data	695
Step 2: Add a Machine Learning Transform	695
Step 3: Teach Your Machine Learning Transform	696
Step 4: Estimate the Quality of Your Machine Learning Transform	696
Step 5: Add and Run a Job with Your Machine Learning Transform	697
Step 6: Verify Output Data from Amazon S3	698
Finding Incremental Matches	699
Running an incremental matching job	699
AWS Glue API	701
Security	712
— data types —	712
DataCatalogEncryptionSettings	712
EncryptionAtRest	713
ConnectionPasswordEncryption	713

EncryptionConfiguration	714
S3Encryption	714
CloudWatchEncryption	714
JobBookmarksEncryption	714
SecurityConfiguration	715
GluePolicy	715
— operations —	715
GetDataCatalogEncryptionSettings (get_data_catalog_encryption_settings)	716
PutDataCatalogEncryptionSettings (put_data_catalog_encryption_settings)	716
PutResourcePolicy (put_resource_policy)	717
GetResourcePolicy (get_resource_policy)	718
DeleteResourcePolicy (delete_resource_policy)	718
CreateSecurityConfiguration (create_security_configuration)	719
DeleteSecurityConfiguration (delete_security_configuration)	720
GetSecurityConfiguration (get_security_configuration)	720
GetSecurityConfigurations (get_security_configurations)	721
GetResourcePolicies (get_resource_policies)	721
Catalog	722
Databases	722
Tables	728
Partitions	752
Connections	767
User-Defined Functions	776
Importing an Athena; Catalog	781
Crawlers and Classifiers	782
Classifiers	782
Crawlers	792
Scheduler	806
Autogenerating ETL Scripts	808
— data types —	808
CodeGenNode	808
CodeGenNodeArg	808
CodeGenEdge	809
Location	809
CatalogEntry	809
MappingEntry	810
— operations —	810
CreateScript (create_script)	810
GetDataflowGraph (get_dataflow_graph)	811
GetMapping (get_mapping)	811
GetPlan (get_plan)	812
Visual Job API (Preview)	813
— data types —	813
CodeGenConfigurationNode	814
JDBCConnectorOptions	817
StreamingDataPreviewOptions	818
AthenaConnectorSource	818
JDBCConnectorSource	819
SparkConnectorSource	819
CatalogSource	820
CatalogKinesisSource	820
DirectKinesisSource	821
KinesisStreamingSourceOptions	821
CatalogKafkaSource	823
DirectKafkaSource	823
KafkaStreamingSourceOptions	824
RedshiftSource	825

S3CatalogSource	826
S3SourceAdditionalOptions	826
S3CSVSource	826
S3JsonSource	828
S3ParquetSource	829
DynamoDBELTConnectorSource	830
DDBELTConnectionOptions	830
JDBCConnectorTarget	831
SparkConnectorTarget	831
BasicCatalogTarget	832
RedshiftTarget	833
S3CatalogTarget	833
S3GlueParquetTarget	834
CatalogSchemaChangePolicy	834
S3DirectTarget	834
DirectSchemaChangePolicy	835
ApplyMapping	835
Mapping	836
SelectFields	837
DropFields	837
RenameField	837
Spigot	838
Join	838
JoinColumn	838
SplitFields	839
SelectFromCollection	839
FillMissingValues	839
Filter	840
FilterExpression	840
FilterValue	841
CustomCode	841
SparkSQL	841
SqlAlias	842
DropNullFields	842
NullCheckBoxList	843
NullValueField	843
Datatype	843
Merge	843
Union	844
Jobs	844
Jobs	845
Job Runs	856
Triggers	866
Interactive sessions	875
— data types —	875
Session	876
SessionCommand	877
Statement	877
StatementOutput	878
StatementOutputData	878
— operations —	878
CreateSession (create_session)	879
StopSession (stop_session)	881
DeleteSession (delete_session)	881
GetSession (get_session)	882
ListSessions (list_sessions)	882
RunStatement (run_statement)	883

CancelStatement (cancel_statement)	884
GetStatement (get_statement)	884
ListStatements (list_statements)	885
DevEndpoints	886
— data types —	886
DevEndpoint	886
DevEndpointCustomLibraries	888
— operations —	889
CreateDevEndpoint (create_dev_endpoint)	889
UpdateDevEndpoint (update_dev_endpoint)	893
DeleteDevEndpoint (delete_dev_endpoint)	894
GetDevEndpoint (get_dev_endpoint)	894
GetDevEndpoints (get_dev_endpoints)	895
BatchGetDevEndpoints (batch_get_dev_endpoints)	895
ListDevEndpoints (list_dev_endpoints)	896
Schema Registry	897
— data types —	897
RegistryId	897
RegistryListItem	897
MetadataInfo	898
OtherMetadataValueListItem	898
SchemaListItem	899
SchemaVersionListItem	899
MetadataKeyValuePair	900
SchemaVersionErrorItem	900
ErrorDetails	900
SchemaVersionNumber	900
SchemaId	901
— operations —	901
CreateRegistry (create_registry)	902
CreateSchema (create_schema)	903
GetSchema (get_schema)	905
ListSchemaVersions (list_schema_versions)	907
GetSchemaVersion (get_schema_version)	907
GetSchemaVersionsDiff (get_schema_versions_diff)	908
ListRegistries (list_registries)	909
ListSchemas (list_schemas)	910
RegisterSchemaVersion (register_schema_version)	911
UpdateSchema (update_schema)	912
CheckSchemaVersionValidity (check_schema_version_validity)	913
UpdateRegistry (update_registry)	913
GetSchemaByDefinition (get_schema_by_definition)	914
GetRegistry (get_registry)	915
PutSchemaVersionMetadata (put_schema_version_metadata)	916
QuerySchemaVersionMetadata (query_schema_version_metadata)	917
RemoveSchemaVersionMetadata (remove_schema_version_metadata)	918
DeleteRegistry (delete_registry)	919
DeleteSchema (delete_schema)	920
DeleteSchemaVersions (delete_schema_versions)	920
Workflows	921
— data types —	921
JobNodeDetails	922
CrawlerNodeDetails	922
TriggerNodeDetails	922
Crawl	922
Node	923
Edge	923

Workflow	924
WorkflowGraph	924
WorkflowRun	925
WorkflowRunStatistics	926
StartingEventBatchCondition	926
Blueprint	926
BlueprintDetails	927
LastActiveDefinition	927
BlueprintRun	928
— operations —	929
CreateWorkflow (create_workflow)	929
UpdateWorkflow (update_workflow)	930
DeleteWorkflow (delete_workflow)	931
GetWorkflow (get_workflow)	932
ListWorkflows (list_workflows)	932
BatchGetWorkflows (batch_get_workflows)	933
GetWorkflowRun (get_workflow_run)	933
GetWorkflowRuns (get_workflow_runs)	934
GetWorkflowRunProperties (get_workflow_run_properties)	935
PutWorkflowRunProperties (put_workflow_run_properties)	935
CreateBlueprint (create_blueprint)	936
UpdateBlueprint (update_blueprint)	937
DeleteBlueprint (delete_blueprint)	937
ListBlueprints (list_blueprints)	938
BatchGetBlueprints (batch_get_blueprints)	938
StartBlueprintRun (start_blueprint_run)	939
GetBlueprintRun (get_blueprint_run)	940
GetBlueprintRuns (get_blueprint_runs)	940
StartWorkflowRun (start_workflow_run)	941
StopWorkflowRun (stop_workflow_run)	942
ResumeWorkflowRun (resume_workflow_run)	942
Machine Learning	943
— data types —	943
TransformParameters	944
EvaluationMetrics	944
MLTransform	944
FindMatchesParameters	946
FindMatchesMetrics	947
ConfusionMatrix	948
GlueTable	948
TaskRun	949
TransformFilterCriteria	949
TransformSortCriteria	950
TaskRunFilterCriteria	950
TaskRunSortCriteria	951
TaskRunProperties	951
FindMatchesTaskRunProperties	951
ImportLabelsTaskRunProperties	952
ExportLabelsTaskRunProperties	952
LabelingSetGenerationTaskRunProperties	952
SchemaColumn	952
TransformEncryption	953
MLUserDataEncryption	953
ColumnImportance	953
— operations —	954
CreateMLTransform (create_ml_transform)	954
UpdateMLTransform (update_ml_transform)	957

DeleteMLTransform (delete_ml_transform)	958
GetMLTransform (get_ml_transform)	959
GetMLTransforms (get_ml_transforms)	961
ListMLTransforms (list_ml_transforms)	961
StartMLEvaluationTaskRun (start_ml_evaluation_task_run)	962
StartMLLabelingSetGenerationTaskRun (start_ml_labeling_set_generation_task_run)	963
GetMLTaskRun (get_ml_task_run)	964
GetMLTaskRuns (get_ml_task_runs)	965
CancelMLTaskRun (cancel_ml_task_run)	966
StartExportLabelsTaskRun (start_export_labels_task_run)	966
StartImportLabelsTaskRun (start_import_labels_task_run)	967
Sensitive Data	968
— data types —	968
CustomEntityType	968
— operations —	969
CreateCustomEntityType (create_custom_entity_type)	969
DeleteCustomEntityType (delete_custom_entity_type)	970
GetCustomEntityType (get_custom_entity_type)	970
BatchGetCustomEntityTypes (batch_get_custom_entity_types)	971
ListCustomEntityTypes (list_custom_entity_types)	972
Tagging APIs	972
— data types —	972
Tag	972
— operations —	973
TagResource (tag_resource)	973
UntagResource (untag_resource)	973
GetTags (get_tags)	974
Common Data Types	974
Tag	974
DecimalNumber	975
ErrorDetail	975
PropertyPredicate	975
ResourceUri	976
ColumnStatistics	976
ColumnStatisticsError	976
ColumnError	977
ColumnStatisticsData	977
BooleanColumnStatisticsData	977
DateColumnStatisticsData	978
DecimalColumnStatisticsData	978
DoubleColumnStatisticsData	978
LongColumnStatisticsData	979
StringColumnStatisticsData	979
BinaryColumnStatisticsData	979
String Patterns	980
Exceptions	981
AccessDeniedException	981
AlreadyExistsException	981
ConcurrentModificationException	981
ConcurrentRunsExceededException	981
CrawlerNotRunningException	982
CrawlerRunningException	982
CrawlerStoppingException	982
EntityNotFoundException	982
GlueEncryptionException	982
IdempotentParameterMismatchException	983
IllegalWorkflowStateException	983

InternalServiceException	983
InvalidExecutionEngineException	983
InvalidInputException	983
InvalidStateException	984
InvalidTaskStatusTransitionException	984
JobDefinitionErrorException	984
JobRunInTerminalStateException	984
JobRunInvalidStateTransitionException	984
JobRunNotInTerminalStateException	985
LateRunnerException	985
NoScheduleException	985
OperationTimeoutException	985
ResourceNotReadyException	985
ResourceNumberLimitExceededException	986
SchedulerNotRunningException	986
SchedulerRunningException	986
SchedulerTransitioningException	986
UnrecognizedRunnerException	986
ValidationException	987
VersionMismatchException	987
Troubleshooting	988
Gathering AWS Glue Troubleshooting Information	988
Troubleshooting Connection Issues	988
Troubleshooting Errors	989
Error: Resource Unavailable	989
Error: Could Not Find S3 Endpoint or NAT Gateway for subnetId in VPC	990
Error: Inbound Rule in Security Group Required	990
Error: Outbound Rule in Security Group Required	990
Error: Job Run Failed Because the Role Passed Should Be Given Assume Role Permissions for the AWS Glue Service	990
Error: DescribeVpcEndpoints Action Is Unauthorized. Unable to Validate VPC ID vpc-id	990
Error: DescribeRouteTables Action Is Unauthorized. Unable to Validate Subnet Id: subnet-id in VPC id: vpc-id	991
Error: Failed to Call ec2:DescribeSubnets	991
Error: Failed to Call ec2:DescribeSecurityGroups	991
Error: Could Not Find Subnet for AZ	991
Error: Job Run Exception When Writing to a JDBC Target	991
Error: Amazon S3 Timeout	992
Error: Amazon S3 Access Denied	992
Error: Amazon S3 Access Key ID Does Not Exist	992
Error: Job Run Fails When Accessing Amazon S3 with an s3a:// URL	992
Error: Amazon S3 Service Token Expired	993
Error: No Private DNS for Network Interface Found	994
Error: Development Endpoint Provisioning Failed	994
Error: Notebook Server CREATE_FAILED	994
Error: Local Notebook Fails to Start	994
Error: Notebook Usage Errors	994
Error: Running Crawler Failed	995
Error: Partitions Were Not Updated	995
Error: Upgrading Athena Data Catalog	995
Error: A Job is Reprocessing Data When Job Bookmarks Are Enabled	995
AWS Glue Machine Learning Exceptions	996
CancelMLTaskRunActivity	996
CreateMLTaskRunActivity	996
DeleteMLTransformActivity	997
GetMLTaskRunActivity	997
GetMLTaskRunsActivity	998

GetMLTransformActivity	998
GetMLTransformsActivity	998
GetSaveLocationForTransformArtifactActivity	998
GetTaskRunArtifactActivity	999
PublishMLTransformModelActivity	999
PullLatestMLTransformModelActivity	999
PutJobMetadataForMLTransformActivity	1000
StartExportLabelsTaskRunActivity	1000
StartImportLabelsTaskRunActivity	1000
StartMLEvaluationTaskRunActivity	1001
StartMLLabelingSetGenerationTaskRunActivity	1001
UpdateMLTransformActivity	1002
AWS Glue Quotas	1003
Known Issues	1004
Preventing Cross-Job Data Access	1004
AWS Glue Release Notes	1006
AWS Glue Versions	1006
Running Spark ETL Jobs with Reduced Startup Times	1009
New Features Supported	1009
Logging Behavior	1011
Features Not Supported	1012
Migrating AWS Glue jobs to AWS Glue version 3.0	1012
New Features Supported	1012
Actions to migrate to AWS Glue 3.0	1013
Migration check list	1013
Migrating from AWS Glue 0.9 to AWS Glue 3.0	1014
Migrating from AWS Glue 1.0 to AWS Glue 3.0	1015
Migrating from AWS Glue 2.0 to AWS Glue 3.0	1016
Appendix A: notable dependency upgrades	1017
Appendix B: JDBC driver upgrades	1017
AWS Glue version support policy	1018
Support Policy	1018
Document History	1019
Earlier Updates	1036
AWS glossary	1037

AWS Glue and AWS Glue Studio preview features

The following features are in preview release for AWS Glue or AWS Glue Studio and are subject to change.

- AWS Glue Studio Visual Job API and [Visual Job API \(Preview\)](#) (p. 813)

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 and data streams. 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, which is a data abstraction used to organize data into rows and columns, 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 static or streaming data 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 data warehouses and data lakes and generate output streams. 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 targets.

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.

Data Sources

AWS Glue supports the following data sources:

- Data stores
 - Amazon S3
 - Amazon Relational Database Service (Amazon RDS)
 - Third-party JDBC-accessible databases
 - Amazon DynamoDB
 - MongoDB and Amazon DocumentDB (with MongoDB compatibility)
- Data streams
 - Amazon Kinesis Data Streams
 - Apache Kafka

Data Targets

AWS Glue supports the following data targets:

- Amazon S3
- Amazon Relational Database Service (Amazon RDS)
- Third-party JDBC-accessible databases
- MongoDB and Amazon DocumentDB (with MongoDB compatibility)

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 \(p. 5\)](#)
- [AWS Glue Concepts \(p. 5\)](#)
- [AWS Glue Components \(p. 8\)](#)
- [Converting Semi-Structured Schemas to Relational Schemas \(p. 10\)](#)

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.

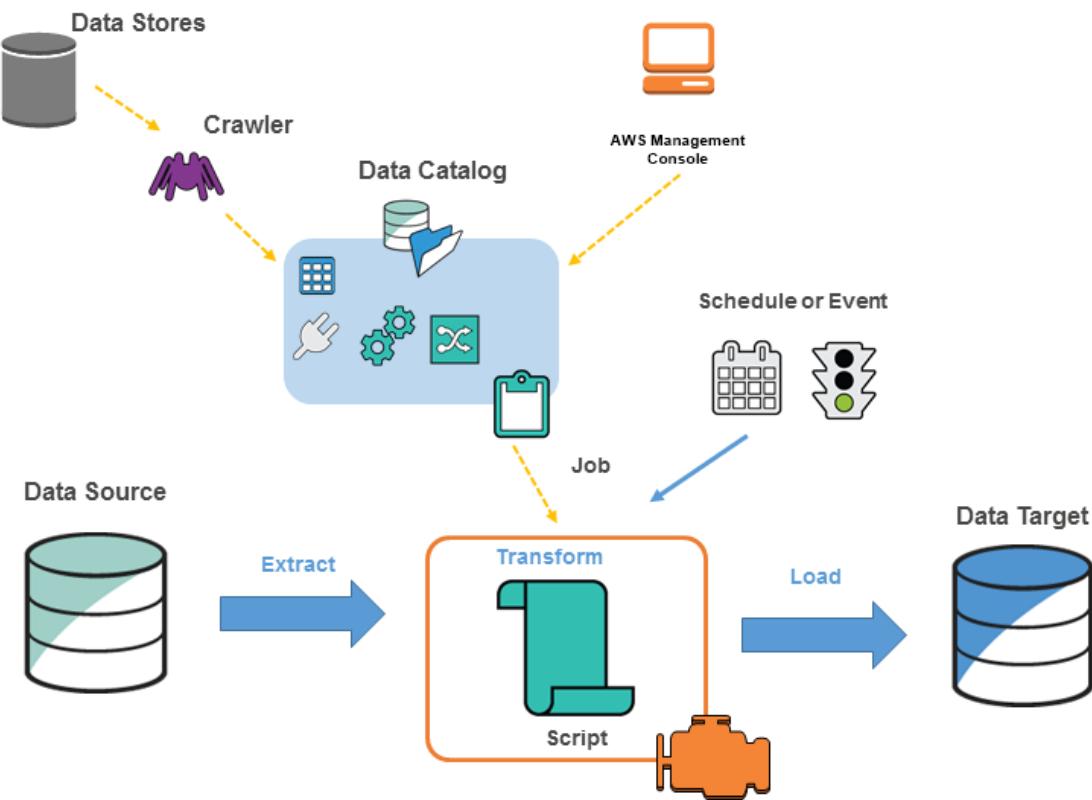
During provisioning of an ETL job, you provide input data sources and output data targets in your virtual private cloud (VPC). In addition, you provide the IAM role, VPC ID, subnet ID, and security group that are needed to access data sources and targets. For each tuple (customer account ID, IAM role,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

- For data store sources, 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. For streaming sources, you manually define Data Catalog tables and specify data stream properties.

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. It contains table definitions, job definitions, and other control information to manage your AWS Glue environment. Each AWS account has one AWS Glue Data Catalog per region.

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

A Data Catalog object that contains the properties that are required to connect to a particular 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 tables in the AWS Glue Data Catalog.

Database

A set of associated Data Catalog table definitions organized into a logical group.

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 ETL 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. PySpark is a Python dialect for ETL programming. 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.

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, partition information, 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.

Worker

With AWS Glue, you only pay for the time your ETL job takes to run. There are no resources to manage, no upfront costs, and you are not charged for startup or shutdown time. You are charged an hourly rate based on the number of **Data Processing Units** (or DPUs) used to run your ETL job. A single Data Processing Unit (DPU) is also referred to as a *worker*. AWS Glue comes with three worker types to help you select the configuration that meets your job latency and cost requirements. Workers come in Standard, G.1X, and G.2X configurations.

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 \(p. 701\)](#).

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 technical metadata store in the AWS Cloud.

Each AWS account has one AWS Glue Data Catalog per AWS Region. Each Data Catalog is a highly scalable collection of tables organized into databases. A table is metadata representation of a collection of structured or semi-structured data stored in sources such as Amazon RDS, Apache Hadoop Distributed File System, Amazon OpenSearch Service, and others. The AWS Glue Data Catalog provides a uniform repository where disparate systems can store and find metadata to keep track of data in data silos. You can then use the metadata to query and transform that data in a consistent manner across a wide variety of applications.

You use the Data Catalog together with AWS Identity and Access Management policies and Lake Formation to control access to the tables and databases. By doing this, you can allow different groups in your enterprise to safely publish data to the wider organization while protecting sensitive information in a highly granular fashion.

The Data Catalog, along with CloudTrail and Lake Formation, also provides you with comprehensive audit and governance capabilities, with schema change tracking and data access controls. This helps ensure that data is not inappropriately modified or inadvertently shared.

For information about securing and auditing the AWS Glue Data Catalog, see:

- **AWS Lake Formation** – For more information, see [What Is AWS Lake Formation?](#) in the *AWS Lake Formation Developer Guide*.
- **CloudTrail** – For more information, see [What Is CloudTrail?](#) in the *What Is CloudTrail User Guide*.

The following are other AWS services and open-source projects that 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. The AWS Glue Data Catalog can then be used to guide ETL operations.

For information about how to set up crawlers and classifiers, see [Defining Crawlers \(p. 125\)](#). For information about how to program crawlers and classifiers using the AWS Glue API, see [Crawlers and Classifiers API \(p. 782\)](#).

AWS Glue ETL Operations

Using the metadata in the Data Catalog, AWS Glue can automatically generate 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 \(p. 472\)](#).

Streaming ETL in AWS Glue

AWS Glue enables you to perform ETL operations on streaming data using continuously-running jobs. AWS Glue streaming ETL is built on the Apache Spark Structured Streaming engine, and can ingest streams from Amazon Kinesis Data Streams, Apache Kafka, and Amazon Managed Streaming for Apache Kafka (Amazon MSK). Streaming ETL can clean and transform streaming data and load it into Amazon S3 or JDBC data stores. Use Streaming ETL in AWS Glue to process event data like IoT streams, clickstreams, and network logs.

If you know the schema of the streaming data source, you can specify it in a Data Catalog table. If not, you can enable schema detection in the streaming ETL job. The job then automatically determines the schema from the incoming data.

The streaming ETL job can use both AWS Glue built-in transforms and transforms that are native to Apache Spark Structured Streaming. For more information, see [Operations on streaming DataFrames/Datasets](#) on the Apache Spark website.

For more information, see [the section called “Adding Streaming ETL Jobs” \(p. 206\)](#).

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 \(p. 300\)](#). For information about programming using the AWS Glue Jobs system API, see [Jobs API \(p. 844\)](#).

Converting Semi-Structured Schemas to Relational Schemas

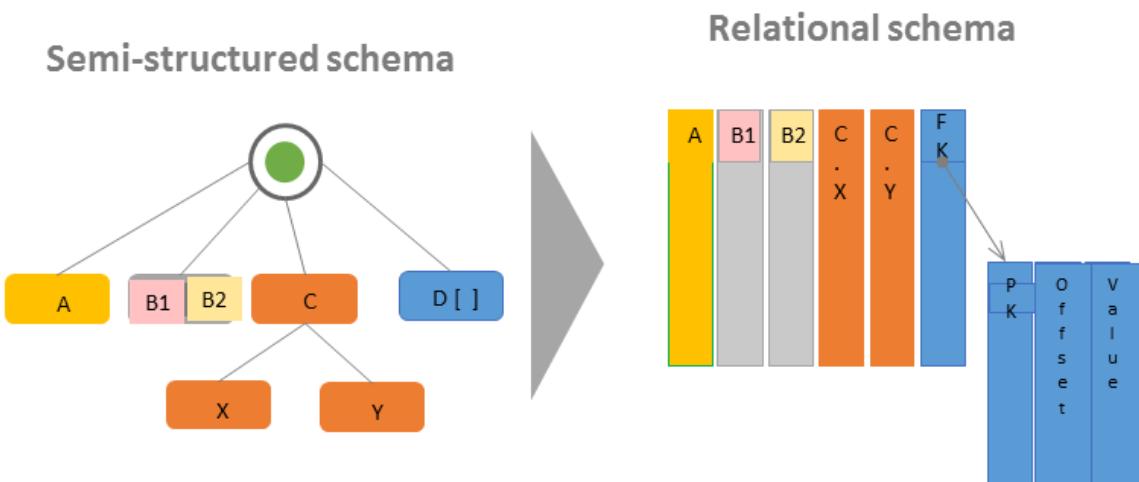
It's common to want to convert semi-structured data into relational tables. Conceptually, you are flattening a hierarchical schema to a relational schema. AWS Glue can perform this conversion for you on-the-fly.

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.

Setting up AWS Glue

The following sections provide information on setting up AWS Glue. Not all of the setting up sections are required to start using AWS Glue. You may use the instructions as needed to set up IAM permissions, encryption, DNS if using a VPC, Environment to access data stores or if using Interactive Sessions.

Topics

- [Setting up IAM Permissions for AWS Glue \(p. 12\)](#)
- [Setting Up DNS in Your VPC \(p. 31\)](#)
- [Setting Up Your Environment to Access Data Stores \(p. 32\)](#)
- [Setting Up Encryption in AWS Glue \(p. 36\)](#)
- [Setting Up Your Environment for Development Endpoints \(p. 39\)](#)

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 \(p. 12\)](#): Create a service policy that allows access to AWS Glue resources.
2. [Create an IAM Role for AWS Glue \(p. 16\)](#): 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 \(p. 17\)](#): Attach policies to any IAM user that signs in to the AWS Glue console.
4. [Create an IAM Policy for Notebooks \(p. 25\)](#): Create a notebook server policy to use in the creation of notebook servers on development endpoints.
5. [Create an IAM Role for Notebooks \(p. 28\)](#): Create an IAM role and attach the notebook server policy.
6. [Create an IAM Policy for Amazon SageMaker Notebooks \(p. 29\)](#): Create an IAM policy to use when creating Amazon SageMaker notebooks on development endpoints.
7. [Create an IAM Role for Amazon SageMaker Notebooks \(p. 30\)](#): 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",
                "s3:PutBucketPublicAccessBlock"
            ],
            "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", "s3:PutBucketPublicAccessBlock"	"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- . Enables AWS Glue to create buckets that block public access.
"s3GetObject", "s3:PutObject", "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", "logs:PutLogEvents"	"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 **s3GetObject** permissions. Data targets require **s3>ListBucket**, **s3PutObject**, and **s3DeleteObject** 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:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::mybucket/*"
      ]
    }
  ]
}
```

```
        "Action": [
            "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.

Per security best practices, it is recommended to restrict access by tightening policies to further restrict access to Amazon S3 bucket and Amazon CloudWatch log groups. For an example Amazon S3 policy, see [Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket](#).

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>ListUsers",  
                "iam>ListGroups",  
                "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",  
                "rds:DescribeDBClusters",  
                "rds:DescribeDBSubnetGroups",  
                "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-*"  
            ]  
        }  
    ]  
}
```

```

},
{
    "Effect": "Allow",
    "Action": [
        "tag:GetResources"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:CreateBucket",
        "s3:PutBucketPublicAccessBlock"
    ],
    "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", "s3:PutBucketPublicAccessBlock"	"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- . Enables AWS Glue to create buckets that block public access.
"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:*::*: aws-glue*/*"	Allows managing AWS CloudFormation stacks when working with notebook servers. Naming convention: AWS Glue creates stacks whose names begin with aws-glue- .

Action	Resource	Description
"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.
"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**.

3. 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.
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 **AWSGlueConsoleSageMakerNotebookFullAccess** managed policy

You can attach the **AWSGlueConsoleSageMakerNotebookFullAccess** policy to a user to manage 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 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 **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:ListTables"  
            ]  
        }  
    ]  
}
```

```
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:GetTable",
        "glue:GetTableVersions",
        "glue:GetTables",
        "glue:UpdateDatabase",
        "glue:UpdatePartition",
        "glue:UpdateTable",
        "glue:GetJobBookmark",
        "glue:ResetJobBookmark",
        "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:DescribeTags"
    ],
    "Resource": [
        "arn:aws:ec2:*
```

```
        "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.
"s3:GetObject"	"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- .
"s3:PutObject", "s3>DeleteObject"	"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 .
"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 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 SageMaker Notebooks

If you plan to use 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 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:dev-endpoint/*"  
            ]  
        }  
    ]  
}
```

```

        "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 SageMaker notebooks.
"logs>CreateLogStream", "logs>DescribeLogStreams", "logs>PutLogEvents", "logs>CreateLogGroup"	"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-*"	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:region- code:account- id:devEndpoint/*"	Grants permission to use a development endpoint from SageMaker notebooks.
"sagemaker>ListTags"	"arn:aws:sagemaker:region- code:account-id:notebook- instance/*"	Grants permission to return tags for an SageMaker resource. The aws-glue-dev-endpoint tag is required on the SageMaker notebook for connecting the notebook to a development endpoint.

- 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 SageMaker Notebooks

If you plan to use 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 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 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 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](#). Also, you can use the AWS CLI and call the `modify-vpc-attribute` command to configure the VPC network attributes.

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 \(p. 34\)](#).

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 \(p. 32\)](#) 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 \(p. 31\)](#).

Topics

- [Amazon VPC Endpoints for Amazon S3 \(p. 32\)](#)
- [Setting Up a VPC to Connect to JDBC Data Stores \(p. 34\)](#)

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.

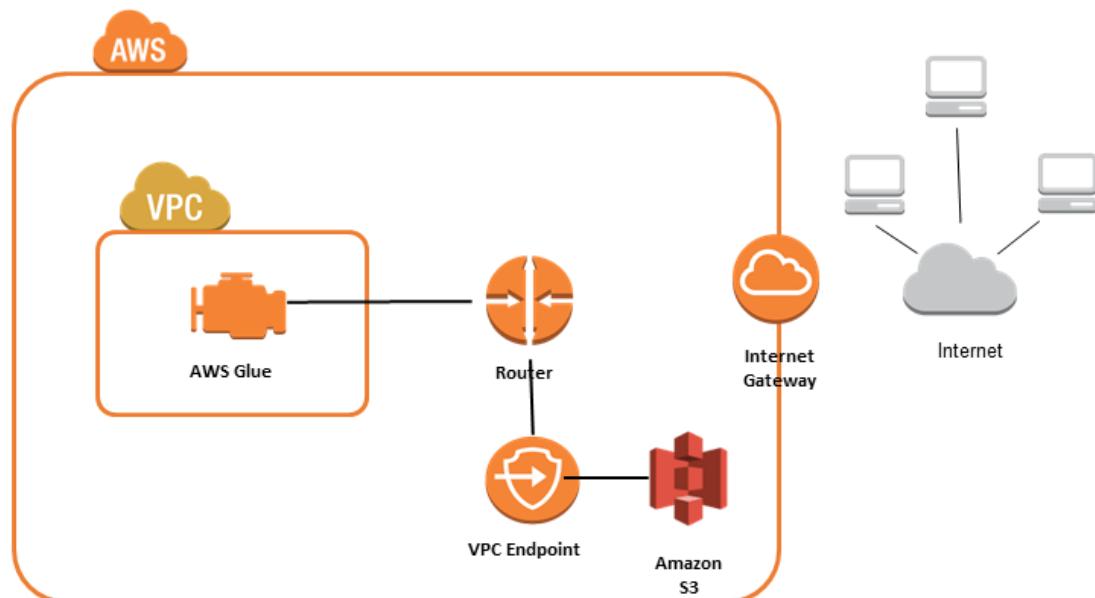
Many customers have legitimate privacy and security concerns about sending and receiving data across the public internet. Customers can address these concerns by using a virtual private network (VPN) to route all Amazon S3 network traffic through their own corporate network infrastructure. However, this 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 Amazon 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 VPC endpoint of type Gateway.

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.

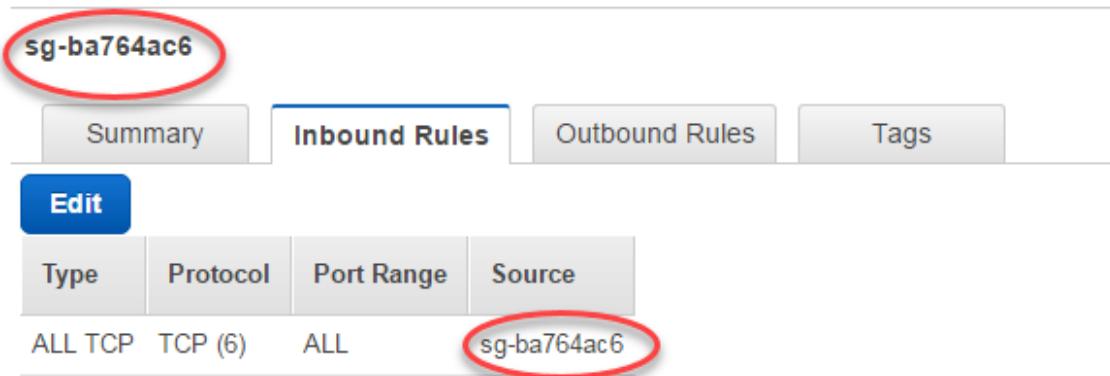
To set up access for Amazon Redshift data stores

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	<i>database-security-group</i>

For example:



- 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>

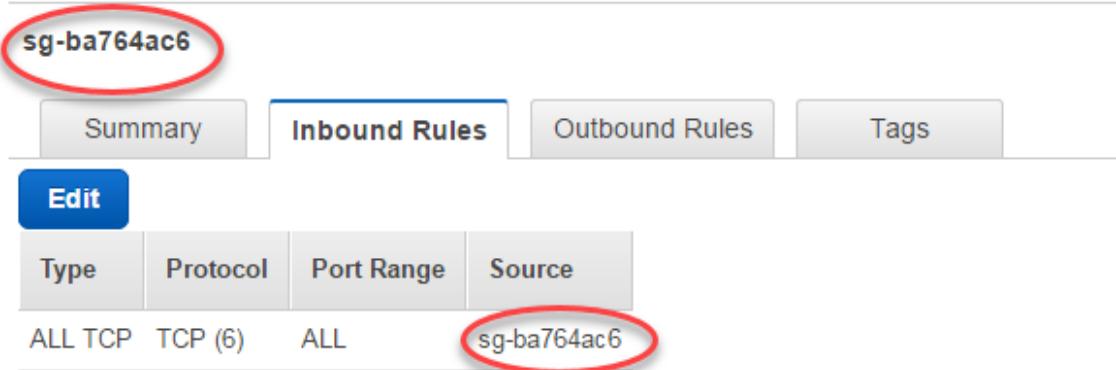
To set up access for Amazon RDS data stores

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
- In the left navigation pane, choose **Instances**.
- Choose the Amazon RDS **Engine** and **DB Instance** name that you want to access from AWS Glue.
- 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.
- Choose the security group to open the Amazon EC2 console.
- Confirm that your **Group ID** from Amazon RDS is chosen, then choose the **Inbound** tab.
- 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	database-security-group

For example:



- 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 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.

- 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: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 AWS 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 Glue 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 AWS 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.

10 In 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.

11 On 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.

Setting Up Your Environment for Development Endpoints

To run your extract, transform, and load (ETL) scripts with AWS Glue, you can develop and test your scripts using a *development endpoint*. Development endpoints are not supported for use with AWS Glue version 2.0 jobs. For versions 2.0 and later, the preferred development method is using Jupyter Notebook with one of the AWS Glue kernels. For more information, see [the section called "Getting started with AWS Glue interactive sessions" \(p. 231\)](#).

Setting Up Your Network for 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 \(p. 31\)](#).

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:

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:

Security Group: sg-19e1b768

Description Inbound Outbound Tags

Edit

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

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 \(p. 4\)](#).

Topics

- [AWS Glue Console Workflow Overview \(p. 42\)](#)
- [Getting Started with the AWS Glue Data Catalog \(p. 43\)](#)

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 or data lake. The following steps describe the general workflow and some of the choices that you make when working with AWS Glue.

Note

You can follow the steps below, or you can create a workflow that automatically performs steps 1 through 3. For more information, see [Performing Complex ETL Activities Using Blueprints and Workflows \(p. 373\)](#).

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

In the console, for persistent data stores, 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. For data streams, you can manually create the table definition, and define stream properties.

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 \(p. 157\)](#).

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 \(p. 102\)](#).

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:

- Choose 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 source and interpret the format of your data.
- Choose 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 \(p. 196\)](#).

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 [Starting Jobs and Crawlers Using Triggers \(p. 296\)](#).

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 \(p. 300\)](#).

Getting Started with the AWS Glue Data Catalog

The AWS Glue Data Catalog is your persistent technical metadata store. It is a managed service that you can use to store, annotate, and share metadata in the AWS Cloud. For more information, see [AWS Glue Data Catalog](#).

Overview

You can use this tutorial to create your first AWS Glue Data Catalog, which uses an Amazon S3 bucket as your data source.

In this tutorial, you'll do the following:

1. Create a database
2. Create a table

After completing these steps, you will have successfully used an Amazon S3 bucket as the data source to populate the AWS Glue Data Catalog.

Step 1: Create a Database

To get started, sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue.html>.

1. In the AWS Glue console, choose **Databases** from the left-hand menu.
2. Choose **Add database**.

3. Enter a name for the database. For this tutorial, you can name the database 'My First Database'. In **Description and location (optional)**, you can optionally add a database description and location.

Congratulations, you've just set up your first database using the AWS Glue console. Your new database will appear in the list of available databases. You can edit the database by choosing the database's name from the **Databases** dashboard.

Next Steps

In the next section, you'll create a table and add that table to your database.

You can also explore the settings and permissions for your Data Catalog. See [Working with Data Catalog Settings in the AWS Glue Console](#).

You just created a database using the AWS Glue console, but there are other ways to create a database:

- You can use crawlers to create a database and tables for you automatically. To set up a database using crawlers, see [Working with Crawlers in the AWS Glue Console](#).
- You can use AWS CloudFormation templates. See [Creating AWS Glue Resources Using AWS Glue Data Catalog Templates](#).
- You can also create a database using the AWS Glue Database API operations.

To create a database using the `create` operation, structure the request by including the `DatabaseInput` (required) parameters.

For example:

CLI

```
aws glue create-database --database-input "{\"Name\":\"clidb\"}"
```

Boto3

```
glueClient = boto3.client('glue')

response = glueClient.create_database(
    DatabaseInput={
        'Name': 'boto3db'
    }
)
```

For more information about the Database API data types, structure, and operations, see [Database API](#).

Step 2. Create a Table

In this step, you create a table using the AWS Glue console.

1. In the AWS Glue console, choose **Tables** in the left-hand menu.
2. Choose **Add tables**. From the drop-down menu, choose **Add tables manually**.
3. In the **Add table** wizard, set up your table's properties by entering a name for your table.
4. In the **Database** section, choose the database that you created in Step 1 ('My First Database') from the drop-down menu. Choose **Next**.

5. In **Add a data store**, choose **S3** as the type of source.
6. In the section **Data is located in**, choose **Specified path in another account**.
 - a. Copy and paste the path for the **Include path** section.
`s3://crawler-public-us-west-2/flight/2016/csv/`
 - b. Choose **Next**.
7. For **Classification**, choose **CSV**. and for **Delimiter**, choose **comma (,)**. Choose **Next**.
8. You are asked to define a schema. A schema defines the structure and format of a data record. Choose **Add column**. (For more information, see See [Schema registries](#)).
9. Specify the column properties:
 - a. Enter a column name.
 - b. For **Column type**, 'string' is already selected by default.
 - c. For **Column number**, '1' is already selected by default.
 - d. Choose **Add**.
10. You are asked to add partition indexes. This is optional. To skip this step, choose **Next**.
11. A summary of the table properties is displayed. If everything looks as expected, choose **Finish**. Otherwise, choose **Back** and make edits as needed.

Congratulations, you've successfully created a table manually and associated it to a database. Your newly created table will appear in the Tables dashboard. From the dashboard, you can modify and manage all your tables.

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

Next steps

Now that the Data Catalog is populated, you can begin authoring jobs in AWS Glue. See [Authoring Jobs](#).

In addition to using the console, there are other ways to define tables in the Data Catalog including:

- [Creating and running a crawler](#)
- [Using the AWS Glue Table API](#)
- [Using the AWS Glue Data Catalog template](#)
- [Migrating an Apache Hive metastore](#)
- [Using the AWS CLI, Boto3, or data definition language \(DDL\)](#)

The following are examples of how you can use the CLI, Boto3, or DDL to define a table based on the same flights_data.csv file from the S3 bucket that you used in the tutorial.

CLI

```
{  
    "Name": "flights_data_cli",  
    "StorageDescriptor": {  
        "Columns": [  
            {  
                "Name": "year",  
                "Type": "bigint"  
            },  
            {  
                "Name": "quarter",  
                "Type": "bigint"  
            }  
        ],  
        "Location": "s3://crawler-public-us-west-2/flight/2016/csv",  
        "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",  
        "OutputFormat": "org.apache.hadoop.mapred.TextOutputFormat"  
    }  
}
```

```
        "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
        "OutputFormat":
    "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
        "Compressed": false,
        "NumberOfBuckets": -1,
        "SerdeInfo": {
            "SerializationLibrary":
    "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
            "Parameters": {
                "field.delim": ",",
                "serialization.format": ","
            }
        }
    },
    "PartitionKeys": [
        {
            "Name": "mon",
            "Type": "string"
        }
    ],
    "TableType": "EXTERNAL_TABLE",
    "Parameters": {
        "EXTERNAL": "TRUE",
        "classification": "csv",
        "columnsOrdered": "true",
        "compressionType": "none",
        "delimiter": ",",
        "skip.header.line.count": "1",
        "typeOfData": "file"
    }
}
```

Boto3

```
import boto3

glue_client = boto3.client("glue")

response = glue_client.create_table(
    DatabaseName='sampledb',
    TableInput={
        'Name': 'flights_data_manual',
        'StorageDescriptor': {
            'Columns': [
                {'Name': 'year',
                 'Type': 'bigint'},
                {'Name': 'quarter',
                 'Type': 'bigint'}
            ],
            'Location': 's3://crawler-public-us-west-2/flight/2016/csv',
            'InputFormat': 'org.apache.hadoop.mapred.TextInputFormat',
            'OutputFormat':
    'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat',
            'Compressed': False,
            'NumberOfBuckets': -1,
            'SerdeInfo': {
                'SerializationLibrary':
    "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                'Parameters': {
                    'field.delim': ',',
                    'serialization.format': ','
                }
            }
        }
    }
)
```

```
        },
    },
    'PartitionKeys': [
        {
            'Name': 'mon',
            'Type': 'string'
        }],
    'TableType': 'EXTERNAL_TABLE',
    'Parameters': {
        'EXTERNAL': 'TRUE',
        'classification': 'csv',
        'columnsOrdered': 'true',
        'compressionType': 'none',
        'delimiter': ',',
        'skip.header.line.count': '1',
        'typeOfData': 'file'
    }
}
)
```

DDL

```
CREATE EXTERNAL TABLE `sampledb`.`flights_data` (
    `year` bigint,
    `quarter` bigint)
PARTITIONED BY (
    `mon` string)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
    'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
    'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
    's3://crawler-public-us-west-2/flight/2016/csv/'
TBLPROPERTIES (
    'classification'='csv',
    'columnsOrdered'='true',
    'compressionType'='none',
    'delimiter'=',',
    'skip.header.line.count'='1',
    'typeOfData'='file')
```

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 \(p. 48\)](#)
- [Identity and access management in AWS Glue \(p. 56\)](#)
- [Logging and Monitoring in AWS Glue \(p. 96\)](#)
- [Compliance Validation for AWS Glue \(p. 96\)](#)
- [Resilience in AWS Glue \(p. 97\)](#)
- [Infrastructure Security in AWS Glue \(p. 97\)](#)

Data Protection in AWS Glue

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

Topics

- [Encryption at Rest \(p. 48\)](#)
- [Encryption in Transit \(p. 54\)](#)
- [FIPS Compliance \(p. 55\)](#)
- [Key Management \(p. 55\)](#)
- [AWS Glue Dependency on Other AWS Services \(p. 55\)](#)
- [Development Endpoints \(p. 55\)](#)

Encryption at Rest

AWS Glue supports data encryption at rest for [Authoring Jobs in AWS Glue \(p. 196\)](#) and [Developing Scripts Using Development Endpoints \(p. 252\)](#). 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 \(p. 9\)](#) 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 turn on 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 turned on, 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 Transport Layer Security (TLS) 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 \(p. 49\)](#)
- [Encrypting Connection Passwords \(p. 51\)](#)
- [Encrypting Data Written by Crawlers, Jobs, and Development Endpoints \(p. 51\)](#)

Encrypting Your Data Catalog

You can turn on encryption of your AWS Glue Data Catalog objects in the **Settings** of the Data Catalog on the AWS Glue console. You can turn on or turn off 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 turn on 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 AWS 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 turned on, 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 turn on 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 turned on, 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 turn on 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 AWS 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 \(p. 173\)](#).

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 \(p. 53\)](#).

Topics

- [Setting Up AWS Glue to Use Security Configurations \(p. 52\)](#)
- [Creating a Route to AWS KMS for VPC Jobs and Crawlers \(p. 52\)](#)
- [Working with Security Configurations on the AWS Glue Console \(p. 53\)](#)

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 [Defining Connections in the AWS Glue Data Catalog \(p. 110\)](#).

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

CloudWatch logs encryption mode

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

Job bookmark encryption mode

If turned on, the Amazon S3 encryption mode such as SSE-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 \(p. 51\)](#).

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 allow 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 AWS 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 turn it on, 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 turn it on, 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 User 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 Transport Layer Security (TLS) 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 turn on 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.

FIPS Compliance

If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

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.

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.
- AWS CloudFormation 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 \(p. 56\)](#)
- [Managing access permissions for AWS Glue resources \(p. 57\)](#)
- [Permissions required to use the AWS Glue console \(p. 61\)](#)
- [Identity-based policies \(IAM policies\) for access control \(p. 61\)](#)
- [AWS Glue resource policies for access control \(p. 68\)](#)
- [Granting cross-account access \(p. 71\)](#)
- [Specifying AWS Glue resource ARNs \(p. 76\)](#)
- [AWS Glue access control policy examples \(p. 81\)](#)
- [AWS Glue API permissions: actions and resources reference \(p. 95\)](#)

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 on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
 - **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.

Note

You can grant access to your data by using AWS Glue methods or by using AWS Lake Formation grants. The AWS Glue methods use AWS Identity and Access Management (IAM) policies to achieve fine-grained access control. Lake Formation uses a simpler GRANT/REVOKE permissions model similar to the GRANT/REVOKE commands in a relational database system.

This section describes using the AWS Glue methods. For information about using Lake Formation grants, see [Granting Lake Formation Permissions](#) in the *AWS Lake Formation Developer Guide*.

Topics

- [Using permissions policies to manage access to resources \(p. 58\)](#)
- [AWS Glue resources and operations \(p. 58\)](#)
- [Understanding resource ownership \(p. 58\)](#)
- [Managing access to resources \(p. 59\)](#)

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 \(p. 95\)](#).

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 \(p. 61\)](#)
- [AWS Glue resource policies for access control \(p. 68\)](#)

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 \(p. 91\)](#).

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 \(p. 701\)](#).

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

- [Overview of identity-based policies \(IAM policies\) \(p. 59\)](#)
- [Overview of resource-based policies \(p. 60\)](#)
- [Specifying policy elements: Actions, effects, and principals \(p. 60\)](#)
- [Specifying conditions in a policy \(p. 60\)](#)

Overview of 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 \(p. 61\)](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

Overview of 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.

For more information and examples, see [AWS Glue resource policies for access control \(p. 68\)](#).

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 \(p. 58\)](#) and [AWS Glue API \(p. 701\)](#).

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 \(p. 58\)](#).
- **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 \(p. 95\)](#).

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*.

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 \(p. 17\)](#).

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

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.

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 \(p. 71\)](#).

Topics

- [Identity-based policy examples \(p. 62\)](#)
- [Identity-based policies \(IAM policies\) with tags \(p. 63\)](#)
- [Identity-Based Policies \(IAM Policies\) that Control Settings Using Condition Keys or Context Keys \(p. 63\)](#)
- [Resource-level permissions only apply to specific AWS Glue objects \(p. 65\)](#)
- [Permissions Required to Use the AWS Glue Console \(p. 61\)](#)
- [AWS Managed \(Predefined\) Policies for AWS Glue \(p. 66\)](#)

- AWS Glue updates to AWS managed policies (p. 67)

Identity-based policy examples

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 `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 \(p. 81\)](#).

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 \(p. 310\)](#).

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 \(p. 95\)](#).

For information about how to control access using tags, see [Examples of AWS Glue identity-based \(IAM\) access control policies with tags \(p. 86\)](#).

Identity-Based Policies (IAM Policies) that Control Settings Using Condition Keys or Context Keys

You can use condition keys or context keys when granting permissions to create and update jobs. These sections discuss the keys:

- [IAM Policies that Control Settings Using Condition Keys \(p. 64\)](#)

- [IAM Policies that Control Settings Using Context Keys \(p. 64\)](#)

IAM Policies that Control Settings Using Condition Keys

AWS Glue provides three IAM condition keys `glue:VpcIds`, `glue:SubnetIds` and, `glue:SecurityGroupIds`. You can use the condition keys in IAM policies when granting permissions to create and update jobs. You can use this setting to ensure that jobs are not created (or updated to) to run outside of a desired VPC environment. The VPC setting information is not a direct input from the `CreateJob` request, but inferred from the Job "connections" field which points to an AWS Glue connection.

Example usage

Create an AWS Glue Network type connection named "traffic-monitored-connection" with the desired VpcId "vpc-id1234", SubnetIds, and SecurityGroupIds.

Specify the condition keys condition for the `CreateJob` and `UpdateJob` action in the IAM policy.

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateJob",
    "glue:UpdateJob"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "ForAnyValue:StringLike": {
      "glue:VpcIds": [
        "vpc-id1234"
      ]
    }
  }
}
```

You can create a similar IAM policy to prohibit creating an AWS Glue job without specifying connection information.

IAM Policies that Control Settings Using Context Keys

AWS Glue provides a context key (`glue:CredentialIssuingService= glue.amazonaws.com`) to each role session that AWS Glue makes available to the job and developer endpoint. This allows you to implement security controls for the actions taken by AWS Glue scripts. AWS Glue provides another context key (`glue:RoleAssumedBy=glue.amazonaws.com`) to each role session where AWS Glue makes a call to another AWS service on the customer's behalf (not by a job/dev endpoint but directly by the AWS Glue service).

Example usage

Specify the conditional permission in IAM policy and attach it to the role to be used by an AWS Glue job. This ensures certain actions are allowed/denied based on whether the role session is used for an AWS Glue job execution environment.

```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::confidential-bucket/*",
  "Condition": {
```

```

        "StringEquals": {
            "glue:CredentialIssuingService": "glue.amazonaws.com"
        }
    }
}

```

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/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 \(p. 76\)](#).

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 \(p. 17\)](#).

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

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.
- [`AwsGlueSessionUserRestrictedServiceRole`](#) – Provides full access to all AWS Glue resources except for sessions. Allows users to create and use only the interactive sessions that are associated with the user. This policy also includes other permissions needed by AWS Glue to manage Glue resources in other AWS services. The policy also allows adding tags to AWS Glue resources in other AWS services.

Note

To achieve the full security benefits, do not grant this policy to a user that was assigned the `AWSGlueServiceRole`, `AWSGlueConsoleFullAccess`, or `AWSGlueConsoleSageMakerNotebookFullAccess` policy.

- [`AwsGlueSessionUserRestrictedPolicy`](#) – Provides access to create AWS Glue interactive sessions using the `CreateSession` API only if a tag key "owner" and value matching the assignee's AWS user ID is provided. This identity policy is attached to the IAM user that invokes the `CreateSession` API. This policy also permits the assignee to interact with the AWS Glue interactive session resources that were created with a "owner" tag and value matching their AWS user id. This policy denies permission to change or remove "owner" tags from an AWS Glue session resource after the session is created.

Note

To achieve the full security benefits, do not grant this policy to a user that was assigned the `AWSGlueServiceRole`, `AWSGlueConsoleFullAccess`, or `AWSGlueConsoleSageMakerNotebookFullAccess` policy.

- [`AwsGlueSessionUserRestrictedNotebookServiceRole`](#) – Provides sufficient access to the AWS Glue Studio Notebook session to interact with the AWS Glue interactive session resources that are created with the "owner" tag value matching the AWS user id of the principal (IAM user or Role) that creates the Notebook. For more information about these tags, see the [Principal key values](#) chart in the *IAM User Guide*.

This service-role policy is attached to the role that is specified with a magic statement within the notebook or passed as a role to the `CreateSession` API. This policy also permits the principal to create an AWS Glue interactive session from the AWS Glue Studio Notebook interface only if a tag key "owner" and value matching the AWS user id of the principal. This policy denies permission to change or remove "owner" tags from a AWS Glue session resource after the session is created. This policy also includes permissions for writing and reading from Amazon S3 buckets, writing CloudWatch logs, creating and deleting tags for Amazon EC2 resources used by AWS Glue.

Note

To achieve the full security benefits, do not grant this policy to a role that was assigned the `AWSGlueServiceRole`, `AWSGlueConsoleFullAccess`, or `AWSGlueConsoleSageMakerNotebookFullAccess` policy.

- [AwsGlueSessionUserRestrictedNotebookPolicy](#) – Provides access to create an AWS Glue interactive session from the AWS Glue Studio Notebook interface only if there is a tag key "owner" and value matching the AWS user id of the principal (IAM user or Role) that creates the Notebook. For more information about these tags, see the [Principal key values](#) chart in the [IAM User Guide](#).

This policy is attached to the principal (IAM user or role) that creates sessions from the AWS Glue Studio Notebook interface. This policy also permits sufficient access to the AWS Glue Studio Notebook to interact with the AWS Glue interactive session resources that are created with the "owner" tag value matching the AWS user id of the principal. This policy denies permission to change or remove "owner" tags from an AWS Glue session resource after the session is created.

- [AWSGlueServiceNotebookRole](#) – Grants access to AWS Glue sessions started in a notebook created in AWS Glue Studio. This policy allows listing and getting session information for all sessions, but only permits users to create and use the sessions tagged with their AWS user id. This policy denies permission to change or remove "owner" tags from AWS Glue session resources tagged with their AWS id.

Assign this policy to the AWS user who creates jobs using the notebook interface in AWS Glue Studio.

- [AWSGlueConsoleSageMakerNotebookFullAccess](#) – Grants full access to AWS Glue and 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 SageMaker notebooks.
- [AWSGlueSchemaRegistryFullAccess](#) – Grants full access to AWS Glue Schema Registry resources when using the AWS Management Console or AWS CLI. 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 or AWS CLI who manage Schema Registry.
- [AWSGlueSchemaRegistryReadonlyAccess](#) – Grants read-only access to AWS Glue Schema Registry resources when using the AWS Management Console or AWS CLI. 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 or AWS CLI who use Schema Registry.

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 updates to AWS managed policies

View details about updates to AWS managed policies for AWS Glue since this service began tracking these changes. For automatic alerts about these updates, subscribe to the RSS feed on the [AWS Glue Document history \(p. 1019\)](#) page.

Change	Description	Date
New managed policies added for the Interactive Sessions feature <ul style="list-style-type: none">• AwsGlueSessionUserRestrictedServiceRole• AwsGlueSessionUserRestrictedPolicy• AwsGlueSessionUserRestrictedNotebookServiceRole• AwsGlueSessionUserRestrictedNotebookPolicy	These policies were designed to provide additional security for interactive sessions and notebooks in AWS Glue Studio. These policies restrict access to the CreateSession API so that only the owner has access.	November 30, 2021
AWSGlueConsoleSageMakerNotebookFullAccess – Update to an existing policy	Removed a redundant resource ARN (<code>arn:aws:s3:::aws-glue-*/*</code>) for the action that grants read/write permissions on Amazon S3 buckets that AWS Glue uses to store scripts and temporary files. Fixed a syntax issue by changing " <code>StringEquals</code> " to " <code>ForAnyValue:StringLike</code> ", and moved the " <code>Effect</code> ": " <code>Allow</code> " lines to precede the " <code>Action</code> ": line in each place where they were out of order.	July 15, 2021
AWSGlueConsoleFullAccess – Update to an existing policy	Removed a redundant resource ARN (<code>arn:aws:s3:::aws-glue-*/*</code>) for the action that grants read/write permissions on Amazon S3 buckets that AWS Glue uses to store scripts and temporary files.	July 15, 2021
AWS Glue started tracking changes	AWS Glue started tracking changes for its AWS managed policies.	June 10, 2021

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.

Overview of resource policies and syntax

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.

Examples of resource policies

See Also:

- [the section called "Adding or updating the Data Catalog resource policy" \(p. 73\)](#)

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/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:s3:::mybucket/*"
      ]
    }
  ]
}
```

```
    "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 \(put_resource_policy\) \(p. 71\)](#)
- [GetResourcePolicy \(get_resource_policy\) \(p. 718\)](#)
- [DeleteResourcePolicy \(delete_resource_policy\) \(p. 718\)](#)

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 \(p. 173\)](#).

Granting cross-account access

Granting access to Data Catalog resources across accounts enables your extract, transform, and load (ETL) jobs to query and join data from different accounts.

Topics

- [Methods for granting cross-account access in AWS Glue \(p. 71\)](#)
- [Adding or updating the Data Catalog resource policy \(p. 73\)](#)
- [Making a cross-account API call \(p. 74\)](#)
- [Making a cross-account ETL call \(p. 74\)](#)
- [Cross-account CloudTrail logging \(p. 75\)](#)
- [AWS Glue resource ownership and operations \(p. 58\)](#)
- [Cross-account resource ownership and billing \(p. 75\)](#)
- [Cross-account access limitations \(p. 76\)](#)

Methods for granting cross-account access in AWS Glue

You can grant access to your data to external AWS accounts by using AWS Glue methods or by using AWS Lake Formation cross-account grants. The AWS Glue methods use AWS Identity and Access Management (IAM) policies to achieve fine-grained access control. Lake Formation uses a simpler GRANT/REVOKE permissions model similar to the GRANT/REVOKE commands in a relational database system.

This section describes using the AWS Glue methods. For information about using Lake Formation cross-account grants, see [Granting Lake Formation Permissions](#) in the *AWS Lake Formation Developer Guide*.

There are two AWS Glue methods for granting cross-account access to a resource:

- Use a Data Catalog resource policy
- Use an IAM role

Granting cross-account access using a resource policy

The following are the general steps for granting cross-account access using a Data Catalog resource policy:

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.

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.

Granting cross-account access using an IAM role

The following are the general steps for granting cross-account access using an IAM role:

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"  
      ]  
    }  
  ]  
}
```

```
        "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"
      ]
    }
  ]
}
```

Adding or updating the Data Catalog resource policy

You can add or update the AWS Glue Data Catalog resource policy using the console, API, or AWS Command Line Interface (AWS CLI).

Important

If you have already made cross-account permission grants from your account with AWS Lake Formation, adding or updating the Data Catalog resource policy requires an extra step. For more information, see [Managing Cross-Account Permissions Using Both AWS Glue and Lake Formation](#) in the *AWS Lake Formation Developer Guide*.

To determine if Lake Formation cross-account grants exist, use the `glue:GetResourcePolicies` API or AWS CLI. If the API returns any policies other than an already existing Data Catalog policy, then Lake Formation grants exist. For more information, see [Viewing All Cross-Account Grants Using the GetResourcePolicies API](#) in the *AWS Lake Formation Developer Guide*.

To add or update the Data Catalog resource policy (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.

Sign in as an AWS Identity and Access Management (IAM) administrative user or as a user who has the `glue:PutResourcePolicy` permission.

2. In the navigation pane, choose **Settings**.
3. On the **Data catalog settings** page, under **Permissions**, paste a resource policy into the text area. Then choose **Save**.

If the console displays a alert stating that the permissions in the policy will be in addition to any permissions granted using Lake Formation, choose **Proceed**.

To add or update the Data Catalog resource policy (AWS CLI)

- Submit an `aws glue put-resource-policy` command. If Lake Formation grants already exist, ensure that you include the `--enable-hybrid` option with the value '`TRUE`'.

For examples of using this command, see [AWS Glue resource-based access control policy examples \(p. 91\)](#).

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 \(p. 575\)](#). For Scala APIs that support `catalogId`, see [AWS Glue Scala GlueContext APIs \(p. 656\)](#).

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.

Cross-account CloudTrail logging

When an AWS Glue extract, transform, and load (ETL) job accesses the underlying data of a Data Catalog table shared through AWS Lake Formation cross-account grants, there is additional AWS CloudTrail logging behavior.

For purposes of this discussion, the AWS account that shared the table is the owner account, and the account that the table was shared with is the recipient account. When an ETL job in the recipient account accesses data in the table in the owner account, the data-access CloudTrail event that is added to the logs for the recipient account gets copied to the owner account's CloudTrail logs. This is so owner accounts can track data accesses by the various recipient accounts. By default, the CloudTrail events do not include a human-readable principal identifier (principal ARN). An administrator in the recipient account can opt in to include the principal ARN in the logs.

For more information, see [Cross-Account CloudTrail Logging](#) in the *AWS Lake Formation Developer Guide*.

See Also

- [the section called "Logging and Monitoring" \(p. 96\)](#)

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 you created databases and tables using Amazon Athena or Amazon Redshift Spectrum prior to a region's support for AWS Glue and 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 \(get_catalog_import_status\) \(p. 782\)](#). 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.
- Cross-account access to the Data Catalog from Athena requires you to register the catalog as an Athena DataCatalog resource. For instructions, see [Registering an AWS Glue Data Catalog from Another Account](#) in the *Amazon Athena User Guide*.

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 \(p. 76\)](#)
- [ARNs for non-catalog objects in AWS Glue \(p. 78\)](#)
- [Access control for AWS Glue non-catalog singular API operations \(p. 79\)](#)
- [Access control for AWS Glue non-catalog API operations that retrieve multiple items \(p. 80\)](#)
- [Access control for AWS Glue non-catalog BatchGet API operations \(p. 80\)](#)

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>

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

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:<region>:<account-id>:crawler/<crawler-name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:crawler/mycrawler</code>
Job	<code>arn:aws:glue:<region>:<account-id>:job/<job-name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:job/testjob</code>
Trigger	<code>arn:aws:glue:<region>:<account-id>:trigger/<trigger-name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger</code>
Development endpoint	<code>arn:aws:glue:<region>:<account-id>:devEndpoint/<development-endpoint-name></code> For example: <code>arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint</code>
Machine learning transform	<code>arn:aws:glue:<region>:<account-id>:mlTransform/<transform-id></code>

Resource Type	ARN Format
	For example: arn:aws:glue:us-east-1:123456789012:mlTransform/tfm-1234567890

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": "MinimumPermissions",
            "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": "PermissionWithWildcard",
            "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": "CombinedPermissions",
            "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, `GetDevEndpoints`. For this operation, you can specify only a wildcard (*) resource, and not specific ARNs.

For example, to include `GetDevEndpoints` in the policy, the resource must be scoped to the wildcard (*). The singular operations (`GetDevEndpoint`, `CreateDevEndpoint`, and `DeleteDevEndpoint`) are also scoped to all (*) resources in the example.

```
{  
    "Sid": "PluralAPIIncluded",  
    "Effect": "Allow",  
    "Action": [  
        "glue:GetDevEndpoints",  
        "glue:GetDevEndpoint",  
        "glue:CreateDevEndpoint",  
        "glue:UpdateDevEndpoint"  
    ],  
    "Resource": [  
        "*"  
    ]  
}
```

Access control for AWS Glue non-catalog BatchGet 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": "BatchGetAPIIncluded",  
    "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 [Examples of AWS Glue identity-based \(IAM\) access control policies with tags \(p. 86\)](#).

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 \(p. 81\)](#)
- [Examples of AWS Glue identity-based \(IAM\) access control policies with tags \(p. 86\)](#)
- [AWS Glue resource-based access control policy examples \(p. 91\)](#)

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 \(p. 76\)](#).

```
{  
    "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": [  
        {  
            "Action": "glue:CreateTable",  
            "Resource": "arn:aws:glue:us-west-2:123456789012:database/db1",  
            "Effect": "Allow",  
            "Sid": "CreateTableOnDb1"  
        }  
    ]  
}
```

```
{
    "Effect": "Allow",
    "Action": [
        "glue>CreateTable"
    ],
    "Resource": [
        "arn:aws:glue:us-west-2:123456789012:table/db1/tbl1",
        "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: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/*/*/dev-*",
                "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/dev-/*/*",
                "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.

Examples of AWS Glue identity-based (IAM) access control policies with tags

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"
                }
            ],
            "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` (`aws: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": {
                    "aws: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": {
                    "aws: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": {
                    "aws: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"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws: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:/*"
            ]
        }
    ]
}

```

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"
    }
}

```

}

2. 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:/*"
            ]
        }
    ]
}'
```

3. 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 table of [Actions, resources, and condition keys for AWS Glue](#) in the *Service Authorization Reference* as a reference when you're setting up [Identity and access management in AWS Glue \(p. 56\)](#) and writing a permissions policy to attach to an IAM identity (identity-based policy) or to a resource (resource policy). The table 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 `Action` element of your IAM policy statement, and you specify the resource value in the `Resource` element of your policy statement.

Actions on some AWS Glue resources require that ancestor and child resource ARNs are also included in the `Resource` element of your policy statement. For more information, see [Data Catalog ARNs \(p. 76\)](#).

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*.

However, the `aws:referer` and `aws:UserAgent` condition keys are currently not supported by the AWS Glue Data Catalog APIs.

Note

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

Related topics

- [Identity and access management \(p. 56\)](#)

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.

See Also

- [Automating AWS Glue with CloudWatch Events \(p. 314\)](#)
- [Cross-account CloudTrail logging \(p. 75\)](#)

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

- [AWS Glue and interface VPC endpoints \(AWS PrivateLink\)](#) (p. 97)
- [Shared Amazon VPCs](#) (p. 99)

AWS Glue and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS Glue by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#), a technology that enables you to privately access AWS Glue APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS Glue APIs. Traffic between your VPC and AWS Glue does not leave the Amazon network.

Each interface endpoint is represented by one or more [Elastic Network Interfaces](#) in your subnets.

For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the [Amazon VPC User Guide](#).

Considerations for AWS Glue VPC endpoints

Before you set up an interface VPC endpoint for AWS Glue, ensure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

AWS Glue supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for AWS Glue

You can create a VPC endpoint for the AWS Glue service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

Create a VPC endpoint for AWS Glue using the following service name:

- com.amazonaws.*region*.glue

If you enable private DNS for the endpoint, you can make API requests to AWS Glue using its default DNS name for the Region, for example, glue.us-east-1.amazonaws.com.

For more information, see [Accessing a service through an interface endpoint](#) in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for AWS Glue

You can attach an endpoint policy to your VPC endpoint that controls access to AWS Glue. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for AWS Glue to allow job creation and update

The following is an example of an endpoint policy for AWS Glue. When attached to an endpoint, this policy grants access to the listed AWS Glue actions for all principals on all resources.

```
{  
    "Statement": [  
        {  
            "Principal": "*",  
            "Effect": "Allow",  
            "Action": [  
                "glue>CreateJob",  
                "glue>UpdateJob",  
                "iam:PassRole"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Example: VPC endpoint policy to allow read-only Data Catalog access

The following is an example of an endpoint policy for AWS Glue. When attached to an endpoint, this policy grants access to the listed AWS Glue actions for all principals on all resources.

```
{  
    "Statement": [  
        {  
            "Principal": "*",  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetDatabase",  
                "glue:GetDatabases",  
                "glue:GetTable",  
                "glue:GetTables",  
                "glue:GetTableVersion",  
                "glue:GetTableVersions",  
                "glue:GetPartition",  
                "glue:GetPartitions",  
                "glue:BatchGetPartition",  
                "glue:SearchTables"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Shared Amazon VPCs

AWS Glue supports shared virtual private clouds (VPCs) in Amazon Virtual Private Cloud. Amazon VPC sharing allows multiple AWS accounts to create their application resources, such as Amazon EC2 instances and Amazon Relational Database Service (Amazon 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.

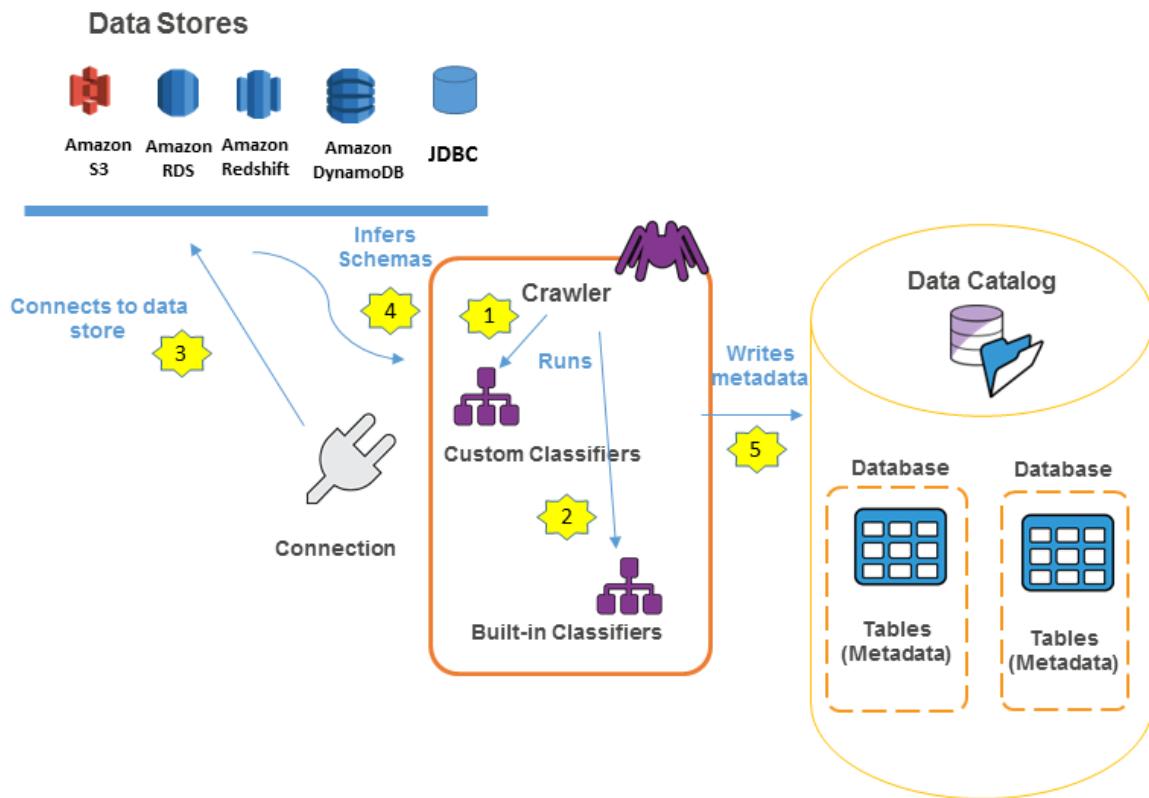
For more information, see these topics:

- [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 or data lake, 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 \(p. 102\)](#).

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.

2. 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.
3. The crawler connects to the data store. Some data stores require connection properties for crawler access.
4. The inferred schema is created for your data.
5. 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 \(p. 101\)](#)
- [Defining Tables in the AWS Glue Data Catalog \(p. 102\)](#)
- [Defining Connections in the AWS Glue Data Catalog \(p. 110\)](#)
- [Defining Crawlers \(p. 125\)](#)
- [Adding Classifiers to a Crawler \(p. 157\)](#)
- [Working with Data Catalog Settings on the AWS Glue Console \(p. 173\)](#)
- [Creating Tables, Updating Schema, and Adding New Partitions in the Data Catalog from AWS Glue ETL Jobs \(p. 174\)](#)

Defining a Database in Your Data Catalog

Databases are used to organize metadata tables in the AWS Glue. When you define a table in the AWS Glue Data Catalog, you add it to a database. A table can be in only one database.

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 from the AWS Glue Data Catalog, 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 \(p. 102\)](#).

Database Resource Links

The Data Catalog can also contain *resource links* to databases. A database resource link is a link to a local or shared database. Currently, you can create resource links only in AWS Lake Formation. After you create a resource link to a database, you can use the resource link name wherever you would use the database name. Along with databases that you own or that are shared with you, database resource links are returned by `glue:GetDatabases()` and appear as entries on the **Databases** page of the AWS Glue console.

The Data Catalog can also contain table resource links.

For more information about resource links, see [Creating Resource Links](#) in the *AWS Lake Formation Developer Guide*.

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 \(p. 125\)](#).

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 \(p. 125\)](#).
- 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 \(p. 104\)](#).
- Use the `CreateTable` operation in the [AWS Glue API \(p. 701\)](#) to create a table in the AWS Glue Data Catalog. For more information, see [CreateTable Action \(Python: `create_table`\) \(p. 738\)](#).
- Use AWS CloudFormation templates. For more information, see [Creating AWS Glue resources using AWS CloudFormation templates \(p. 178\)](#).
- 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 \(p. 104\)](#).

Topics

- [Table Partitions \(p. 103\)](#)

- [Table Resource Links \(p. 103\)](#)
- [Updating Manually Created Data Catalog Tables Using Crawlers \(p. 104\)](#)
- [Working with Tables on the AWS Glue Console \(p. 104\)](#)
- [Working with Partition Indexes \(p. 107\)](#)

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.

You can create partition indexes on a table to fetch a subset of the partitions instead of loading all the partitions in the table. For information about working with partition indexes, see [Working with Partition Indexes \(p. 107\)](#).

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
```

Table Resource Links

The Data Catalog can also contain *resource links* to tables. A table resource link is a link to a local or shared table. Currently, you can create resource links only in AWS Lake Formation. After you create a resource link to a table, you can use the resource link name wherever you would use the table name. Along with tables that you own or that are shared with you, table resource links are returned by `glue:GetTables()` and appear as entries on the **Tables** page of the AWS Glue console.

The Data Catalog can also contain database resource links.

For more information about resource links, see [Creating Resource Links](#) in the *AWS Lake Formation Developer Guide*.

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 \(p. 133\)](#).

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 \(p. 125\)](#).

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 \(p. 732\)](#).

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.

Working with Partition Indexes

Over time, hundreds of thousands of partitions get added to a table. The **GetPartitions API** is used to fetch the partitions in the table. The API returns partitions which match the expression provided in the request.

Lets take a *sales_data* table as an example which is partitioned by the keys *Country*, *Category*, *Year*, and *Month*. If you want to obtain sales data for all the items sold for the *Books* category in the year *2020*, you have to make a **GetPartitions** request with the expression "Category = Books and year = 2020" to the Data Catalog.

If no partition indexes are present on the table, AWS Glue loads all the partitions of the table, and then filters the loaded partitions using the query expression provided by the user in the **GetPartitions** request. The query takes more time to run as the number of partitions increase on a table with no indexes. With an index, the **GetPartitions** query will try to fetch a subset of the partitions instead of loading all the partitions in the table.

Topics

- [About Partition Indexes \(p. 107\)](#)
- [Creating a Table with Partition Indexes \(p. 108\)](#)
- [Adding a Partition Index to an Existing Table \(p. 108\)](#)
- [Describing Partition Indexes on a Table \(p. 108\)](#)
- [Limitations on Using Partition Indexes \(p. 109\)](#)
- [Using Indexes for an Optimized GetPartitions Call \(p. 109\)](#)
- [Integration with engines \(p. 110\)](#)

About Partition Indexes

When you create a partition index, you specify a list of partition keys that already exist on a given table. Partition index is sub list of partition keys defined in the table. A partition index can be created on any permutation of partition keys defined on the table. For the above *sales_data* table, the possible indexes are (country, category, year, month), (country, category, year), (country, category), (country), (category, country, year, month), and so on.

The Data Catalog will concatenate the partition values in the order provided at the time of index creation. The index is built consistently as partitions are added to the table. Indexes can be created for String (string, char, and varchar) and Numeric (int, bigint, long, tinyint, and smallint) column types. For example, for a table with the partition keys *country* (String), *item* (String), *creationDate* (date), an index cannot be created on the partition key *creationDate*.

Indexes on Numeric and String data types support =, >, >=, <, <= and between operators. The indexing solution currently only supports the AND logical operator. Sub-expressions with the operators "LIKE", "IN",

"OR", and "NOT" are ignored in the expression for filtering using an index. Filtering for the ignored sub-expression is done on the partitions fetched after applying index filtering.

For each partition added to a table, there is a corresponding index item created. For a table with 'n' partitions, 1 partition index will result in 'n' partition index items. 'm' partition index on same table will result into ' $m \times n$ ' partition index items. Each partition index item will be charged according to the current AWS Glue pricing policy for data catalog storage. For details on storage object pricing, see [AWS Glue pricing](#).

Creating a Table with Partition Indexes

You can create a partition index during table creation. The `CreateTable` request takes a list of [PartitionIndex objects](#) as an input. A maximum of 3 partition indexes can be created on a given table. Each partition index requires a name and a list of `partitionKeys` defined for the table. Created indexes on a table can be fetched using the [GetPartitionIndexes API](#)

Adding a Partition Index to an Existing Table

To add a partition index to an existing table, use the `CreatePartitionIndex` operation. You can create one `PartitionIndex` per `CreatePartitionIndex` operation. Adding an index does not affect the availability of a table, as the table continues to be available while indexes are being created.

The index status for an added partition is set to `CREATING` and the creation of the index data is started. If the process for creating the indexes is successful, the `indexStatus` is updated to `ACTIVE` and for an unsuccessful process, the index status is updated to `FAILED`. Index creation can fail for multiple reasons, and you can use the `GetPartitionIndexes` operation to retrieve the failure details. The possible failures are:

- `ENCRYPTED_PARTITION_ERROR` — Index creation on a table with encrypted partitions is not supported.
- `INVALID_PARTITION_TYPE_DATA_ERROR` — Observed when the `partitionKey` value is not a valid value for the corresponding `partitionKey` data type. For example: a `partitionKey` with the 'int' datatype has a value 'foo'.
- `MISSING_PARTITION_VALUE_ERROR` — Observed when the `partitionValue` for an `indexedKey` is not present. This can happen when a table is not partitioned consistently.
- `UNSUPPORTED_PARTITION_CHARACTER_ERROR` — Observed when the value for an indexed partition key contains the characters '\u0000, '\u0001 or '\u0002
- `INTERNAL_ERROR` — An internal error occurred while indexes were being created.

Describing Partition Indexes on a Table

To fetch the partition indexes created on a table, use the `GetPartitionIndexes` operation. The response returns all the indexes on the table, along with the current status of each index (the `IndexStatus`).

The `IndexStatus` for a partition index will be one of the following:

- `CREATING` — The index is currently being created, and is not yet available for use.
- `ACTIVE` — The index is ready for use. Requests can use the index to perform an optimized query.
- `DELETING` — The index is currently being deleted, and can no longer be used. An index in the active state can be deleted using the `DeletePartitionIndex` request, which moves the status from `ACTIVE` to `DELETING`.
- `FAILED` — The index creation on an existing table failed. Each table stores the last 10 failed indexes.

The possible state transitions for indexes created on an existing table are:

- CREATING → ACTIVE → DELETING
- CREATING → FAILED

Limitations on Using Partition Indexes

Once you have created a partition index, note these changes to table and partition functionality:

New Partition Creation (after Index Addition)

After a partition index is created on a table, all new partitions added to the table will be validated for the data type checks for indexed keys. The partition value of the indexed keys will be validated for data type format. If the data type check fails, the create partition operation will fail. For the *sales_data* table, if an index is created for keys (category, year) where the category is of type *string* and year of type *int*, the creation of the new partition with a value of YEAR as "foo" will fail.

After indexes are enabled, the addition of partitions with indexed key values having the characters U+0000, U+00001, and U+0002 will start to fail.

Table Updates

Once a partition index is created on a table, you cannot modify the partition key names for existing partition keys, and you cannot change the type, or order, of keys which are registered with the index.

Using Indexes for an Optimized GetPartitions Call

When you call *GetPartitions* on a table with an index, you can include an expression, and if applicable the Data Catalog will use an index if possible. The first key of the index should be passed in the expression for the indexes to be used in filtering. Index optimization in filtering is applied as a best effort. The Data Catalog tries to use index optimization as much as possible, but in case of a missing index, or unsupported operator, it falls back to the existing implementation of loading all partitions.

For the *sales_data* table above, lets add the index [Country, Category, Year]. If "Country" is not passed in the expression, the registered index will not be able to filter partitions using indexes. You can add up to 3 indexes to support various query patterns.

Lets take some example expressions and see how indexes work on them:

Expressions	How Index Will Be Used
Country = 'US'	Index will be used to filter partitions.
Country = 'US' and Category = 'Shoes'	Index will be used to filter partitions.
Category = 'Shoes'	Indexes will not be used as "country" is not provided in the expression. All partitions will be loaded to return a response.
Country = 'US' and Category = 'Shoes' and Year > '2018'	Index will be used to filter partitions.
Country = 'US' and Category = 'Shoes' and Year > '2018' and month = 2	Index will be used to fetch all partitions with country = "US" and category = "shoes" and year > 2018. Then, filtering on the month expression will be performed.

Expressions	How Index Will Be Used
Country = 'US' AND Category = 'Shoes' OR Year > '2018'	Indexes will not be used as an OR operator is present in the expression.
Country = 'US' AND Category = 'Shoes' AND (Year = 2017 OR Year = '2018')	Index will be used to fetch all partitions with country = "US" and category = "shoes", and then filtering on the year expression will be performed.
Country in ('US', 'UK') AND Category = 'Shoes'	Indexes will not be used for filtering as the IN operator is not supported currently.
Country = 'US' AND Category in ('Shoes', 'Books')	Index will be used to fetch all partitions with country = "US", and then filtering on the Category expression will be performed.

Integration with engines

Redshift Spectrum, Amazon EMR and AWS Glue ETL Spark DataFrames are able to utilize indexes for fetching partitions after indexes are in an ACTIVE state in AWS Glue. [Athena](#) and [AWS Glue ETL Dynamic frames](#) require you to follow extra steps to utilize indexes for query improvement.

Defining Connections in the AWS Glue Data Catalog

AWS Glue crawlers, jobs, and development endpoints use AWS Glue *connections* to access certain types of data stores.

Topics

- [AWS Glue Connections \(p. 110\)](#)
- [AWS Glue Connection Properties \(p. 111\)](#)
- [Adding an AWS Glue Connection \(p. 115\)](#)
- [Testing an AWS Glue Connection \(p. 115\)](#)
- [Configuring AWS calls to go through your VPC \(p. 116\)](#)
- [Connecting to a JDBC Data Store in a VPC \(p. 116\)](#)
- [Using a MongoDB Connection \(p. 118\)](#)
- [Crawling an Amazon S3 Data Store using a VPC Endpoint \(p. 119\)](#)

AWS Glue Connections

An AWS Glue connection is a Data Catalog object that stores connection information for a particular data store. Connections store login credentials, URI strings, virtual private cloud (VPC) information, and more. Creating connections in the Data Catalog saves the effort of having to specify all connection details every time you create a crawler or job. You can use connections for both sources and targets.

The following connection types are available:

- JDBC
- Amazon Redshift

- Amazon Relational Database Service (Amazon RDS)
- Amazon DocumentDB
- DynamoDB
- Kafka
- Amazon Kinesis
- MongoDB
- Network (designates a connection to a data source within an Amazon Virtual Private Cloud environment (Amazon VPC))
- Amazon S3

With AWS Glue Studio, you can also create connections for custom connectors or connectors you purchase from AWS Marketplace. For more information, see [Using connectors and connections with AWS Glue Studio](#)

When you create a crawler or extract, transform, and load (ETL) job for any of these data sources, you specify the connection to use. You can also optionally specify a connection when creating a development endpoint or writing data to a target.

Typically, a connection is not required for Amazon Simple Storage Service (Amazon S3) sources or targets that are on the public Internet. However, to access Amazon S3 from within your virtual private cloud (VPC), an Amazon S3 VPC endpoint of type Gateway is required. For more information, see [Amazon VPC Endpoints for Amazon S3 \(p. 32\)](#).

Additionally, if you want to access Amazon S3 data sources located in your virtual private cloud (VPC), you must create a Network type connection.

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 AWS Direct Connect because they provide private IP access to on-premises databases.

For information about how to connect to on-premises databases, see [How to access and analyze on-premises data stores using AWS Glue](#) at the AWS Big Data Blog website.

AWS Glue Connection Properties

When you define a connection on the AWS Glue console, you must provide values for the following properties:

Connection name

Type a unique name for your connection.

Connection type

Choose **JDBC** or one of the specific connection types.

For details about the JDBC connection type, see the section called [“AWS Glue JDBC Connection Properties” \(p. 112\)](#)

Choose **Network** to connect to a data source within an Amazon Virtual Private Cloud environment (Amazon VPC)).

Depending on the type that you choose, the AWS Glue console displays other required fields. For example, if you choose **Amazon RDS**, you must then choose the database engine.

Require SSL connection

When you select this option, AWS Glue must verify that the connection to the data store is connected over a trusted Secure Sockets Layer (SSL).

For more information, including additional options that are available when you select this option, see [the section called "AWS Glue Connection SSL Properties" \(p. 114\)](#).

Select MSK cluster (Amazon managed streaming for Apache Kafka (MSK) only)

Specifies an MSK cluster from another AWS account.

Kafka bootstrap server URLs (Kafka only)

Specifies a comma-separated list of bootstrap server URLs. Include the port number. For example: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-2.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-3.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094

AWS Glue JDBC Connection Properties

AWS Glue can connect to the following data stores through a JDBC connection:

- Amazon Redshift
- Amazon Aurora
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- Amazon RDS for MariaDB

Important

Currently, an ETL job can use JDBC connections within only one subnet. If you have multiple data stores in a job, they must be on the same subnet.

The following are additional properties for the JDBC connection type.

JDBC URL

Type the URL for your JDBC data store. For most database engines, this field is in the following format. In this format, replace `protocol`, `host`, `port`, and `db_name` with your own information.

`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. In these patterns, replace `host`, `port`, `service_name`, and `SID` with your own information.

- `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. In these patterns, replace `server_name`, `port`, and `db_name` with your own information.

- `jdbc:sqlserver://server_name:port;database=db_name`
- `jdbc:sqlserver://server_name:port;databaseName=db_name`
- To connect to an Amazon Aurora PostgreSQL instance of the employee database, specify the endpoint for the database instance, the port, and the database name:

```
jdbc:postgresql://employee_instance_1.xxxxxxxxxxxxxx.us-  
east-2.rds.amazonaws.com:5432/employee
```

- To connect to an Amazon RDS for MariaDB data store with an employee database, specify the endpoint for the database instance, the port, and the database name:

```
jdbc:mysql://xxx-cluster.cluster-xxx.aws-region.rds.amazonaws.com:3306/  
employee
```

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.

AWS Glue Connection SSL Properties

The following are details about the **Require SSL connection** property of AWS Glue connections.

If this option is not selected, AWS Glue ignores failures when it uses SSL to encrypt a connection to the data store. See the documentation for your data store 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. For JDBC connections, AWS Glue only connects over SSL with certificate and host name validation. SSL connection support is available for:

- Oracle Database
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL (Amazon RDS instances only)
- Amazon Aurora MySQL (Amazon RDS instances only)
- Amazon Aurora PostgreSQL (Amazon RDS instances only)
- Kafka, which includes Amazon Managed Streaming for Apache Kafka

Note

To enable an **Amazon RDS Oracle** data store to use **Require SSL connection**, you must 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](#) in the *Amazon RDS User Guide*. For more information about the Oracle SSL option, see [Oracle SSL](#) in the *Amazon RDS User Guide*.
4. On the AWS Glue console, create a connection to the Amazon RDS Oracle instance. In the connection definition, select **Require SSL connection**. When requested, enter the **Port** that you used in the Amazon RDS Oracle SSL option.

The following additional optional properties are available when **Require SSL connection** is selected for a connection:

Custom JDBC certificate in S3

If you have a certificate that you are currently using for SSL communication with your on-premises or cloud databases, you can use that certificate for SSL connections to AWS Glue data sources or targets. Enter an Amazon Simple Storage Service (Amazon S3) location that contains 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.

Custom JDBC certificate string

Enter certificate information specific to your JDBC database. This string is used for domain matching or distinguished name (DN) matching. For Oracle Database, this string maps to the

`SSL_SERVER_CERT_DN` parameter in the security section of the `tnsnames.ora` file. For Microsoft SQL Server, this string 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
```

Kafka private CA certificate location

If you have a certificate that you are currently using for SSL communication with your Kafka data store, you can use that certificate with your AWS Glue connection. This option is required for Kafka data stores, and optional for Amazon Managed Streaming for Apache Kafka data stores. Enter an Amazon Simple Storage Service (Amazon S3) location that contains a custom root certificate. AWS Glue uses this certificate to establish an SSL connection to the Kafka data store. AWS Glue handles only X.509 certificates. The certificate must be DER-encoded and supplied in base64 encoding PEM format.

Skip certificate validation

Select the **Skip certificate validation** 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.

Kafka client keystore location

The Amazon S3 location of the client keystore file for Kafka client side authentication. Path must be in the form `s3://bucket/prefix/filename.jks`. It must end with the file name and `.jks` extension.

Kafka client keystore password (optional)

The password to access the provided keystore.

Kafka client key password (optional)

A keystore can consist of multiple keys, so this is the password to access the client key to be used with the Kafka server side key.

Adding an AWS Glue Connection

You can use the AWS Glue console to add, edit, delete, and test connections. For information about AWS Glue connections, see the section called “AWS Glue Connections” (p. 110).

To add an AWS Glue connection

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 **Data catalog**, choose **Connections**.
3. Choose **Add connection** and then complete the wizard, entering connection properties as described in the section called “AWS Glue Connection Properties” (p. 111).

Testing an AWS Glue Connection

As a best practice, before you use an AWS Glue connection in an extract, transform, and load (ETL) job, use the AWS Glue console to test the connection. AWS Glue uses the parameters in your connection

to confirm that it can access your data store, and reports any errors. For information about AWS Glue connections, see [the section called “AWS Glue Connections” \(p. 110\)](#).

To test an AWS Glue connection

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 **Data catalog**, choose **Connections**.
3. Select the check box next to the desired connection, and then choose **Test connection**.
4. In the **Test connection** dialog box, select a role or choose **Create IAM role** to go to the AWS Identity and Access Management (IAM) console to create a new role. The role must have permissions on the data store.
5. Choose **Test connection**.

The test begins and can take several minutes to complete. If the test fails, choose **logs** in the banner at the top of the screen to view the Amazon CloudWatch Logs.

You must have the required IAM permissions to view the logs. For more information, see [AWS Managed \(Predefined\) Policies for CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

Configuring AWS calls to go through your VPC

The special job parameter `disable-proxy` allows you to route your calls to Amazon S3 for downloading scripts and libraries through your VPC. AWS Glue uses a local proxy to send traffic through the AWS Glue VPC to download scripts and libraries and to send requests to CloudWatch for publishing logs and metrics. This proxy allows the job to function normally even if your VPC doesn't configure a proper route to Amazon S3 and CloudWatch. AWS Glue now offers a parameter for you to turn off this behavior for all Amazon S3 APIs used in downloading scripts and libraries. For more information, see [Special Parameters Used by AWS Glue](#).

Note

This feature is supported for AWS Glue jobs with AWS Glue version 2.0 and above. When using this feature, you need to ensure that your VPC has configured a route to Amazon S3 through a NAT or service VPC endpoint.

Example usage

Create an AWS Glue job with `disable-proxy`:

```
aws glue create-job \
--name no-proxy-job \
--role GlueDefaultRole \
--command "Name=glueetl,ScriptLocation=s3://my-bucket/glue-script.py" \
--connections Connections="traffic-monitored-connection" \
--default-arguments '{"--disable-proxy" : "true"}'
```

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, AWS Glue can't access resources inside a VPC. 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 Relational Database Service (Amazon RDS), see [Working with an Amazon RDS DB Instance in a VPC](#).

Using a MongoDB Connection

After creating a connection for MongoDB, you can use this connection in your ETL job. You create a table in the AWS Glue Data Catalog and specify the MongoDB connection for the `connection` attribute of the table. The connection `url`, `username` and `password` are stored in the MongoDB connection. Other options can be specified in the job script as additional options. The other options can include:

- "database": (Required) The MongoDB database to read from.
- "collection": (Required) The MongoDB collection to read from.
- "ssl": (Optional) If `true`, then AWS Glue initiates an SSL connection. The default value is `false`.
- "ssl.domain_match": (Optional) If `true` and `ssl` is `true`, then AWS Glue performs a domain match check. The default value is `true`.
- "batchSize": (Optional): The number of documents to return per batch, used within the cursor of internal batches.
- "partitioner": (Optional): The class name of the partitioner for reading input data from MongoDB. The connector provides the following partitioners:
 - `MongoDefaultPartitioner` (default)
 - `MongoSamplePartitioner` (Requires MongoDB 3.2 or later)
 - `MongoShardedPartitioner`
 - `MongoSplitVectorPartitioner`
 - `MongoPaginateByCountPartitioner`
 - `MongoPaginateBySizePartitioner`
- "partitionerOptions": (Optional): Options for the designated partitioner. The following options are supported for each partitioner:
 - `MongoSamplePartitioner`-`partitionKey`, `partitionSizeMB`, and `samplesPerPartition`
 - `MongoShardedPartitioner`-`shardkey`
 - `MongoSplitVectorPartitioner`-`partitionKey` and `partitionSizeMB`
 - `MongoPaginateByCountPartitioner`-`partitionKey` and `numberOfPartitions`
 - `MongoPaginateBySizePartitioner`-`partitionKey` and `partitionSizeMB`

For more information about these options, see <https://docs.mongodb.com/spark-connector/master/configuration/#partitioner-conf>.

The following examples demonstrate how to get a `DynamicFrame` from the catalog source.

Python

```
glue_context.create_dynamic_frame_from_catalog(  
    database = nameSpace,  
    table_name = tableName,  
    additional_options = {"database":"database_name",  
                         "collection":"collection_name"})
```

Scala

```
val resultFrame: DynamicFrame = glueContext.getCatalogSource(  
    database = nameSpace,  
    tableName = tableName,  
    additionalOptions = JsonOptions(Map("database" -> DATABASE_NAME,  
                                       "collection" -> COLLECTION_NAME))  
).getDynamicFrame()
```

Crawling an Amazon S3 Data Store using a VPC Endpoint

For security, auditing, or control purposes you may want your Amazon S3 data store or Amazon S3 backed Data Catalog tables to only be accessed through an Amazon Virtual Private Cloud environment (Amazon VPC). This topic describes how to create and test a connection to the Amazon S3 data store or Amazon S3 backed Data Catalog tables in a VPC endpoint using the Network connection type.

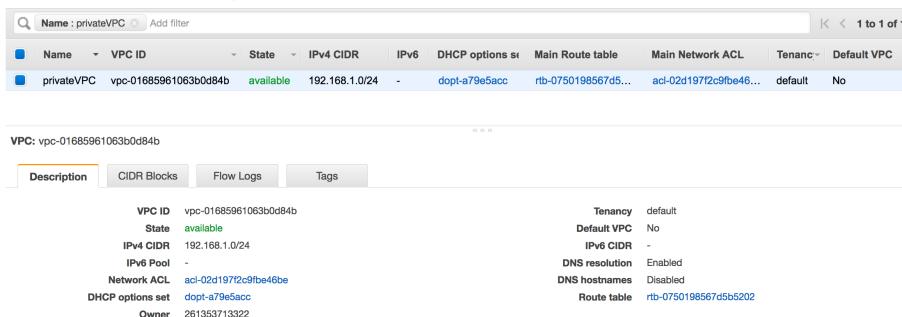
Perform the following tasks to run a crawler on the data store:

- the section called “Prerequisites” (p. 119)
- the section called “Creating the Connection to Amazon S3” (p. 120)
- the section called “Testing the Connection to Amazon S3” (p. 121)
- the section called “Creating a Crawler for an Amazon S3 data store” (p. 122)
- the section called “Running a Crawler” (p. 124)

Prerequisites

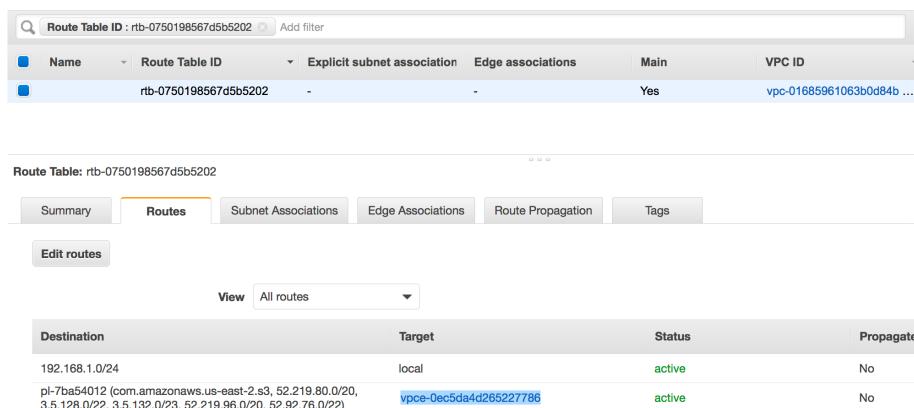
Check that you have met these prerequisites for setting up your Amazon S3 data store or Amazon S3 backed Data Catalog tables to be accessed through an Amazon Virtual Private Cloud environment (Amazon VPC)

- A configured VPC. For example: vpc-01685961063b0d84b. For more information, see [Getting started with Amazon VPC](#) in the *Amazon VPC User Guide*.
- An Amazon S3 endpoint attached to the VPC. For example: vpc-01685961063b0d84b. For more information, see [Endpoints for Amazon S3](#) in the *Amazon VPC User Guide*.



The screenshot shows two parts of the AWS VPC console. The top part is a table listing VPCs with columns: Name, VPC ID, State, IPv4 CIDR, IPv6, DHCP options set, Main Route table, Main Network ACL, Tenancy, and Default VPC. One row is selected for 'privateVPC'. The bottom part is a detailed view of this VPC, showing tabs for Description, CIDR Blocks, Flow Logs, and Tags. The Description tab displays various configuration details such as VPC ID, State, IPv4 CIDR, IPv6 Pool, Network ACL, DHCP options set, and Owner. The right side of the screen shows corresponding settings for Tenancy, Default VPC, IPv6 CIDR, DNS resolution, DNS hostnames, and Route table.

- A route entry pointing to the VPC endpoint. For example vpce-0ec5da4d265227786 in the route table used by the VPC endpoint(vpce-0ec5da4d265227786).



The screenshot shows the AWS Route Table configuration page. At the top, there is a search bar with the placeholder "Route Table ID : rtb-0750198567d5b5202" and a "Add filter" button. Below the search bar is a table with columns: Name, Route Table ID, Explicit subnet association, Edge associations, Main, and VPC ID. A single row is selected, showing "rtb-0750198567d5b5202" in the Route Table ID column, and "vpc-01685961063b0d84b" in the VPC ID column. Below the table, tabs for "Summary", "Routes" (which is selected), "Subnet Associations", "Edge Associations", "Route Propagation", and "Tags" are visible. Under the "Routes" tab, there is a "Edit routes" button and a dropdown menu set to "All routes". The main content area displays a table with columns: Destination, Target, Status, and Propagate. One route is listed: "192.168.1.0/24" with "local" as the target, "active" status, and "No" propagate setting. Another route is listed: "pl-7ba54012 (com.amazonaws.us-east-2.s3, 52.219.80.0/20, 3.5.128.0/22, 3.5.132.0/23, 52.219.96.0/20, 52.92.76.0/22)" with "vpce-0ec5da4d265227786" as the target, "active" status, and "No" propagate setting.

- A network ACL attached to the VPC allows the traffic.
- A security group attached to the VPC allows the traffic.

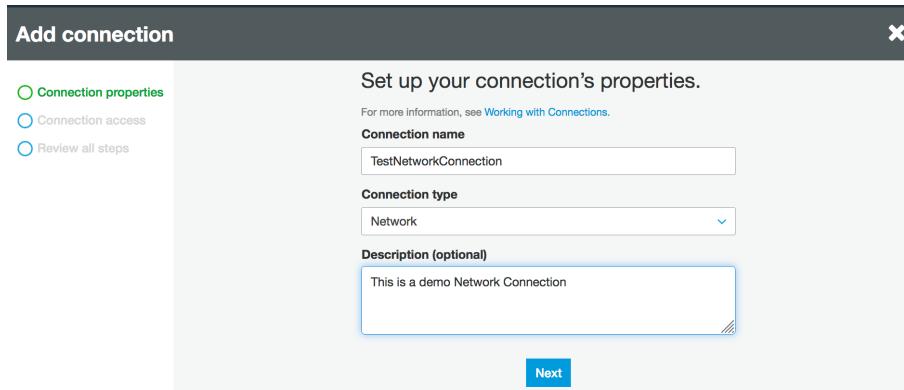
Creating the Connection to Amazon S3

Typically, you create resources inside Amazon Virtual Private Cloud (Amazon VPC) so that they cannot be accessed over the public internet. By default, AWS Glue can't access resources inside a VPC. 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. To create a Network connection you need to specify the following information:

- A VPC ID
- A subnet within the VPC
- A security group

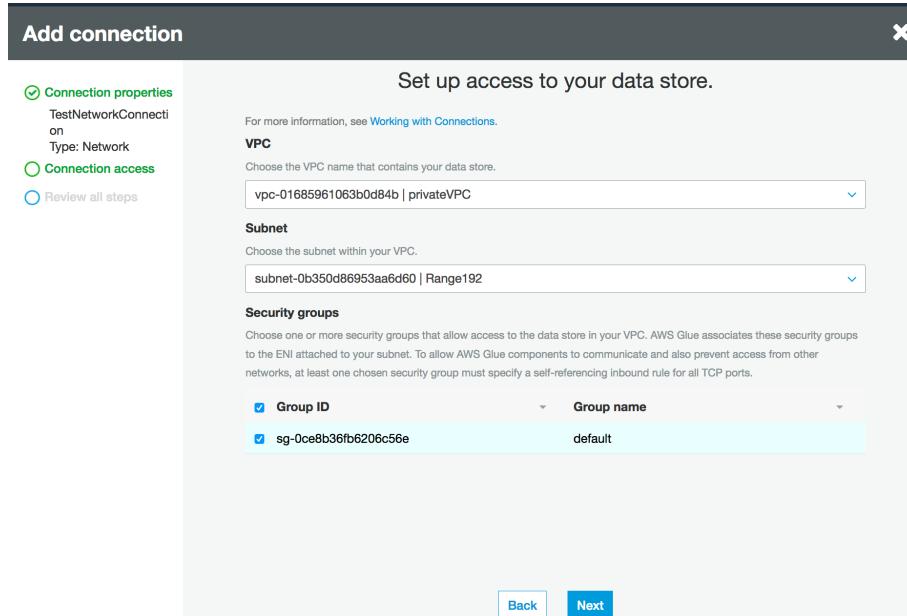
To set up a Network connection:

1. Choose **Add connection** in the navigation pane of the AWS Glue console.
2. Enter the connection name, choose **Network** as the connection type. Choose **Next**.

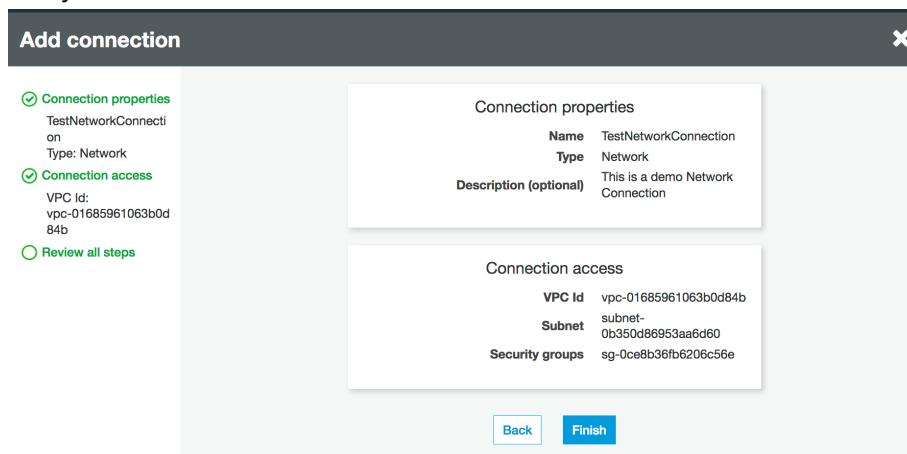


The screenshot shows the "Add connection" wizard. The title bar says "Add connection" and has a close button. On the left, there is a sidebar with three tabs: "Connection properties" (selected), "Connection access", and "Review all steps". The main area is titled "Set up your connection's properties." It contains a "Connection name" field with "TestNetworkConnection", a "Connection type" dropdown set to "Network", and a "Description (optional)" text area with "This is a demo Network Connection". At the bottom right is a blue "Next" button.

3. Configure the VPC, Subnet and Security groups information.
 - VPC: choose the VPC name that contains your data store.
 - Subnet: choose the subnet within your VPC.
 - Security groups: choose one or more security groups that allow access to the data store in your VPC.



4. Choose **Next**.
5. Verify the connection information and choose **Finish**.



Testing the Connection to Amazon S3

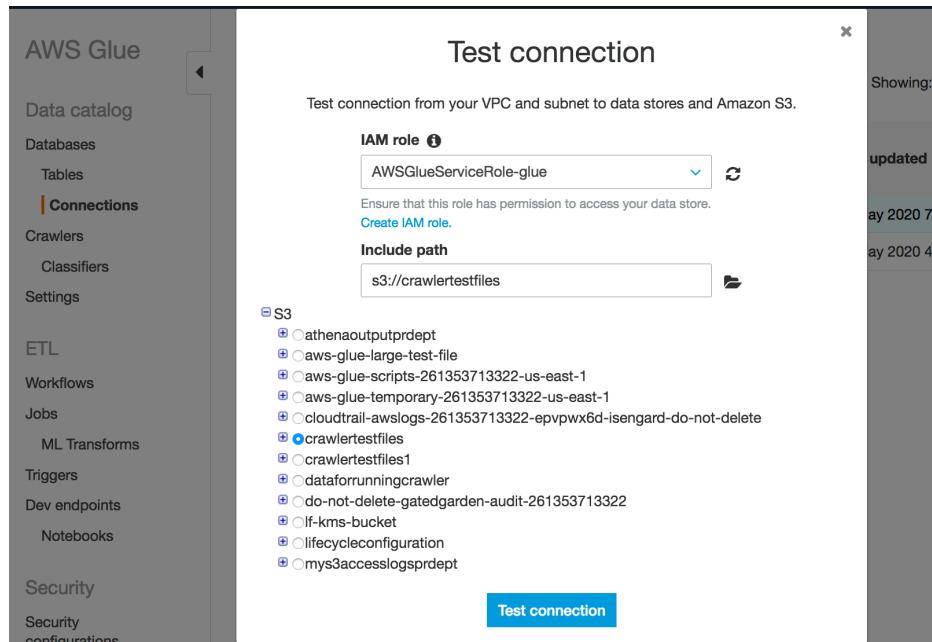
Once you have created your Network connection, you can test the connectivity to your Amazon S3 data store in a VPC endpoint.

The following errors may occur when testing a connection:

- **INTERNET CONNECTION ERROR:** indicates an Internet connection issue
- **INVALID BUCKET ERROR:** indicates a problem with the Amazon S3 bucket
- **S3 CONNECTION ERROR:** indicates a failure to connect to Amazon S3
- **INVALID CONNECTION TYPE:** indicates the Connection type does not have the expected value, NETWORK
- **INVALID CONNECTION TEST TYPE:** indicates a problem with the type of network connection test
- **INVALID TARGET:** indicates that the Amazon S3 bucket has not been specified properly

To test a Network connection:

1. Select the **Network** connection in the AWS Glue console.
2. Choose **Test connection**.
3. Choose the IAM role that you created in the previous step and specify an Amazon S3 Bucket.
4. Choose **Test connection** to start the test. It might take few moments to show the result.



If you receive an error, check the following:

- The correct privileges are provided to the role selected.
- The correct Amazon S3 bucket is provided.
- The security groups and Network ACL allow the required incoming and outgoing traffic.
- The VPC you specified is connected to an Amazon S3 VPC endpoint.

Once you have successfully tested the connection, you can create a crawler.

Creating a Crawler for an Amazon S3 data store

You can now create a crawler that specifies the Network connection you've created. For more details on creating a crawler, see the section called "Working with Crawlers on the Console" (p. 144).

1. Start by choosing **Crawlers** in the navigation pane on the AWS Glue console.
2. Choose **Add crawler**.
3. Specify the crawler name and choose **Next**.
4. When asked for the data source, choose **S3**, and specify the Amazon S3 bucket prefix and the connection you created earlier.

Add crawler

Crawler info
TestNetworkConnecti
on

Crawler source type
Data stores

Data store
S3:
S3:
s3://crawler...
IAM Role
Schedule
Output
Review all steps

Add a data store

Choose a data store: **S3**

Connection: **AddNetworkConnection**

Optional: Include a Network connection to use with this S3 target. Note that each crawler is limited to one Network connection so any future S3 targets will also use the same connection (or none, if left blank).

Add connection

Crawl data in:
 Specified path

Include path: **s3://crawler.../testfiles**

All folders and files contained in the include path are crawled. For example, type s3://MyBucket/MyFolder/ to crawl all objects in MyFolder within MyBucket.

Exclude patterns (optional)

Chosen data stores:
S3:

X

Back **Next**

5. If you need to, add another data store on the same network connection.
6. Choose IAM role. The IAM role must allow access to the AWS Glue service and the Amazon S3 bucket. For more information, see [the section called "Working with Crawlers on the Console" \(p. 144\)](#).

Add crawler

Crawler info
TestNetworkConnecti
on

Crawler source type
Data stores

Data store
S3: s3://crawler.../testfiles

IAM Role

Schedule

Output

Review all steps

Choose an IAM role

The IAM role allows the crawler to run and access your Amazon S3 data stores. [Learn more](#)

Update a policy in an IAM role
 Choose an existing IAM role
 Create an IAM role

IAM role: **AWSGlueServiceRole-glue**

This role must provide permissions similar to the AWS managed policy, **AWSGlueServiceRole**, plus access to your data stores.

- s3://crawler.../testfiles

You can also create an IAM role on the [IAM console](#).

Back **Next**

7. Define the schedule for the crawler.
8. Choose an existing database in the Data Catalog, or create a new database entry.

Add crawler

Crawler info
TestNetworkConnecti
on

Crawler source type
Data stores

Data store
S3: s3://crawler.../testfiles

IAM Role
arn:aws:iam::2613537
13322:role/service-
role/AWSGlueService
Role-glue

Schedule
Run on demand

Output

Review all steps

Configure the crawler's output

Database: **testnetworkconnectiondb**

Add database

Prefix added to tables (optional): **Type a prefix added to table names**

Grouping behavior for S3 data (optional)

Configuration options (optional)

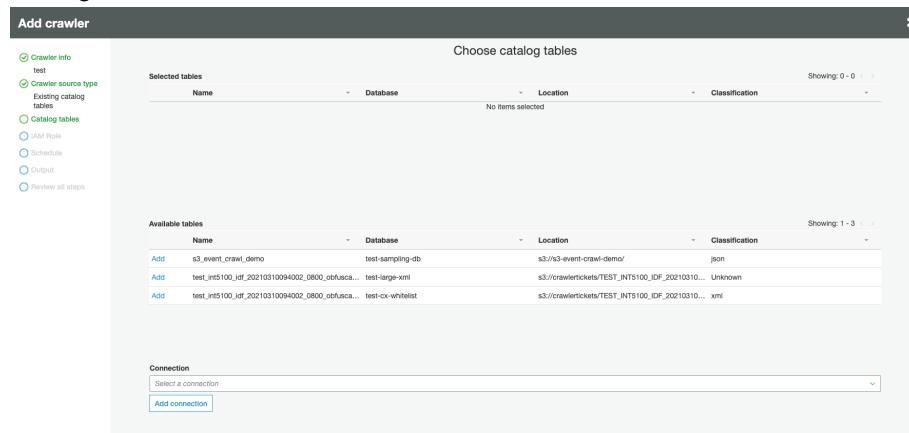
Back **Next**

9. Finish the remaining setup.

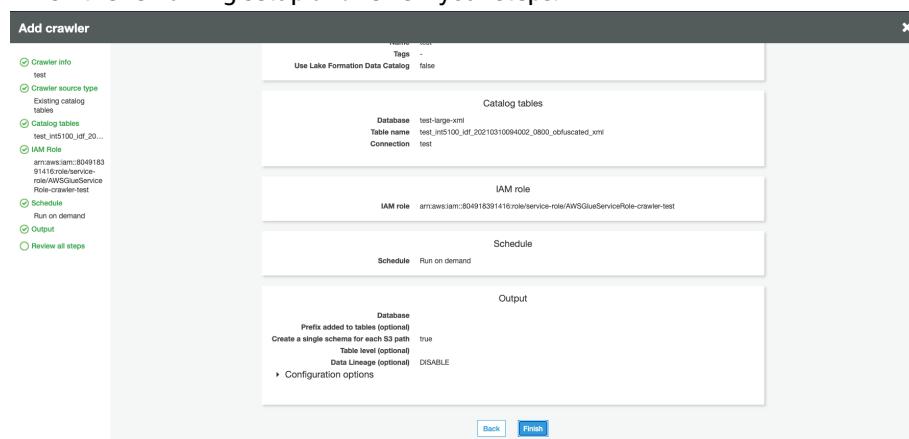
Creating a crawler for Amazon S3 backed Data Catalog tables

You can now create a crawler that specifies the Network connection you've created and a Catalog source type. For more details on creating a crawler, see [the section called "Working with Crawlers on the Console" \(p. 144\)](#).

1. Start by choosing **Crawlers** in the navigation pane on the AWS Glue console.
2. Choose **Add crawler**.
3. Specify the crawler name and choose **Next**.
4. When asked for the crawler source type, choose **Existing catalog tables**, and specify the existing catalog tables to crawl from the list of available tables.

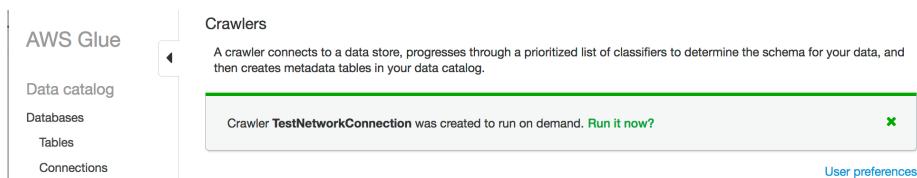


5. Choose IAM role. The IAM role must allow access to the AWS Glue service and the Amazon S3 bucket. For more information, see [the section called "Working with Crawlers on the Console" \(p. 144\)](#).
6. Define the schedule for the crawler.
7. Choose an existing database in the Data Catalog, or create a new database entry.
8. Finish the remaining setup and review your steps.



Running a Crawler

Run your crawler.



Troubleshooting

For troubleshooting related to Amazon S3 buckets using a VPC gateway, see [Why can't I connect to an S3 bucket using a gateway VPC endpoint?](#)

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 \(p. 144\)](#).

Topics

- [Which Data Stores Can I Crawl? \(p. 125\)](#)
- [How Crawlers Work \(p. 126\)](#)
- [Crawler Prerequisites \(p. 130\)](#)
- [Crawler Properties \(p. 131\)](#)
- [Setting Crawler Configuration Options \(p. 137\)](#)
- [Scheduling an AWS Glue Crawler \(p. 143\)](#)
- [Working with Crawlers on the AWS Glue Console \(p. 144\)](#)
- [Accelerating Crawls Using Amazon S3 Event Notifications \(p. 145\)](#)
- [Using Encryption with the Amazon S3 Event Crawler \(p. 152\)](#)

Which Data Stores Can I Crawl?

Crawlers can crawl the following file-based and table-based data stores.

Access type that crawler uses	Data stores
Native client	<ul style="list-style-type: none">• Amazon Simple Storage Service (Amazon S3)• Amazon DynamoDB• Delta Lake
JDBC	Amazon Redshift Within Amazon Relational Database Service (Amazon RDS) or external to Amazon RDS: <ul style="list-style-type: none">• Amazon Aurora• MariaDB

Access type that crawler uses	Data stores
	<ul style="list-style-type: none"> Microsoft SQL Server MySQL Oracle PostgreSQL
MongoDB client	<ul style="list-style-type: none"> MongoDB Amazon DocumentDB (with MongoDB compatibility)

Note

Currently AWS Glue does not support crawlers for data streams.

For JDBC, MongoDB, and Amazon DocumentDB (with MongoDB compatibility) data stores, you must specify an AWS Glue *connection* that the crawler can use to connect to the data store. For Amazon S3, you can optionally specify a connection of type Network. A connection is a Data Catalog object that stores connection information, such as credentials, URL, Amazon Virtual Private Cloud information, and more. For more information, see [Defining Connections in the AWS Glue Data Catalog \(p. 110\)](#).

The following are notes about the various data stores.

Amazon S3

You can choose to crawl a path in your account or in another account. 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 and partition information is added to the Data Catalog for that table.

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.*

Amazon DynamoDB

When defining a crawler using the AWS Glue console, you specify one DynamoDB table. If you're using the AWS Glue API, you can specify a list of tables. You can choose to crawl only a small sample of the data to reduce crawler run times.

Delta Lake

For each Delta Lake data store, the crawler scans the Delta table's transaction log to detect metadata. It populates the `_symlink_manifest` folder with the manifest files that are partitioned by the partition keys, based on configuration parameters that you choose.

MongoDB and Amazon DocumentDB (with MongoDB compatibility)

MongoDB versions 3.2 and later are supported. You can choose to crawl only a small sample of the data to reduce crawler run times.

Relational database

Authentication is with a database user name and password. Depending on the type of database engine, you can choose which objects are crawled, such as databases, schemas, and tables.

How Crawlers Work

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.

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 \(p. 158\)](#).

The metadata tables that a crawler creates are contained in a database when you define a crawler. If your crawler does not specify 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.

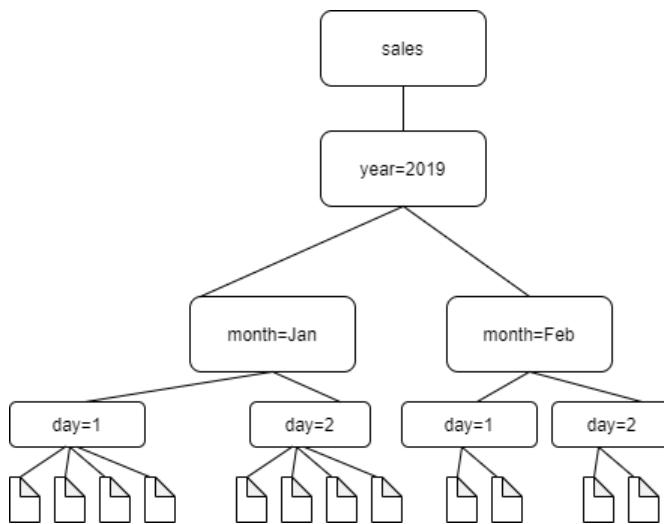
Topics

- [How Does a Crawler Determine When to Create Partitions? \(p. 127\)](#)
- [Incremental Crawls in AWS Glue \(p. 129\)](#)

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 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, consider the following Amazon S3 folder structure.



The paths to the four lowest level folders are the following:

```
s3://sales/year=2019/month=Jan/day=1
s3://sales/year=2019/month=Jan/day=2
s3://sales/year=2019/month=Feb/day=1
s3://sales/year=2019/month=Feb/day=2
```

Assume that the crawler target is set at **Sales**, and that all files in the **day=n** folders have the same format (for example, JSON, not encrypted), and have the same or very similar schemas. The crawler will create a single table with four partitions, with partition keys **year**, **month**, and **day**.

In the next example, consider 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 files under **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 key columns. The first partition key column contains **table1** and **table2**, and the second partition key column contains **partition1** through **partition3** for the **table1** partition and **partition4** and **partition5** for the **table2** partition. 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/`.

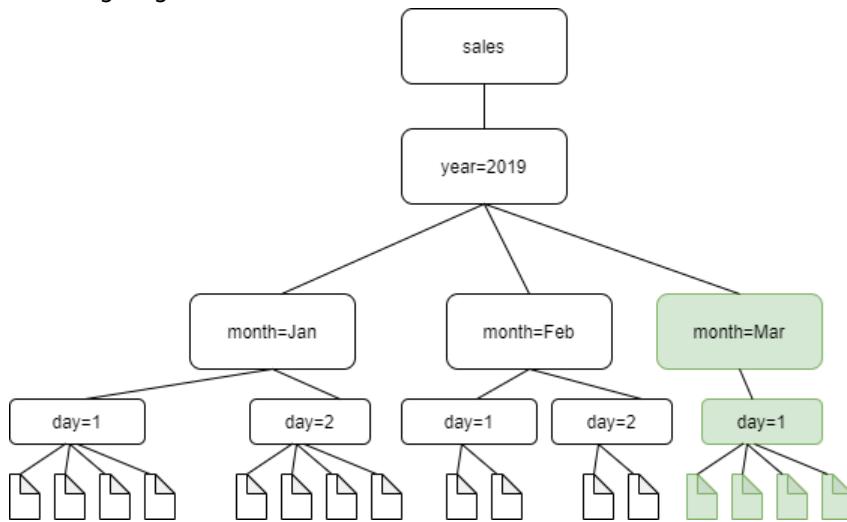
Note

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](#).

Incremental Crawls in AWS Glue

For an Amazon Simple Storage Service (Amazon S3) data source, incremental crawls only crawl folders that were added since the last crawler run. Without this option, the crawler crawls the entire dataset. Incremental crawls can save significant time and cost. To perform an incremental crawl, you can set the **Crawl new folders only** option in the AWS Glue console or set the `RecrawlPolicy` property in the `CreateCrawler` request in the API.

Incremental crawls are best suited to incremental datasets with a stable table schema. The typical use case is for scheduled crawlers, where during each crawl, new partitions are added. Continuing with the example in the section called “[How Does a Crawler Determine When to Create Partitions?](#)” (p. 127), the following diagram shows that files for the month of March have been added.



If you set the **Crawl new folders only** option, only the new folder, `month=Mar` is crawled.

Notes and Restrictions for Incremental Crawls

Keep in mind the following additional information about incremental crawls:

- The best practice for incremental crawls is to first run a complete crawl on the target dataset to enable the crawler to record the initial schema and partition structure.
- When this option is turned on, you can't change the Amazon S3 target data stores when editing the crawler.
- This option affects certain crawler configuration settings. When turned on, it forces the update behavior and delete behavior of the crawler to `LOG`. This means that:
 - If an incremental crawl discovers objects with schemas that are different enough from the schema recorded in the Data Catalog such that the crawler cannot create new partitions, the crawler ignores the objects and records the event in CloudWatch Logs.
 - If an incremental crawl discovers deleted objects, it ignores them and doesn't update the Data Catalog.

For more information, see [the section called “Setting Crawler Configuration Options” \(p. 137\)](#).

- If an incremental crawl discovers multiple new partitions or folders added, the majority of them have to match the schema recorded in the Data Catalog to enable the crawler to add them successfully. Otherwise, the crawler might fail to add the partitions because there are too many schema varieties.

Crawler Prerequisites

The crawler assumes the permissions of the AWS Identity and Access Management (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 \(IAM policies\) for access control \(p. 61\)](#).

Note

When crawling a Delta Lake data store, you must have Read/Write permissions to the Amazon S3 location.

For your crawler, you can create a role and attach the following policies:

- The `AWSGlueServiceRole` AWS managed policy, which grants the required permissions on the Data Catalog
- An inline policy that grants permissions on the data source.

A quicker approach is to let the AWS Glue console crawler wizard create a role for you. The role that it creates is specifically for the crawler, and includes the `AWSGlueServiceRole` AWS managed policy plus the required inline policy for the specified data source.

If you specify an existing role for a crawler, ensure that it includes the `AWSGlueServiceRole` policy or equivalent (or a scoped down version of this policy), plus the required inline policies. For example, for an Amazon S3 data store, the inline policy would at a minimum be the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket/object*"
            ]
        }
    ]
}
```

For an Amazon DynamoDB data store, the policy would at a minimum be the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Scan"
            ],
            "Resource": [
                "arn:aws:dynamodb:region:account-id:table/table-name*"
            ]
        }
    ]
}
```

In addition, if the crawler reads AWS Key Management Service (AWS KMS) encrypted Amazon S3 data, then the IAM role must have decrypt permission on the AWS KMS key. For more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 16\)](#).

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 \(p. 310\)](#)
- [Adding Classifiers to a Crawler \(p. 157\)](#)

Crawler source type

The crawler can access data stores directly as the source of the crawl, or it can use existing tables in the Data Catalog 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 \(p. 133\)](#).

Crawl only new folders for S3 data sources

If turned on, only Amazon S3 folders that were added since the last crawler run will be crawled. For more information, see [the section called “Incremental Crawls” \(p. 129\)](#).

Crawler sources: data stores or catalog tables

Choose from among the following:

- One or more data stores
 - A crawler can crawl multiple data stores of different types (Amazon S3, JDBC, and so on).
- 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.

You can configure only one data store at a time. After you have provided the connection information and include paths and exclude patterns, you then have the option of adding another data store.

For more information, see [Crawler Source Type \(p. 133\)](#).

Additional crawler source parameters

Each source type requires a different set of additional parameters. The following is an incomplete list:

Connection

Select or add an AWS Glue connection. For information about connections, see [the section called “Defining Connections in the AWS Glue Data Catalog” \(p. 110\)](#).

Enable data sampling (for Amazon DynamoDB, MongoDB, and Amazon DocumentDB data stores only)

Select whether to crawl a data sample only. If not selected the entire table is crawled. Scanning all the records can take a long time when the table is not a high throughput table.

Enable write manifest (for Delta Lake data stores only)

Select whether to detect table metadata or schema changes in the Delta Lake transaction log; it regenerates the manifest file. You should not choose this option if you configured an automatic manifest update with Delta Lake `SET_TBLPROPERTIES`.

Scanning rate (for DynamoDB data stores only)

Specify the percentage of the configured read capacity units to use by the AWS Glue crawler. Read capacity units is a term defined by DynamoDB, and is a numeric value that acts as rate limiter for the number of reads that can be performed on that table per second. Enter a value between 0.1 and 1.5. If not specified, defaults to 0.5% for provisioned tables and 1/4 of maximum configured capacity for on-demand tables. Note that only provisioned capacity mode should be used with AWS Glue crawlers.

Note

For DynamoDB data stores, set the provisioned capacity mode for processing reads and writes on your tables. The AWS Glue crawler should not be used with the on-demand capacity mode.

Sample size (optional) (for Amazon S3 data stores only)

Specify the number of files in each leaf folder to be crawled when crawling sample files in a dataset. When this feature is turned on, instead of crawling all the files in this dataset, the crawler randomly selects some files in each leaf folder to crawl.

The sampling crawler is best suited for customers who have previous knowledge about their data formats and know that schemas in their folders do not change. Turning on this feature will significantly reduce crawler runtime.

A valid value is an integer between 1 and 249. If not specified, all the files are crawled.

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 Delta Lake data store

Specify one or more Amazon S3 paths to Delta tables as `s3://bucket/prefix/object`.

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.

For a MongoDB or Amazon DocumentDB data store

Enter `database/collection`.

For more information, see [Include and Exclude Patterns \(p. 134\)](#).

Exclude patterns

These enable you to exclude certain files or tables from the crawl. For more information, see [Include and Exclude Patterns \(p. 134\)](#).

IAM role

The crawler assumes this role. It must have permissions similar to the AWS managed policy `AWSGlueServiceRole`. For Amazon S3 and DynamoDB sources, it must also have permissions to access the data store. If the crawler reads Amazon S3 data encrypted with AWS Key Management Service (AWS KMS), then the role must have decrypt permissions on the AWS KMS key.

For an Amazon S3 data store, additional permissions attached to the role would be similar to the following:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::bucket/object*"  
            ]  
        }  
    ]  
}
```

For an Amazon DynamoDB data store, additional permissions attached to the role would be 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 more information, see [Step 2: Create an IAM Role for AWS Glue \(p. 16\)](#) and [Managing access permissions for AWS Glue resources \(p. 57\)](#).

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 \(p. 143\)](#).

Destination database within the Data Catalog for the created catalog tables

For more information, see [Defining a Database in Your Data Catalog \(p. 101\)](#).

Output configuration options

Options include how the crawler should handle detected schema changes, deleted objects in the data store, and more. For more information, see [Setting Crawler Configuration Options \(p. 137\)](#).

Recrawl Policy

When crawling an Amazon S3 data source after the first crawl is complete, choose whether to crawl the entire dataset again or to crawl only folders that were added since the last crawler run. For more information, see [the section called “Incremental Crawls” \(p. 129\)](#).

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 when you create the table manually (because you already know 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 \(p. 104\)](#).

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 \(p. 140\)](#).
- 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, MongoDB, Amazon DocumentDB (with MongoDB compatibility), and relational data stores, you must specify an include path.

For Amazon S3 data stores, include path 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. The exclude pattern is relative to the include path

For MongoDB and Amazon DocumentDB (with MongoDB compatibility), the syntax is `database/collection`.

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. The 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.

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
*.csv	Matches an Amazon S3 path that represents an object name in the current folder ending in .csv
.	Matches all object names that contain a dot
*.{csv,avro}	Matches object names ending with .csv or .avro
foo.?	Matches object names starting with foo. that are followed by a single character extension
myfolder/*	Matches objects in one level of subfolder from myfolder, such as /myfolder/mysource
myfolder/**/*	Matches objects in two levels of subfolders from myfolder, such as /myfolder/mysource/data
myfolder/**	Matches objects in all subfolders of myfolder, such as /myfolder/mysource/mydata and /myfolder/mysource/data
myfolder**	Matches subfolder myfolder as well as files below myfolder, such as /myfolder and /myfolder/mydata.txt
Market*	Matches tables in a JDBC database with names that begin with Market, such as Market_us and Market_fr

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
<code>HR*</code>	Excludes the tables with names that begin with <code>HR</code>
<code>Market_*</code>	Excludes the tables with names that begin with <code>Market_</code>
<code>**_Table</code>	Excludes all tables with names that end with <code>_Table</code>

Setting Crawler Configuration Options

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

- [Setting Crawler Configuration Options on the AWS Glue Console \(p. 137\)](#)
- [Setting Crawler Configuration Options Using the API \(p. 138\)](#)
- [How to Prevent the Crawler from Changing an Existing Schema \(p. 139\)](#)
- [How to Create a Single Schema for Each Amazon S3 Include Path \(p. 140\)](#)
- [How to specify the table location and partitioning level \(p. 141\)](#)
- [How to specify configuration options for a Delta Lake data store \(p. 143\)](#)

Setting Crawler Configuration Options 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 \(p. 144\)](#).

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.

This is the default setting for incremental crawls.

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**

This is the default setting for incremental crawls.

- **Mark the table as deprecated in the Data Catalog** – This is the default setting.

Setting Crawler Configuration Options 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.

This is the default setting for incremental crawls.

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**

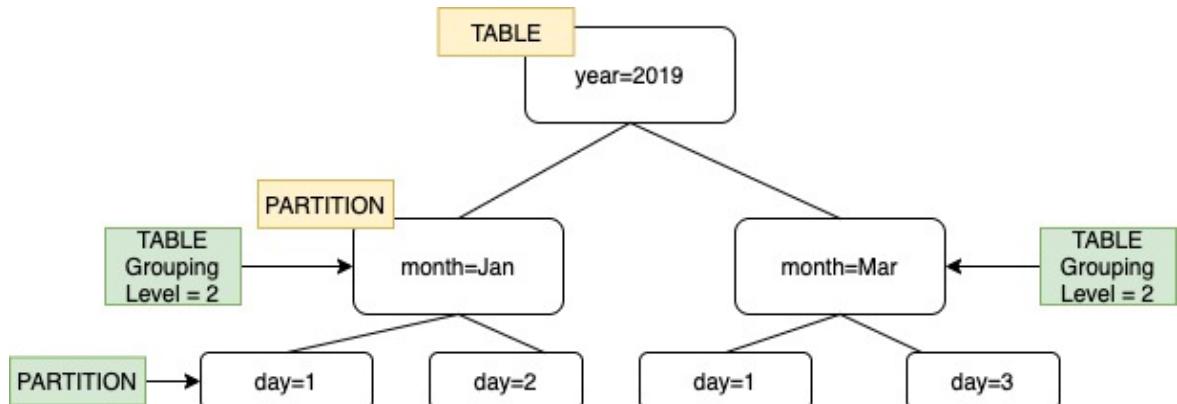
- *File content – { "C": 3, "D": 4}*
- *Schema – C: int, D: int*

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`.

How to specify the table location and partitioning level

By default, when a crawler defines tables for data stored in Amazon S3 the crawler attempts to merge schemas together and create top-level tables (`year=2019`). In some cases, you may expect the crawler to create a table for the folder `month=Jan` but instead the crawler creates a partition since a sibling folder (`month=Mar`) was merged into the same table.

The table level crawler option provides you the flexibility to tell the crawler where the tables are located, and how you want partitions created. When you specify a **Table level**, the table is created at that absolute level from the Amazon S3 bucket.



When configuring the crawler on the console, you can specify a value for the **Table level** crawler option. The value must be a positive integer that indicates the table location (the absolute level in the dataset). The level for the top level folder is 1. For example, for the path `mydataset/a/b`, if the level is set to 3, the table is created at location `mydataset/a/b`.

Console

API

When you configure the crawler using the API, set the `Configuration` field with a string representation of the following JSON object; for example:

```
configuration = jsonencode(
{
    "Version": 1.0,
    "Grouping" = {
        TableLevelConfiguration = 2
    }
})
```

CloudFormation

In this example, you set the **Table level** option available in the console within your CloudFormation template:

```
"Configuration": "{\\"Version\\":1.0,\\"Grouping\\":{\\"TableLevelConfiguration\\":2}}"
```

How to specify configuration options for a Delta Lake data store

When you configure a crawler for a Delta Lake data store, you specify these configuration parameters:

Connection

Optionally select or add a Network connection to use with this Amazon S3 target. For information about connections, see [the section called “Defining Connections in the AWS Glue Data Catalog” \(p. 110\)](#).

Enable write manifest

Select whether to detect table metadata or schema changes in the Delta Lake transaction log; it regenerates the manifest file. You should not choose this option if you configured an automatic manifest update with Delta Lake `SET TBLPROPERTIES`.

Include delta lake table path(s)

Specify one or more Amazon S3 paths to Delta tables as `s3://bucket/prefix/object`.

The screenshot shows the 'Add crawler' step of the AWS Glue wizard. On the left, a sidebar lists steps: 'Crawler info' (selected), 'Crawler source type' (Data stores selected), 'Data store' (Delta Lake selected), 'IAM Role', 'Schedule', 'Output', and 'Review all steps'. The main panel is titled 'Add a data store' and contains fields: 'Choose a data store' (Delta Lake selected), 'Connection' (test-connection selected), and 'Include delta lake table path(s)' with three entries: 's3://123', 's3://abc', and 's3://bucket/prefix/object'. Below these is a note: 'Specify the S3 location that contains the metadata and schema for your Delta lake tables. If there are multiple tables, please specify an S3 path for each table.' Under 'Enable write manifest', a checked checkbox says: 'When enabled, if the crawler detects table metadata or schema changes in the Delta Lake transaction log, it regenerates the manifest file. You should not choose this option if you configured automatic manifest updated with Delta Lake SET TBLPROPERTIES.' At the bottom are 'Back' and 'Next' buttons.

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 \(p. 301\)](#).

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 \(p. 310\)](#).

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.

When you crawl a JDBC data store, a connection is required. For more information, see [Adding an AWS Glue Connection \(p. 115\)](#). 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.

Tip

For more information about configuring crawlers, see the section called “[Crawler Properties](#)” (p. 131).

Viewing Crawler Results and Details

After the crawler runs successfully, it creates table definitions in the Data Catalog. Choose **Tables** in the navigation pane to see the tables that were created by your crawler in the database that you specified.

You can view information related to the crawler itself as follows:

- The **Crawlers** page on the AWS Glue console displays the following properties for a crawler:

Property	Description
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. 143) .

Property	Description
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.

- To view the actions and log messages for a crawler, choose **Crawlers** in the navigation pane to see the crawlers you created. 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](#).

For more information about viewing the log information, see [the section called "Automated Tools" \(p. 301\)](#) in this guide and [Querying AWS CloudTrail Logs](#) in the *Amazon Athena User Guide*. Also, see the blog [Easily query AWS service logs using Athena](#) for information about how to use the Athena Glue Service Logs (AGSlogger) Python library in conjunction with AWS Glue ETL jobs to allow a common framework for processing log data.

- To see detailed information for 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.

Accelerating Crawls Using Amazon S3 Event Notifications

Instead of listing the objects from an Amazon S3 target, you can configure the crawler to use Amazon S3 events to find any changes. This feature improves the recrawl time by using Amazon S3 events to identify the changes between two crawls by listing all the files from the subfolder which triggered the event instead of listing the full Amazon S3 target.

The first crawl lists all Amazon S3 objects from the target. After the first successful crawl, recrawls will rely on events.

The advantages of moving to an Amazon S3 event based crawler are:

- A faster recrawl as the listing of all the objects from the target is not required, instead the listing of specific folders is done where objects are added or deleted.

- A reduction in the overall crawl cost as the listing of specific folders is done where objects are added or deleted.

The Amazon S3 event crawl runs by consuming Amazon S3 events from the SQS queue based on the crawler schedule. Amazon S3 events can be configured to go directly to the SQS queue or in cases where multiple consumers need the same event, a combination of SNS and SQS. For more information, see [the section called "Setting up your Account for Amazon S3 Event Notifications" \(p. 146\)](#).

After creating and configuring the crawler in event mode, the first crawl runs in listing mode by performing full a listing of the Amazon S3 target. The following log confirms the operation of the crawl by consuming Amazon S3 events after the first successful crawl: "The crawl is running by consuming Amazon S3 events."

After creating the Amazon S3 event crawl and updating the crawler properties which may impact the crawl, the crawl operates in list mode and the following log is added: "Crawl is not running in S3 event mode".

Topics

- [Setting up your Account for Amazon S3 Event Notifications \(p. 146\)](#)
- [Setting up a Crawler for Amazon S3 Event Notifications Using the Console \(p. 151\)](#)

Setting up your Account for Amazon S3 Event Notifications

This section describes how to set up your account for Amazon S3 event notifications, and provides instructions for doing so using a script, or the AWS Glue console.

Prerequisites

Complete the following setup tasks. Note the values in parenthesis reference the configurable settings from the script.

1. Create an Amazon S3 bucket (`s3_bucket_name`).
2. Identify a crawler target (`folder_name`, such as "test1") which is a path in the identified bucket.
3. Prepare a crawler name (`crawler_name`)
4. Prepare an SNS Topic name (`sns_topic_name`) which could be the same as the crawler name.
5. Prepare the AWS Region where the crawler is to run and the S3 bucket exists (`region`).
6. Optionally prepare an email address if email is used to get the Amazon S3 events (`subscribing_email`).

Limitations:

- Only a single target is supported by the Amazon S3 event crawler.
- SQS on private VPC is not supported.
- Amazon S3 sampling is not supported.
- The crawler target should be a folder.
- The 'everything' path wildcard is not supported: `s3://%`

To use the Amazon S3 event based crawler, you should enable event notification on the S3 bucket with events filtered from the prefix which is the same as the S3 target and store in SQS. You can set up SQS and event notification through the console by following the steps in [Walkthrough: Configuring a bucket](#)

for notifications or using the [the section called “Script to generate SQS and configure Amazon S3 Events from the target” \(p. 147\)](#).

SQS Policy

Add the following SQS policy which is required to be attached to the role used by the crawler.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sns:DeleteMessage",
                "sns:GetQueueUrl",
                "sns>ListDeadLetterSourceQueues",
                "sns>DeleteMessageBatch",
                "sns:ReceiveMessage",
                "sns:GetQueueAttributes",
                "sns>ListQueueTags",
                "sns:SetQueueAttributes",
                "sns:PurgeQueue"
            ],
            "Resource": "*"
        }
    ]
}
```

Script to generate SQS and configure Amazon S3 Events from the target

After ensuring the prerequisites are met, you can run the following Python script to create the SQS. Replace the Configurable settings with the names prepared from the prerequisites.

Note

After running the script, login to the SQS console to find the ARN of the SQS created.

Amazon SQS sets a *visibility timeout*, a period of time during which Amazon SQS prevents other consumers from receiving and processing the message. Set the visibility timeout approximately equal to the crawl run time.

```
#!/venv/bin/python
import boto3
import botocore

#-----Start : READ ME FIRST -----
# 1. Purpose of this script is to create the SQS, SNS and enable S3 bucket notification.
#     The following are the operations performed by the scripts:
#         a. Enable S3 bucket notification to trigger 's3:ObjectCreated:' and
#             's3:ObjectRemoved:' events.
#         b. Create SNS topic for fan out.
#         c. Create SQS queue for saving events which will be consumed by the crawler.
#             SQS Event Queue ARN will be used to create the crawler after running the script.
# 2. This script does not create the crawler.
# 3. SNS topic is created to support FAN out of S3 events. If S3 event is also used by
#     another
#     purpose, SNS topic created by the script can be used.
# 1. Creation of bucket is an optional step.
#     To create a bucket set create_bucket variable to true.
# 2. The purpose of crawler_name is to easily locate the SQS/SNS.
#     crawler_name is used to create SQS and SNS with the same name as crawler.
```

```

# 3. 'folder_name' is the target of crawl inside the specified bucket 's3_bucket_name'
#
#-----End : READ ME FIRST -----#


#-----#
# Start : Configurable settings  #
#-----#


#Create
region = 'us-west-2'
s3_bucket_name = 's3eventtestuswest2'
folder_name = "test"
crawler_name = "test33S3Event"
sns_topic_name = crawler_name
sns_queue_name = sns_topic_name
create_bucket = False

#-----#
# End : Configurable settings  #
#-----#


# Define aws clients
dev = boto3.session.Session(profile_name='myprofile')
boto3.setup_default_session(profile_name='myprofile')
s3 = boto3.resource('s3', region_name=region)
sns = boto3.client('sns', region_name=region)
sqc = boto3.client('sqs', region_name=region)
client = boto3.client("sts")
account_id = client.get_caller_identity()["Account"]
queue_arn = ""

def print_error(e):
    print(e.message + ' RequestId: ' + e.response['ResponseMetadata']['RequestId'])

def create_s3_bucket(bucket_name, client):
    bucket = client.Bucket(bucket_name)
    try:
        if not create_bucket:
            return True
        response = bucket.create(
            ACL='private',
            CreateBucketConfiguration={
                'LocationConstraint': region
            },
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
        if 'BucketAlreadyOwnedByYou' in e.message: # we own this bucket so continue
            print('We own the bucket already. Lets continue...')
            return True
    return False

def create_s3_bucket_folder(bucket_name, client, directory_name):
    s3.put_object(Bucket=bucket_name, Key=(directory_name + '/'))

def set_s3_notification_sns(bucket_name, client, topic_arn):
    bucket_notification = client.BucketNotification(bucket_name)
    try:

        response = bucket_notification.put(
            NotificationConfiguration={
                'TopicConfigurations': [
                    {

```

```

        'Id' : crawler_name,
        'TopicArn': topic_arn,
        'Events': [
            's3:ObjectCreated:*',
            's3:ObjectRemoved:*',
        ],
        'Filter' : {'Key': {'FilterRules': [{}Name': 'prefix', 'Value':
    folder_name]}]}
    },
]
}
)
return True
except botocore.exceptions.ClientError as e:
    print_error(e)
return False

def create_sns_topic(topic_name, client):
    try:
        response = client.create_topic(
            Name=topic_name
        )
        return response['TopicArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None

def set_sns_topic_policy(topic_arn, client, bucket_name):
    try:
        response = client.set_topic_attributes(
            TopicArn=topic_arn,
            AttributeName='Policy',
            AttributeValue=''''{
                "Version": "2008-10-17",
                "Id": "s3-publish-to-sns",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": { "AWS" : "*" },
                        "Action": [ "SNS:Publish" ],
                        "Resource": "%s",
                        "Condition": {
                            "StringEquals": {
                                "AWS:SourceAccount": "%s"
                            },
                            "ArnLike": {
                                "aws:SourceArn": "arn:aws:s3:::%s"
                            }
                        }
                    }
                ]
            }''' % (topic_arn, account_id, bucket_name)
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)

    return False

def subscribe_to_sns_topic(topic_arn, client, protocol, endpoint):
    try:
        response = client.subscribe(
            TopicArn=topic_arn,
            Protocol=protocol,
            Endpoint=endpoint
    
```

```

        )
    return response['SubscriptionArn']
except botocore.exceptions.ClientError as e:
    print_error(e)
return None

def create_sqs_queue(queue_name, client):
    try:
        response = client.create_queue(
            QueueName=queue_name,
        )
    return response['QueueUrl']
except botocore.exceptions.ClientError as e:
    print_error(e)
return None

def get_sqs_queue_arn(queue_url, client):
    try:
        response = client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=[
                'QueueArn',
            ]
        )
    return response['Attributes']['QueueArn']
except botocore.exceptions.ClientError as e:
    print_error(e)
return None

def set_sqs_policy(queue_url, queue_arn, client, topic_arn):
    try:
        response = client.set_queue_attributes(
            QueueUrl=queue_url,
            Attributes={
                'Policy': '''{
                    "Version": "2012-10-17",
                    "Id": "AllowSNSPublish",
                    "Statement": [
                        {
                            "Sid": "AllowSNSPublish01",
                            "Effect": "Allow",
                            "Principal": "*",
                            "Action": "SQS:SendMessage",
                            "Resource": "%s",
                            "Condition": {
                                "ArnEquals": {
                                    "aws:SourceArn": "%s"
                                }
                            }
                        }
                    ]
                }''' % (queue_arn, topic_arn)
            }
        )
    return True
except botocore.exceptions.ClientError as e:
    print_error(e)
return False

if __name__ == "__main__":
    print('Creating S3 bucket %s.' % s3_bucket_name)
    if create_s3_bucket(s3_bucket_name, s3):
        print('\nCreating SNS topic %s.' % sns_topic_name)

```

```

topic_arn = create_sns_topic(sns_topic_name, sns)
if topic_arn:
    print('SNS topic created successfully: %s' % topic_arn)

    print('Creating SQS queue %s' % sqs_queue_name)
    queue_url = create_sqs_queue(sqs_queue_name, sqs)
    if queue_url is not None:
        print('Subscribing sqs queue with sns.')
        queue_arn = get_sqs_queue_arn(queue_url, sqs)
        if queue_arn is not None:
            if set_sqs_policy(queue_url, queue_arn, sqs, topic_arn):
                print('Successfully configured queue policy.')
                subscription_arn = subscribe_to_sns_topic(topic_arn, sns, 'sns',
queue_arn)
            if subscription_arn is not None:
                if 'pending confirmation' in subscription_arn:
                    print('Please confirm SNS subscription by visiting the
subscribe URL.')
                else:
                    print('Successfully subscribed SQS queue: ' + queue_arn)
            else:
                print('Failed to subscribe SNS')
        else:
            print('Failed to set queue policy.')
    else:
        print("Failed to get queue arn for %s" % queue_url)
# ----- End subscriptions to SNS topic -----

    print('\nSetting topic policy to allow s3 bucket %s to publish.' % s3_bucket_name)
    if set_sns_topic_policy(topic_arn, sns, s3_bucket_name):
        print('SNS topic policy added successfully.')
        if set_s3_notification_sns(s3_bucket_name, s3, topic_arn):
            print('Successfully configured event for S3 bucket %s' % s3_bucket_name)
            print('Create S3 Event Crawler using SQS ARN %s' % queue_arn)
        else:
            print('Failed to configure S3 bucket notification.')
    else:
        print('Failed to add SNS topic policy.')
else:
    print('Failed to create SNS topic.')

```

Setting up a Crawler for Amazon S3 Event Notifications Using the Console

To set up a crawler for Amazon S3 event notifications using the AWS Glue console:

1. Choose the **Crawler source type** as **Data stores**.
2. Choose **Repeat crawls of S3 data stores** as **Crawl changes identified by Amazon S3 events**.

AWS Glue Developer Guide

Using Encryption with the Amazon S3 Event Crawler

The screenshot shows the 'Add crawler' wizard in the AWS Glue console. The current step is 'Specify crawler source type'. On the left sidebar, 'Crawler info' is selected. In the main area, 'Crawler source type' is set to 'Data stores', and 'Crawl changes identified by Amazon S3 events' is selected. Buttons for 'Back' and 'Next' are visible at the bottom.

3. Click **Next**.

4. Optionally specify a **Connection** for the data store.

5. Specify the **Include path** where folders and files are crawled.

6. Specify the data store parameters including the a valid SQS ARN in **Include SQS ARN**.

For example, `arn:aws:sqs:region:account:sqs`

7. Specify a valid Amazon dead-letter SQS ARN in **Include dead-letter SQS ARN**.

For example, `arn:aws:sqs:region:account:deadLetterQueue`

The screenshot shows the 'Add crawler' wizard in the AWS Glue console. The current step is 'Add a data store'. A 'Data store' connection named 'S3' is selected. Under 'Crawl data in', 'Specified path in my account' is selected. The 'Include path' field contains 's3://bucket/prefix/object'. The 'Include SQS ARN' field contains 'arn:aws:sqs:region:account:sqs'. The 'Include dead-letter SQS ARN' field contains 'arn:aws:sqs:region:account:deadLetterQueue'. Buttons for 'Back' and 'Next' are visible at the bottom.

Using Encryption with the Amazon S3 Event Crawler

This section describes using encryption on SQS only or on both SQS and Amazon S3.

Topics

- [Enabling Encryption on SQS Only \(p. 153\)](#)
- [Enabling Encryption on Both SQS and S3 \(p. 154\)](#)
- [FAQ \(p. 156\)](#)

Enabling Encryption on SQS Only

Amazon SQS provides encryption in-transit by default. To add optional Server-Side Encryption (SSE) to your queue you can attach a [customer master key \(CMK\)](#) in the edit panel. This means that SQS encrypts all customer data at-rest on SQS servers.

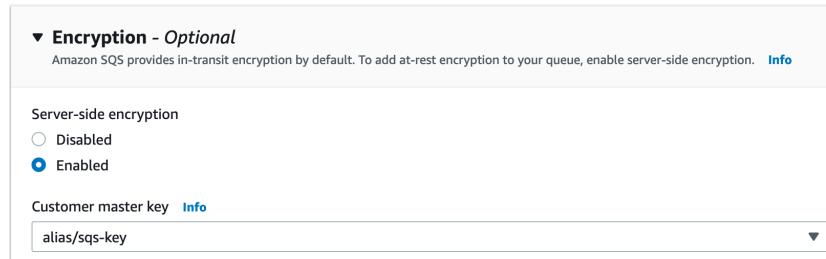
Create a Customer Master Key (CMK)

1. Choose **Key Management Service (KMS)** > **Customer Managed Keys** > **Create key**.
2. Follow the steps to add your own alias and description.
3. Add the respective IAM users/roles you would like to be able to use this key.
4. In the key policy, add another statement to the "Statement" list so that your [custom key policy](#) gives the Amazon SNS sufficient key usage permissions.

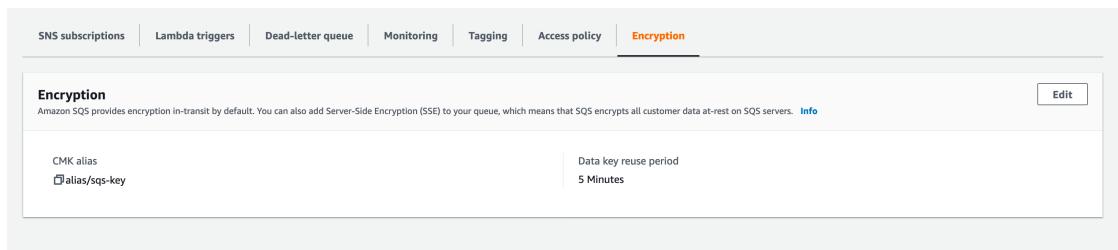
```
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "sns.amazonaws.com"
        },
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "*"
    }
]
```

Enable Server-Side Encryption (SSE) on your queue

1. Choose **Amazon SQS** > **Queues** > `sqs_queue_name` > **Encryption** tab.
2. Choose **Edit**, and scroll down to the **Encryption** drop down.
3. Select **Enabled** to add SSE.
4. Select the CMK you created earlier, and not the default key with the name `alias/aws/sqs`.



After adding this, your Encryption tab is updated with the key you added.



Note

Amazon SQS automatically deletes messages that have been in a queue for more than the maximum message retention period. The default message retention period is 4 days. To avoid missing events change the SQS **MessageRetentionPeriod** to the maximum of 14 days.

Enabling Encryption on Both SQS and S3

Enable Server-Side Encryption (SSE) on SQS

1. Follow the steps in [the section called "Enabling Encryption on SQS Only" \(p. 153\)](#).
2. In the last step of the CMK setup, give Amazon S3 sufficient key usage permissions.

Paste the following in to the "Statement" list:

```
"Statement": [  
    {  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "s3.amazonaws.com"  
        },  
        "Action": [  
            "kms:GenerateDataKey",  
            "kms:Decrypt"  
        ],  
        "Resource": "*"  
    }  
]
```

Enable Server-Side Encryption (SSE) on your S3 bucket

1. Follow the steps in [the section called "Enabling Encryption on SQS Only" \(p. 153\)](#).
2. Do one of the following:
 - To enable SSE for your entire S3 bucket, navigate to the **Properties** tab in your target bucket.

Here you can enable SSE and choose the encryption type you would like to use. Amazon S3 provides an encryption key that Amazon S3 creates, manages, and uses for you, or you can choose a key from KMS as well.

Edit default encryption

Default encryption
Automatically encrypt new objects stored in this bucket. [Learn more](#)

Server-side encryption

Disable
 Enable

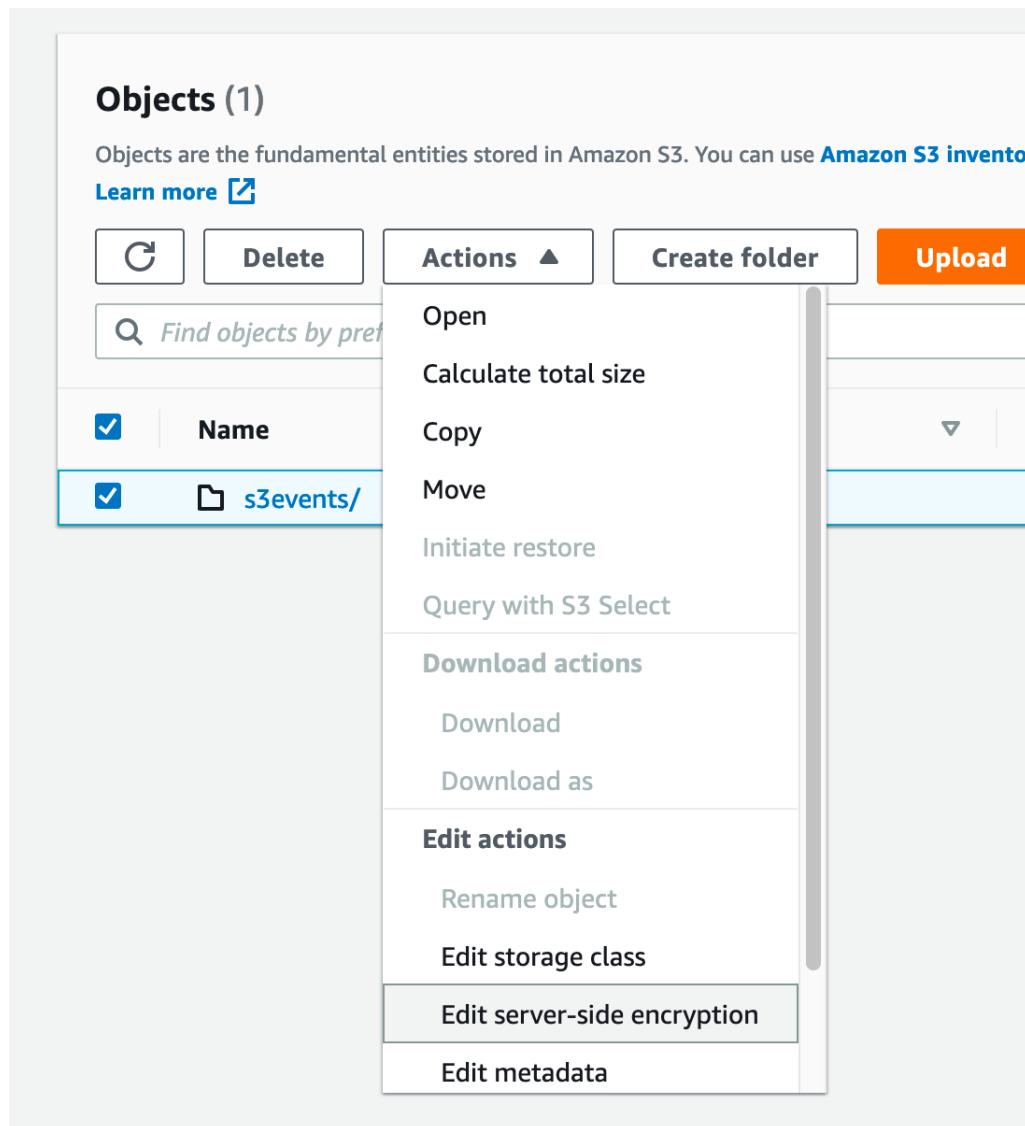
Encryption key type
To upload an object with a customer-provided encryption key (SSE-C), use the AWS CLI, AWS SDK, or Amazon S3 REST API.

Amazon S3 key (SSE-S3)
An encryption key that Amazon S3 creates, manages, and uses for you. [Learn more](#)

AWS Key Management Service key (SSE-KMS)
An encryption key protected by AWS Key Management Service (AWS KMS). [Learn more](#)

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

- To enable SSE on a specific folder, click the checkbox beside your target folder and choose **Edit server-side encryption** under the **Actions** drop down.



FAQ

Why aren't messages that I publish to my Amazon SNS topic getting delivered to my subscribed Amazon SQS queue that has server-side encryption (SSE) enabled?

Double check that your Amazon SQS queue is using:

1. A [customer master key \(CMK\)](#) that is customer managed. Not the default one provided by SQS.
2. Your CMK from (1) includes a [custom key policy](#) that gives the Amazon SNS sufficient key usage permissions.

For more information, [see this article](#) in the knowledge center.

I've subscribed to email notifications, but I don't receive any email updates when I edit my Amazon S3 bucket.

Make sure that you have confirmed your email address by clicking the "Confirm Subscription" link in your email. You can verify the status of your confirmation by checking the **Subscriptions** table under your SNS topic.

Choose **Amazon SNS > Topics > sns_topic_name > Subscriptions table**.

If you followed our prerequisite script, you will find that the `sns_topic_name` is equal to your `sqs_queue_name`. It should look similar to the following:

Subscriptions (2)				
	ID	Endpoint	Status	Protocol
<input type="radio"/>	13fa3254-a252-4621-8d5a-05b1bb9245c6	chengjes@amazon.com	Confirmed	EMAIL
<input type="radio"/>	f0a36c7b-3ec2-4890-bbe6-6503f11a9577	arn:aws:sqs:us-east-2:804918391416:s3-sqs-both-encrypted	Confirmed	SQS

Only some of the folders I added are showing up in my table after enabling server-side encryption on my SQS queue. Why am I missing some parquets?

If the Amazon S3 bucket changes were made before enabling SSE on your SQS queue, they may not be picked up by the crawler. To ensure that you have crawled all the updates to your S3 bucket, run the crawler again in listing mode ("Crawl All Folders"). Another option is to start fresh by creating a new crawler with S3 events enabled.

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 \(p. 171\)](#).

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 \(p. 160\)](#).

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. 164) .
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.

Classifier type	Classification string	Notes
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.
CSV	<code>csv</code>	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	<code>redshift</code>	Uses JDBC connection to import metadata.
MySQL	<code>mysql</code>	Uses JDBC connection to import metadata.
PostgreSQL	<code>postgresql</code>	Uses JDBC connection to import metadata.
Oracle database	<code>oracle</code>	Uses JDBC connection to import metadata.
Microsoft SQL Server	<code>sqlserver</code>	Uses JDBC connection to import metadata.
Amazon DynamoDB	<code>dynamodb</code>	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 \(p. 171\)](#).

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 \(p. 160\)](#)
- [Writing XML Custom Classifiers \(p. 164\)](#)
- [Writing JSON Custom Classifiers \(p. 165\)](#)
- [Writing CSV Custom Classifiers \(p. 170\)](#)

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. 162) 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.

```
#<noloc>&GLU;</noloc> Built-in patterns
USERNAME [a-zA-Z0-9._-]+
USER %{USERNAME:UNWANTED}
INT (?:[+-]?(?:[0-9]+))
BASE10NUM (?<![[0-9.-+]])(?>[+-]?(?:(:[0-9]+(?:\.[0-9]+)?))|(?:\.\.[0-9]+)))
NUMBER (?:%{BASE10NUM:UNWANTED})
BASE16NUM (?<![[0-9A-Fa-f]])(?:[+-]?(?:0x)?(?:[0-9A-Fa-f]+))
BASE16FLOAT \b(?<![[0-9A-Fa-f.]])(?:[+-]?(?:0x)?(?:(:[0-9A-Fa-f]+(?:\.[0-9A-Fa-f]*))|(?:\.[0-9A-Fa-f]+)))\b
BOOLEAN (?i)(true|false)

POSINT \b(?:[1-9][0-9]*)\b
NONNEGINT \b(?:[0-9]+)\b
WORD \b\w+\b
NOTSPACE \s+
SPACE \s*
DATA .?
GREEDYDATA .*
#QUOTEDSTRING (?:(?<!\\)(?:"(?:\\.|[^\\"])*")|(?:(?:\\.|[^\\'])*)'|(?:(?:\\.|[^\\~])*)~))
QUOTEDSTRING (?>(?<!\\)(?>"(?>\\.|[^\\"])+"+")|"|"|(?>'(?>\\.|[^\\']+)+')|' '|(?>`(?>\\.|[^\\`]+)+`)|``)
UUID [A-Fa-f0-9]{8}-(:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}

# Networking
MAC (?:%{CISCOMAC:UNWANTED}|%{WINDOWS_MAC:UNWANTED}|%{COMMONMAC:UNWANTED})
CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
WINDOWS_MAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
```

```

IPV6 ((([0-9A-Za-z]{1,4}):){7}([0-9A-Za-z]{1,4}|:))|(([0-9A-Za-z]{1,4}):){6}(:[0-9A-Za-z]{1,4}|((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})|:))|(([0-9A-Za-z]{1,4}):){5}((((:[0-9A-Za-z]{1,4}):{1,2})|:((25[0-5]|2[0-4])\d|1\d\d|[1-9]?|\d))|(([0-9A-Za-z]{1,4}):){4}(((:[0-9A-Za-z]{1,4}):{1,3})|(([0-9A-Za-z]{1,4}):){3}:((:[0-9A-Za-z]{1,4}):{1,4})|(([0-9A-Za-z]{1,4}):){0,2}:((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}))|:))|(([0-9A-Za-z]{1,4}):){2}((((:[0-9A-Za-z]{1,4}):{1,5})|(([0-9A-Za-z]{1,4}):){0,3}:((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}))|:))|(([0-9A-Za-z]{1,4}):){1}((((:[0-9A-Za-z]{1,4}):{1,6})|(([0-9A-Za-z]{1,4}):){0,4}:((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}))|:))|((((:[0-9A-Za-z]{1,4}):{1,7})|(([0-9A-Za-z]{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})[.](: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, 03, 12, December
MONTH \b(?:Jan(?:uary)?|Feb(?:bruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|Aug(?:uest)?|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{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](?:[.,][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} %{MONTHNUM2} %{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.

Description
Dot [dot]notated child. Specifies a child field in a JSON object.
Bracket [bracket]notated child. Specifies child field in a JSON object. Only a single child field can be specified.
Array [arrayindex.]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"
}
]
```

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-eef6e80fadec3"}  
{"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": "Alaska"}
 {"record": "#### #####"}
 {"record": "#####"}
 {"record": "Alaska"}
 {"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.

Viewing Classifiers

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.

Managing Classifiers

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.

Creating Classifiers

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.
- **Classifier type** – The classification type of tables inferred by this classifier.
- **Last updated** – The last time this classifier was updated.

Classifier name

Provide a unique name for your classifier.

Classifier type

Choose the type of classifier to create.

Depending on the type of classifier you choose, configure the following properties for your classifier:

Grok

- **Classification**

Describe the format or type of data that is classified or provide a custom label.

- **Grok pattern**

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 .*-.*-.*-.*-.*
```

XML

- **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.

For more information, see [Writing XML Custom Classifiers \(p. 164\)](#).

JSON

- **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 \(p. 165\)](#).

CSV

- **Column delimiter**

A single character or symbol to denote what separates each column entry in the row. Choose the delimiter from the list, or choose `Other` to enter a custom delimiter.

- **Quote symbol**

A single character or symbol to denote what combines content into a single column value. Must be different from the column delimiter. Choose the quote symbol from the list, or choose `Other` to enter a custom quote character.

- **Column headings**

Indicates the behavior for how column headings should be detected in the CSV file. You can choose Has headings, No headings, or Detect headings. If your custom CSV file has column headings, enter a comma-delimited list of the column headings.

- **Allow files with single column**

To be classified as CSV, the data must have at least two columns and two rows of data. Use this option to allow the processing of files that contain only one column.

- **Trim whitespace before identifying column values**

This option specifies whether to trim values before identifying the type of column values.

For more information, see [Writing Custom Classifiers \(p. 160\)](#).

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 \(p. 712\)](#).

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 AWS 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 [Defining Connections in the AWS Glue Data Catalog \(p. 110\)](#).

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 \(p. 68\)](#).

Creating Tables, Updating Schema, and Adding New Partitions in the Data Catalog from AWS Glue ETL Jobs

Your extract, transform, and load (ETL) job might create new table partitions in the target data store. Your dataset schema can evolve and diverge from the AWS Glue Data Catalog schema over time. AWS Glue ETL jobs now provide several features that you can use within your ETL script to update your schema and partitions in the Data Catalog. These features allow you to see the results of your ETL work in the Data Catalog, without having to rerun the crawler.

New Partitions

If 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.

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)

```

Now, you can create new catalog tables, update existing tables with modified schema, and add new table partitions in the Data Catalog using an AWS Glue ETL job itself, without the need to re-run crawlers.

Updating Table Schema

If you want to overwrite the Data Catalog table's schema you can do one of the following:

- When the job finishes, rerun the crawler and make sure your crawler is configured to update the table definition as well. View the new partitions on the console along with any schema updates, when the crawler finishes. For more information, see [Configuring a Crawler Using the API](#).
- When the job finishes, view the modified schema 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 `enableUpdateCatalog` set to `true`, and also `updateBehavior` set to `UPDATE_IN_DATABASE`, which indicates to overwrite the schema and add new partitions in the Data Catalog during the job run.

Python

```

additionalOptions = {
  "enableUpdateCatalog": True,
  "updateBehavior": "UPDATE_IN_DATABASE"}
additionalOptions["partitionKeys"] = ["partition_key0", "partition_key1"]

```

```
sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
    database=<dst_db_name>,
    table_name=<dst_tbl_name>, transformation_ctx="write_sink",
    additional_options=additionalOptions)
job.commit()
```

Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("partition_0", "partition_1"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(database = nameSpace, tableName = tableName,
    additionalOptions = options)
sink.writeDynamicFrame(df)
```

You can also set the `updateBehavior` value to `LOG` if you want to prevent your table schema from being overwritten, but still want to add the new partitions. The default value of `updateBehavior` is `UPDATE_IN_DATABASE`, so if you don't explicitly define it, then the table schema will be overwritten.

If `enableUpdateCatalog` is not set to true, regardless of whichever option selected for `updateBehavior`, the ETL job will not update the table in the Data Catalog.

Creating New Tables

You can also use the same options to create a new table in the Data Catalog. You can specify the database and new table name using `setCatalogInfo`.

Python

```
sink = glueContext.getSink(connection_type="s3", path="s3://path/to/data",
    enableUpdateCatalog=True, updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=["partition_key0", "partition_key1"])
sink.setFormat("<format>")
sink.setCatalogInfo(catalogDatabase=<dst_db_name>, catalogTableName=<dst_tbl_name>)
sink.writeFrame(last_transform)
```

Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("<partition_1>", "<partition_2>"),
    "enableUpdateCatalog" -> true,
    "updateBehavior" -> "UPDATE_IN_DATABASE"))
val sink = glueContext.getSink(connectionType = "s3", connectionOptions =
    options).withFormat("<format>")
sink.setCatalogInfo(catalogDatabase = "<dst_db_name>", catalogTableName =
    "<dst_tbl_name>")
sink.writeDynamicFrame(df)
```

Restrictions

Take note of the following restrictions:

- Only Amazon Simple Storage Service (Amazon S3) targets are supported.
- Only the following formats are supported: `json`, `csv`, `avro`, and `parquet`.

- To create or update tables with the `parquet` classification, you must utilize the AWS Glue optimized parquet writer for `DynamicFrames`. This can be achieved in one of three ways:
 - Call `write_dynamic_frame_from_catalog()`, then set a `useGlueParquetWriter` table property to true in the table you are updating.
 - Call `getSink()` in your script with `connection_type="s3"`, then set your format to `glueparquet`.
 - Call `getSink()` in your script with `connection_type="s3"`, then set your format to `parquet` and pass a `useGlueParquetWriter` property as true in your `format_options`, this is especially useful for creating new parquet tables.
- When the `updateBehavior` is set to `LOG`, new partitions will be added only if the `DynamicFrame` schema is equivalent to or contains a subset of the columns defined in the Data Catalog table's schema.
- Your `partitionKeys` must be equivalent, and in the same order, between your parameter passed in your ETL script and the `partitionKeys` in your Data Catalog table schema.
- This feature currently does not yet support updating/creating tables in which the updating schemas are nested (for example, arrays inside of structs).

For more information, see [ETL Programming \(p. 472\)](#).

Creating AWS Glue resources 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. 183) , JSON classifier (p. 183) , XML classifier (p. 184)
Connection	AWS::Glue::Connection	MySQL connection (p. 186)
Crawler	AWS::Glue::Crawler	Amazon S3 crawler (p. 185) , MySQL crawler (p. 187)
Database	AWS::Glue::Database	Empty database (p. 179) , Database with tables (p. 180)

AWS Glue Resource	AWS CloudFormation Template	AWS Glue Samples
Development endpoint	AWS::Glue::DevEndpoint	Development endpoint (p. 194)
Job	AWS::Glue::Job	Amazon S3 job (p. 189), JDBC job (p. 190)
Partition	AWS::Glue::Partition	Partitions of a table (p. 180)
Table	AWS::Glue::Table	Table in a database (p. 180)
Trigger	AWS::Glue::Trigger	On-demand trigger (p. 191), Scheduled trigger (p. 192), Conditional trigger (p. 193)

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-mysampledatabse

# 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.ql.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.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=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.
# 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  
CFNJDBCPath:  
  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-on-demand-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 \(p. 39\)](#).

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:

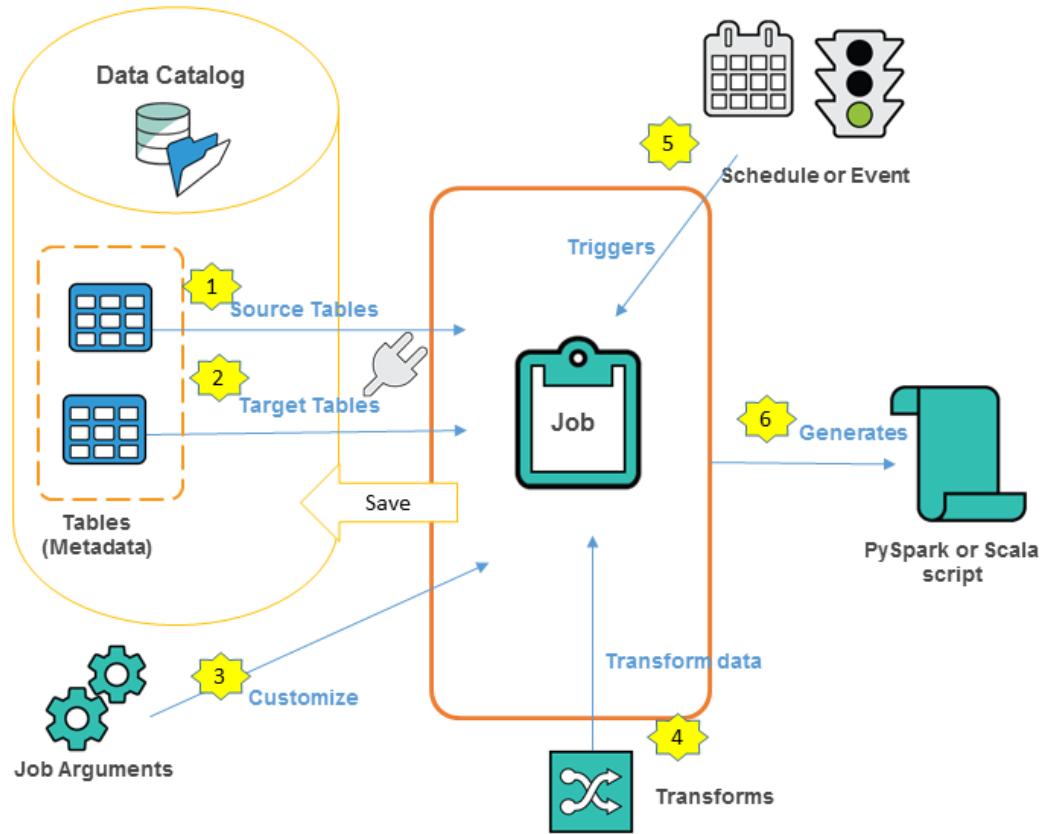
# The name of the crawler to be created
CFNEndpointName:
  Type: String
  Default: cfn-devendpoint-1
CFNIAMRoleArn:
  Type: String
  Default: arn:aws:iam::123456789012:role/AWSGlueServiceRoleGA
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNDevEndpoint:
    Type: AWS::Glue::DevEndpoint
    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 \(p. 216\)](#).

The following diagram summarizes the basic workflow and steps involved in authoring a job in AWS Glue:



Topics

- [Workflow Overview \(p. 197\)](#)
- [Adding Jobs in AWS Glue \(p. 197\)](#)
- [Editing Scripts in AWS Glue \(p. 225\)](#)
- [Working with MongoDB Connections in ETL Jobs \(p. 229\)](#)
- [Interactive applications and script development \(p. 230\)](#)
- [Developing Scripts Using Development Endpoints \(p. 252\)](#)
- [Managing Notebooks \(p. 285\)](#)

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 \(p. 197\)](#).
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 \(p. 214\)](#).
5. You specify how your job is invoked, either on demand, by a time-based schedule, or by an event. For more information, see [Starting Jobs and Crawlers Using Triggers \(p. 296\)](#).
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 \(p. 225\)](#).

Adding Jobs in AWS Glue

An AWS Glue job encapsulates a script that connects to your source data, processes it, and then writes it out to your data target. Typically, a job runs extract, transform, and load (ETL) scripts. Jobs can also run general-purpose Python scripts (Python shell jobs.) AWS Glue *triggers* can start jobs based on a schedule or event, or on demand. You can monitor job runs to understand runtime metrics such as completion status, duration, and start time.

You can use scripts that AWS Glue generates or you can provide your own. With a source schema and target location or schema, the AWS Glue code generator can automatically create an Apache Spark API (PySpark) script. 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 three types of jobs in AWS Glue: *Spark*, *Streaming ETL*, and *Python shell*.

- A Spark job is run in an Apache Spark environment managed by AWS Glue. It processes data in batches.
- A streaming ETL job is similar to a Spark job, except that it performs ETL on data streams. It uses the Apache Spark Structured Streaming framework. Some Spark job features are not available to streaming ETL jobs.
- A Python shell job runs Python scripts as a shell and supports a Python version that depends on the AWS Glue version you are using. You can use these jobs to schedule and run tasks that don't require an Apache Spark environment.

Defining Job Properties for Spark Jobs

When you define your job on the AWS Glue console, you provide values for properties to control the AWS Glue runtime environment.

The following list describes the properties of a Spark job. For the properties of a Python shell job, see [Defining Job Properties for Python Shell Jobs \(p. 202\)](#). For properties of a streaming ETL job, see [the section called "Defining Job Properties for a Streaming ETL Job" \(p. 213\)](#).

The properties are listed in the order in which they appear on the **Add job** wizard on AWS Glue console.

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 \(p. 57\)](#).

Type

Specify the type of job environment to run:

- Choose **Spark** to run an Apache Spark ETL script with the job command `glueetl`.
- Choose **Spark Streaming** to run a Apache Spark streaming ETL script with the job command `gluestreaming`. For more information, see [the section called "Adding Streaming ETL Jobs" \(p. 206\)](#).
- Choose **Python shell** to run a Python script with the job command `pythonshell`. For more information, see [Adding Python Shell Jobs in AWS Glue \(p. 202\)](#).

AWS Glue version

AWS Glue version determines the versions of Apache Spark and Python that are available to the job, as specified in the following table.

AWS Glue version	Supported Spark and Python versions
0.9	<ul style="list-style-type: none">Spark 2.2.1Python 2.7
1.0	<ul style="list-style-type: none">Spark 2.4.3Python 2.7Python 3.6
2.0	<ul style="list-style-type: none">Spark 2.4.3Python 3.7
3.0	<ul style="list-style-type: none">Spark 3.1.1Python 3.7

Jobs that were created without specifying a AWS Glue version default to AWS Glue 2.0.

This job runs (generated or custom script)

The code in the ETL script defines your job's 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 writing scripts, see [Editing Scripts in AWS Glue \(p. 225\)](#).

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.

Note

AWS Glue creates a temporary bucket for jobs if a bucket doesn't already exist in a region. This bucket might permit public access. You can either modify the bucket in Amazon S3 to set the public access block, or delete the bucket later after all jobs in that region have completed.

Note

Source and target are not listed under the console **Details** tab for a job. Review the script to see source and target details.

Advanced Properties

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. For more information, see [the section called "Tracking Processed Data Using Job Bookmarks" \(p. 303\)](#).

Monitoring options

Job metrics

Turn on or turn off 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 turn on and visualize metrics, see [Job Monitoring and Debugging \(p. 349\)](#).

Continuous logging

Turn on continuous logging to Amazon CloudWatch. If this option is not enabled, logs are available only after the job completes. For more information, see [the section called "Continuous Logging for AWS Glue Jobs" \(p. 345\)](#).

Spark UI

Turn on the use of Spark UI for monitoring this job. For more information, see [Enabling the Apache Spark Web UI for AWS Glue Jobs \(p. 319\)](#).

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 \(p. 310\)](#).

Security configuration, script libraries, and job parameters

Security configuration

Choose a security configuration from the list. A security configuration specifies how the data at the Amazon S3 target is encrypted: no encryption, server-side encryption with AWS KMS-managed keys (SSE-KMS), or Amazon S3-managed encryption keys (SSE-S3).

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. This option is passed as a job parameter. For more information, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#) in the *Amazon Simple Storage Service User Guide*.

Important

This option is ignored if a security configuration is specified.

Python library path, Dependent jars path, and Referenced files path

Specify these options if your script requires them. You can define the comma-separated Amazon S3 paths for these options when you define the job. You can override these paths when you run the job. For more information, see [Providing Your Own Custom Scripts \(p. 228\)](#).

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. This is the default **Worker type** for AWS Glue Version 2.0 or later 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 machine learning 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](#).

For AWS Glue version 1.0 or earlier jobs, 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 AWS Glue version 2.0 or later jobs, you cannot instead specify a **Maximum capacity**. Instead, you should specify a **Worker type** and the **Number of workers**. For more information, see [Jobs](#).

Number of workers

For G.1X and G.2X worker types, you must specify the number of workers that are allocated when the job runs.

Maximum capacity

For AWS Glue version 1.0 or earlier jobs, using the standard worker type, you must specify the maximum number of AWS Glue data processing units (DPUs) that can be allocated when this job runs. 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](#).

Choose an integer from 2 to 100. The default is 10. This job type cannot have a fractional DPU allocation.

For AWS Glue version 2.0 or later jobs, you cannot instead specify a **Maximum capacity**. Instead, you should specify a **Worker type** and the **Number of workers**. For more information, see [Jobs](#).

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 (48 hours) for batch jobs. When the job execution time exceeds this limit, the job run state changes to **TIMEOUT**.

For streaming jobs that run indefinitely, leave the value blank, which is the default value for streaming jobs.

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. Jobs that reach the timeout limit are not restarted.

Job parameters

A set of key-value pairs that are passed as named parameters to the script. These are default values that are used when the script is run, but you can override them in triggers or when you run the job. You must prefix the key name with `--`; for example: `--myKey`. You pass job parameters as a map when using the AWS Command Line Interface.

For examples, see Python parameters in [Passing and Accessing Python Parameters in AWS Glue \(p. 535\)](#).

Non-overridable Job parameters

A set of special job parameters that cannot be overridden in triggers or when you run the job. These key-value pairs are passed as named parameters to the script. These values are used when the script is run, and you cannot override them in triggers or when you run the job. You must prefix the key name with `--`; for example: `--myKey`. `getResolvedOptions()` returns both job parameters and non-overridable job parameters in a single map. For more information, see [the section called “getResolvedOptions” \(p. 553\)](#).

Catalog options

Use AWS Glue data catalog as the Hive metastore

Select to use the AWS Glue Data Catalog as the Hive metastore. The IAM role used for the job must have the `glue:CreateDatabase` permission. A database called “default” is created in the Data Catalog if it does not exist.

Data source

Choose a Data Catalog table.

Transform type

Choose one of the following:

Change schema

Adds the `ApplyMapping` transform to the generated script. Allows you to change the schema of the source data and create a new target dataset.

Find matching records

Allows you to choose a machine learning transform to use to find matching records within your source data.

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.

For more information about adding a job using the AWS Glue console, see [Working with Jobs on the AWS Glue Console \(p. 216\)](#).

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 \(p. 202\)](#)
- [Supported Libraries for Python Shell Jobs \(p. 203\)](#)
- [Limitations \(p. 204\)](#)
- [Providing Your Own Python Library \(p. 204\)](#)

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" \(p. 216\)](#)), 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 \(p. 57\)](#).

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 \(p. 225\)](#).

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 for Spark Jobs \(p. 198\)](#). For more information about adding a job using the AWS Glue console, see [Working with Jobs on the AWS Glue Console \(p. 216\)](#).

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_example` 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.

When modifying or renaming .egg files, the file names must use the default names generated by the "python setup.py bdist_egg" command or must adhere to the Python module naming conventions. For more information, see the [Style Guide for Python Code](#).

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.

Adding Streaming ETL Jobs in AWS Glue

You can create streaming extract, transform, and load (ETL) jobs that run continuously, consume data from streaming sources like Amazon Kinesis Data Streams, Apache Kafka, and Amazon Managed Streaming for Apache Kafka (Amazon MSK). The jobs cleanse and transform the data, and then load the results into Amazon S3 data lakes or JDBC data stores.

By default, AWS Glue processes and writes out data in 100-second windows. This allows data to be processed efficiently and permits aggregations to be performed on data arriving later than expected. You can modify this window size to increase timeliness or aggregation accuracy. AWS Glue streaming jobs use checkpoints rather than job bookmarks to track the data that has been read.

Note

AWS Glue bills hourly for streaming ETL jobs while they are running.

Creating a streaming ETL job involves the following steps:

1. For an Apache Kafka streaming source, create an AWS Glue connection to the Kafka source or the Amazon MSK cluster.
2. Manually create a Data Catalog table for the streaming source.
3. Create an ETL job for the streaming data source. Define streaming-specific job properties, and supply your own script or optionally modify the generated script.

For more information, see [Streaming ETL in AWS Glue \(p. 10\)](#).

When creating a streaming ETL job for Amazon Kinesis Data Streams, you don't have to create an AWS Glue connection. However, if there is a connection attached to the AWS Glue streaming ETL job that has Kinesis Data Streams as a source, then a virtual private cloud (VPC) endpoint to Kinesis is required. For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*. When specifying a

Amazon Kinesis Data Streams stream in another account, you must setup the roles and policies to allow cross-account access. For more information, see [Example: Read From a Kinesis Stream in a Different Account](#).

AWS Glue streaming ETL jobs can auto-detect compressed data, transparently decompress the streaming data, perform the usual transformations on the input source, and load to the output store.

AWS Glue supports auto-decompression for the following compression types given the input format:

Compression type	Avro file	Avro datum	JSON	CSV	Grok
BZIP2	Yes	Yes	Yes	Yes	Yes
GZIP	No	Yes	Yes	Yes	Yes
SNAPPY	Yes (raw Snappy)	Yes (framed Snappy)	Yes (framed Snappy)	Yes (framed Snappy)	Yes (framed Snappy)
XZ	Yes	Yes	Yes	Yes	Yes
ZSTD	Yes	No	No	No	No
DEFLATE	Yes	Yes	Yes	Yes	Yes

Topics

- [Creating an AWS Glue Connection for an Apache Kafka Data Stream \(p. 207\)](#)
- [Creating a Data Catalog Table for a Streaming Source \(p. 208\)](#)
- [Notes and Restrictions for Avro Streaming Sources \(p. 210\)](#)
- [Applying Grok Patterns to Streaming Sources \(p. 211\)](#)
- [Defining Job Properties for a Streaming ETL Job \(p. 213\)](#)
- [Streaming ETL Notes and Restrictions \(p. 214\)](#)

Creating an AWS Glue Connection for an Apache Kafka Data Stream

To read from an Apache Kafka stream, you must create an AWS Glue connection.

To create an AWS Glue connection for a Kafka source (Console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Connections**.
3. Choose **Add connection**, and on the **Set up your connection's properties** page, enter a connection name.

Note

For more information about specifying connection properties, see [AWS Glue connection properties..](#)

4. For **Connection type**, choose **Kafka**.
5. For **Kafka bootstrap servers URLs**, enter the host and port number for the bootstrap brokers for your Amazon MSK cluster or Apache Kafka cluster. Use only Transport Layer Security (TLS) endpoints for establishing the initial connection to the Kafka cluster. Plaintext endpoints are not supported.

The following is an example list of hostname and port number pairs for an Amazon MSK cluster.

```
myserver1.kafka.us-east-1.amazonaws.com:9094,myserver2.kafka.us-east-1.amazonaws.com:9094,  
myserver3.kafka.us-east-1.amazonaws.com:9094
```

For more information about getting the bootstrap broker information, see [Getting the Bootstrap Brokers for an Amazon MSK Cluster](#) in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

6. If you want a secure connection to the Kafka data source, select **Require SSL connection**, and for **Kafka private CA certificate location**, enter a valid Amazon S3 path to a custom SSL certificate.

For an SSL connection to self-managed Kafka, the custom certificate is mandatory. It's optional for Amazon MSK.

For more information about specifying a custom certificate for Kafka, see [the section called "AWS Glue Connection SSL Properties" \(p. 114\)](#).

7. Optionally enter a description, and then choose **Next**.
8. For an Amazon MSK cluster, specify its virtual private cloud (VPC), subnet, and security group. The VPC information is optional for self-managed Kafka.
9. Choose **Next** to review all connection properties, and then choose **Finish**.

For more information about AWS Glue connections, see [the section called "AWS Glue Connections" \(p. 110\)](#).

Creating a Data Catalog Table for a Streaming Source

A Data Catalog table that specifies source data stream properties, including the data schema can be manually created for a streaming source. This table is used as the data source for the streaming ETL job.

If you don't know the schema of the data in the source data stream, you can create the table without a schema. Then when you create the streaming ETL job, you can turn on the AWS Glue schema detection function. AWS Glue determines the schema from the streaming data.

Use the [AWS Glue console](#), the AWS Command Line Interface (AWS CLI), or the AWS Glue API to create the table. For information about creating a table manually with the AWS Glue console, see [the section called "Defining Tables in the AWS Glue Data Catalog" \(p. 102\)](#).

Note

You can't use the AWS Lake Formation console to create the table; you must use the AWS Glue console.

Also consider the following information for streaming sources in Avro format or for log data that you can apply Grok patterns to.

- [the section called "Notes and Restrictions for Avro Streaming Sources" \(p. 210\)](#)
- [the section called "Applying Grok Patterns to Streaming Sources" \(p. 211\)](#)

Topics

- [Kinesis data source \(p. 208\)](#)
- [Kafka data source \(p. 210\)](#)
- [AWS Glue Schema Registry table source \(p. 210\)](#)

Kinesis data source

When creating the table, set the following streaming ETL properties (console).

Type of Source

Kinesis

For a Kinesis source in the same account:

Region

The AWS Region where the Amazon Kinesis Data Streams service resides. The Region and Kinesis stream name are together translated to a Stream ARN.

Example: <https://kinesis.us-east-1.amazonaws.com>

Kinesis stream name

Stream name as described in [Creating a Stream](#) in the *Amazon Kinesis Data Streams Developer Guide*.

For a Kinesis source in another account, refer to [this example](#) to set up the roles and policies to allow cross-account access. Configure these settings:

Stream ARN

The ARN of the Kinesis data stream that the consumer is registered with. For more information, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#) in the *AWS General Reference*.

Assumed Role ARN

The Amazon Resource Name (ARN) of the role to assume.

Session name (optional)

An identifier for the assumed role session.

Use the role session name to uniquely identify a session when the same role is assumed by different principals or for different reasons. In cross-account scenarios, the role session name is visible to, and can be logged by the account that owns the role. The role session name is also used in the ARN of the assumed role principal. This means that subsequent cross-account API requests that use the temporary security credentials will expose the role session name to the external account in their AWS CloudTrail logs.

To set streaming ETL properties for Amazon Kinesis Data Streams (AWS Glue API or AWS CLI)

- To set up streaming ETL properties for a Kinesis source in the same account, specify the `streamName` and `endpointUrl` parameters in the `StorageDescriptor` structure of the `CreateTable` API operation or the `create_table` CLI command.

```
"StorageDescriptor": {  
    "Parameters": {  
        "typeOfData": "kinesis",  
        "streamName": "sample-stream",  
        "endpointUrl": "https://kinesis.us-east-1.amazonaws.com"  
    }  
    ...  
}
```

Or, specify the `streamARN`.

Example

```
"StorageDescriptor": {  
    "Parameters": {  
        "typeOfData": "kinesis",  
    }  
}
```

```
    "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream"
}
...
}
```

- To set up streaming ETL properties for a Kinesis source in another account, specify the `streamARN`, `awsSTSRoleARN` and `awsSTSSessionName` (optional) parameters in the `StorageDescriptor` structure in the `CreateTable` API operation or the `create_table` CLI command.

```
"StorageDescriptor": {
  "Parameters": {
    "typeOfData": "kinesis",
    "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream",
    "awsSTSRoleARN": "arn:aws:iam::123456789:role/sample-assume-role-arn",
    "awsSTSSessionName": "optional-session"
  }
}
...
```

Kafka data source

When creating the table, set the following streaming ETL properties (console).

Type of Source

Kafka

For a Kafka source:

Topic name

Topic name as specified in Kafka.

Connection

An AWS Glue connection that references a Kafka source, as described in [the section called “Creating a Connection for a Kafka Data Stream” \(p. 207\)](#).

AWS Glue Schema Registry table source

To use AWS Glue Schema Registry for streaming jobs, follow the instructions at [Use Case: AWS Glue Data Catalog \(p. 463\)](#) to create or update a Schema Registry table.

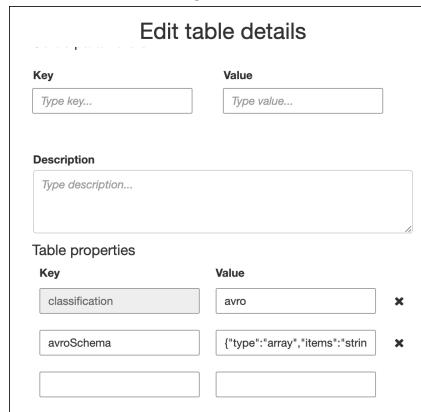
Currently, AWS Glue Streaming supports only Glue Schema Registry Avro format with schema inference set to `false`.

Notes and Restrictions for Avro Streaming Sources

The following notes and restrictions apply for streaming sources in the Avro format:

- When schema detection is turned on, the Avro schema must be included in the payload. When turned off, the payload should contain only data.
- Some Avro data types are not supported in dynamic frames. You can't specify these data types when defining the schema with the **Define a schema** page in the create table wizard in the AWS Glue console. During schema detection, unsupported types in the Avro schema are converted to supported types as follows:
 - `EnumType => StringType`
 - `FixedType => BinaryType`
 - `UnionType => StructType`

- If you define the table schema using the **Define a schema** page in the console, the implied root element type for the schema is `record`. If you want a root element type other than `record`, for example `array` or `map`, you can't specify the schema using the **Define a schema** page. Instead you must skip that page and specify the schema either as a table property or within the ETL script.
- To specify the schema in the table properties, complete the create table wizard, edit the table details, and add a new key-value pair under **Table properties**. Use the key `avroSchema`, and enter a schema JSON object for the value, as shown in the following screenshot.



- To specify the schema in the ETL script, modify the `datasource0` assignment statement and add the `avroSchema` key to the `additional_options` argument, as shown in the following Python and Scala examples.

Python

```
SCHEMA_STRING = '{"type": "array", "items": "string"}'
datasource0 = glueContext.create_data_frame.from_catalog(database = "database",
    table_name = "table_name", transformation_ctx = "datasource0", additional_options
    = {"startingPosition": "TRIM_HORIZON", "inferSchema": "false", "avroSchema": SCHEMA_STRING})
```

Scala

```
val SCHEMA_STRING = """{"type": "array", "items": "string"}"""
val datasource0 = glueContext.getCatalogSource(database = "database", tableName
    = "table_name", redshiftTmpDir = "", transformationContext = "datasource0",
    additionalOptions = JsonOptions(s"""{"startingPosition": "TRIM_HORIZON",
    "inferSchema": "false", "avroSchema": "$SCHEMA_STRING"}""").getDataFrame()
```

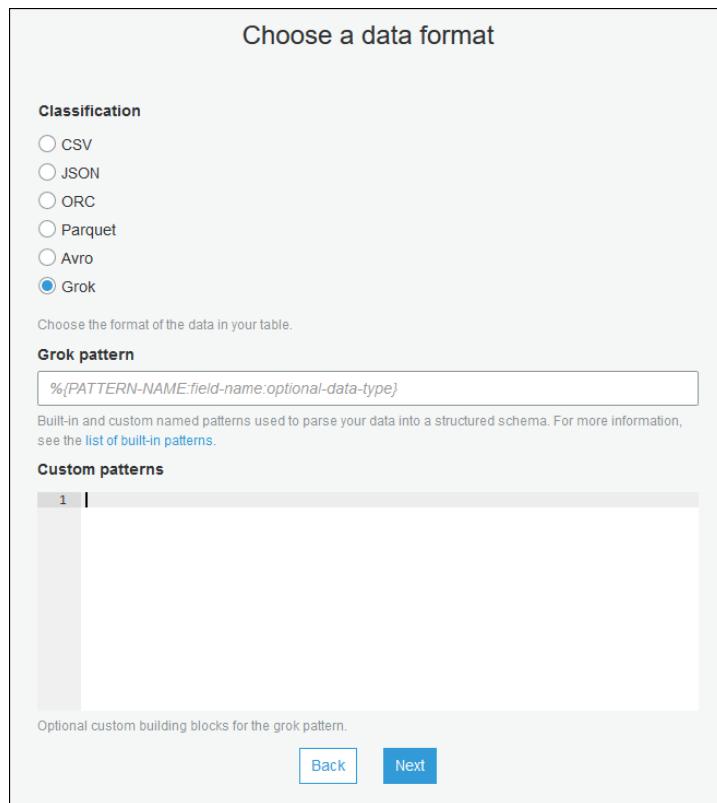
Applying Grok Patterns to Streaming Sources

You can create a streaming ETL job for a log data source and use Grok patterns to convert the logs to structured data. The ETL job then processes the data as a structured data source. You specify the Grok patterns to apply when you create the Data Catalog table for the streaming source.

For information about Grok patterns and custom pattern string values, see [Writing Grok Custom Classifiers \(p. 160\)](#).

To add Grok patterns to the Data Catalog table (console)

- Use the create table wizard, and create the table with the parameters specified in the section called ["Creating a Data Catalog Table for a Streaming Source" \(p. 208\)](#). Specify the data format as Grok, fill in the **Grok pattern** field, and optionally add custom patterns under **Custom patterns (optional)**.



Press **Enter** after each custom pattern.

To add Grok patterns to the Data Catalog table (AWS Glue API or AWS CLI)

- Add the `GrokPattern` parameter and optionally the `CustomPatterns` parameter to the `CreateTable` API operation or the `create_table` CLI command.

```
"Parameters": {  
...  
    "grokPattern": "string",  
    "grokCustomPatterns": "string",  
...  
},
```

Express `grokCustomPatterns` as a string and use "`\n`" as the separator between patterns.

The following is an example of specifying these parameters.

Example

```
"parameters": {  
...  
    "grokPattern": "%{USERNAME:username} %{DIGIT:digit:int}",  
    "grokCustomPatterns": "digit \\d",  
...  
}
```

Defining Job Properties for a Streaming ETL Job

When you define a streaming ETL job in the AWS Glue console, provide the following streams-specific properties. For descriptions of additional job properties, see [Defining Job Properties for Spark Jobs \(p. 198\)](#). For more information about adding a job using the AWS Glue console, see [Working with Jobs on the AWS Glue Console \(p. 216\)](#).

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, access streaming sources, and access target data stores.

For access to Amazon Kinesis Data Streams, attach the `AmazonKinesisFullAccess` AWS managed policy to the role, or attach a similar IAM policy that permits more fine-grained access. For sample policies, see [Controlling Access to Amazon Kinesis Data Streams Resources Using IAM](#).

For more information about permissions for running jobs in AWS Glue, see [Managing access permissions for AWS Glue resources \(p. 57\)](#).

Type

Choose **Spark streaming**.

AWS Glue version

The AWS Glue version determines the versions of Apache Spark, and Python or Scala, that are available to the job. Choose a selection for Glue Version 1.0 or Glue Version 2.0 that specifies the version of Python or Scala available to the job. AWS Glue Version 2.0 with Python 3 support is the default for streaming ETL jobs.

Job timeout

Optionally enter a duration in minutes. The default value is blank, which means the job might run indefinitely.

Data source

Specify the table that you created in [the section called “Creating a Data Catalog Table for a Streaming Source \(p. 208\)](#).

Data target

Do one of the following:

- Choose **Create tables in your data target** and specify the following data target properties.

Data store

Choose Amazon S3 or JDBC.

Format

Choose any format. All are supported for streaming.

- Choose **Use tables in the data catalog and update your data target**, and choose a table for a JDBC data store.

Output schema definition

Do one of the following:

- Choose **Automatically detect schema of each record** to turn on schema detection. AWS Glue determines the schema from the streaming data.
- Choose **Specify output schema for all records** to use the Apply Mapping transform to define the output schema.

Script

Optionally supply your own script or modify the generated script to perform operations that the Apache Spark Structured Streaming engine supports. For information on the available operations, see [Operations on streaming DataFrames/Datasets](#).

Streaming ETL Notes and Restrictions

Keep in mind the following notes and restrictions:

- Auto-decompression for AWS Glue streaming ETL jobs is only available for the supported compression types. Also note the following:
 - Framed Snappy refers to the official [framing format](#) for Snappy.
 - Deflate is supported in Glue version 3.0, not Glue version 2.0.
 - When using schema detection, you cannot perform joins of streaming data.
 - Your ETL script can use AWS Glue's built-in transforms and the transforms native to Apache Spark Structured Streaming. For more information, see [Operations on streaming DataFrames/Datasets](#) on the Apache Spark website or [Built-In Transforms \(p. 214\)](#).
 - AWS Glue streaming ETL jobs use checkpoints to keep track of the data that has been read. Therefore, a stopped and restarted job picks up where it left off in the stream. If you want to reprocess data, you can delete the checkpoint folder referenced in the script.
 - Job bookmarks aren't supported.
 - You cannot register a job as a consumer for the enhanced fan-out feature of Kinesis Data Streams.
 - If you use a Data Catalog table created from AWS Glue Schema Registry, when a new schema version becomes available, to reflect the new schema, you need to do the following:
 1. Stop the jobs associated with the table.
 2. Update the schema for the Data Catalog table.
 3. Restart the jobs associated with the table.

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 \(p. 588\)](#).

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`.

For examples of using these transforms in a job script, see the AWS blog [Building an AWS Glue ETL pipeline locally without an AWS account](#).

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 \(p. 228\)](#).

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 source 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 \(p. 61\)](#).

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 \(p. 16\)](#).

Important

Check [Troubleshooting Errors in AWS Glue \(p. 989\)](#) for known problems when a job runs.

To learn about the properties that are required for each job, see [Defining Job Properties for Spark Jobs \(p. 198\)](#).

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 turn on metrics and interpret the graphs, see [Job Monitoring and Debugging \(p. 349\)](#).

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 \(p. 227\)](#). For information about the functions that are called in your script, see [Program AWS Glue ETL Scripts in Python \(p. 533\)](#).

Metrics

The **Metrics** tab shows metrics collected when a job runs and profiling is turned on. 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

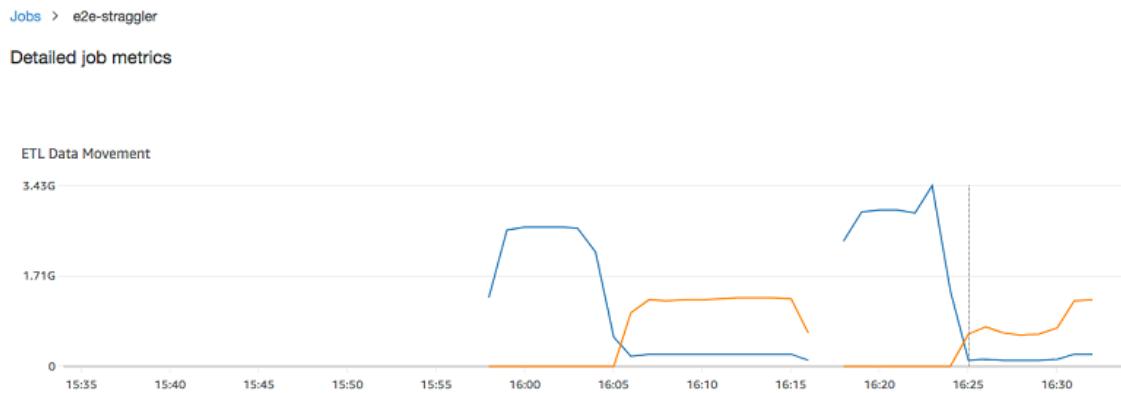
- Data Shuffle Across Executors
- CPU Load: Driver and Executors
- Job Execution: Active Executors, Completed Stages & Maximum Needed Executors

Data for these graphs is pushed to CloudWatch metrics if the job is configured to collect metrics. For more information about how to turn on metrics and interpret the graphs, see [Job Monitoring and Debugging \(p. 349\)](#).

Example of ETL Data Movement Graph

The ETL Data Movement graph shows the following metrics:

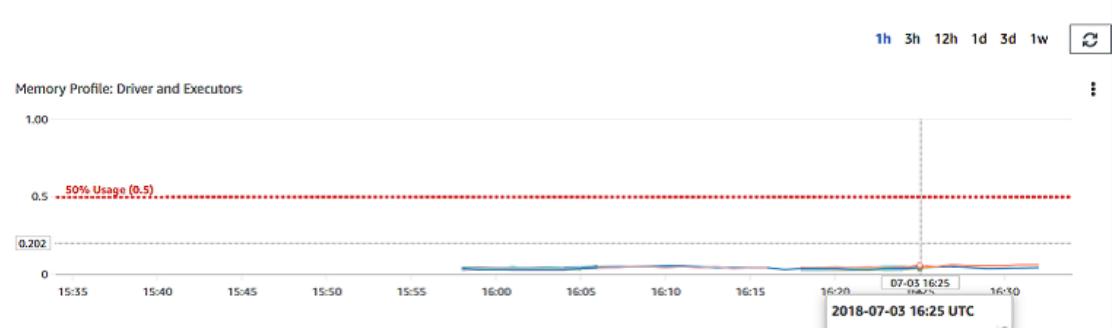
- The number of bytes read from Amazon S3 by all executors
—[glue.ALL.s3.filesystem.read_bytes \(p. 342\)](#)
- The number of bytes written to Amazon S3 by all executors
—[glue.ALL.s3.filesystem.write_bytes \(p. 343\)](#)



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 \(p. 340\)](#)
 - [glue.executorId.jvm.heap.usage \(p. 340\)](#)
 - [glue.ALL.jvm.heap.usage \(p. 340\)](#)



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 \(p. 336\)](#)
- The number of bytes written by all executors to shuffle data between them
—[glue.driver.aggregate.shuffleBytesWritten \(p. 336\)](#)



Example of CPU Load Graph

The CPU Load graph shows the following metrics:

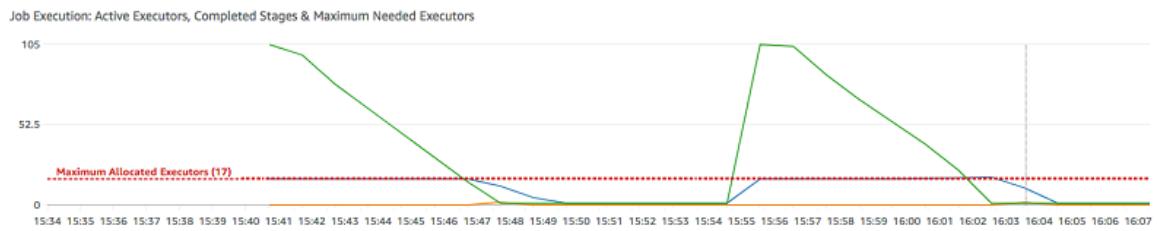
- The fraction of CPU system load used (scale: 0–1) by the driver, an executor identified by *executorId*, or all executors—
 - [glue.driver.system.cpuSystemLoad \(p. 344\)](#)
 - [glue.executorId.system.cpuSystemLoad \(p. 344\)](#)
 - [glue.ALL.system.cpuSystemLoad \(p. 344\)](#)



Example of Job Execution Graph

The Job Execution graph shows the following metrics:

- The number of actively running executors
—[glue.driver.ExecutorAllocationManager.executors.numberAllExecutors \(p. 338\)](#)
- The number of completed stages—[glue.aggregate.numCompletedStages \(p. 333\)](#)
- The number of maximum needed executors
—[glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors \(p. 339\)](#)



Restrictions for Jobs That Access Lake Formation Managed Tables

Keep in mind the following notes and restrictions when creating jobs that read from or write to tables managed by AWS Lake Formation:

- The following features are not supported in jobs that access tables with cell-level filters:
 - [Job bookmarks](#) and [bounded execution](#)
 - [Push-down predicates](#)
 - [Server-side catalog partition predicates](#)
 - [enableUpdateCatalog](#)

Using Auto Scaling for AWS Glue

Auto Scaling is available for your AWS Glue ETL and streaming jobs with AWS Glue version 3.0 or later.

Note

Auto Scaling is generally available for ETL jobs. Auto Scaling for streaming jobs is in preview and is subject to change.

With Auto Scaling enabled, you will get the following benefits:

- AWS Glue automatically adds and removes workers from the cluster depending on the parallelism at each stage or microbatch of the job run.
- It removes the need for you to experiment and decide on the number of workers to assign for your AWS Glue ETL jobs.
- If you choose the maximum number of workers, AWS Glue will choose the right size resources for the workload.
- You can see how the size of the cluster changes during the job run by looking at CloudWatch metrics on the job run details page in AWS Glue Studio.

Auto Scaling for AWS Glue ETL and streaming jobs enables on-demand scaling up and scaling down of the computing resources of your AWS Glue jobs. On-demand scale-up helps you to only allocate the required computing resources initially on job run startup, and also to provision the required resources as per demand during the job.

Auto Scaling also supports dynamic scale-down of the AWS Glue job resources over the course of a job. Over a job run, when more executors are requested by your Spark application, more workers will be added to the cluster. When the executor has been idle without active computation tasks, the executor and the corresponding worker will be removed.

Common scenarios where Auto Scaling helps with cost and utilization for your Spark applications include a Spark driver listing a large number of files in Amazon S3 or performing a load while executors are

inactive, Spark stages running with only a few executors due to overprovisioning, and data skews or uneven computation demand across Spark stages.

Requirements

Auto Scaling is only available for AWS Glue version 3.0. To use Auto Scaling, you can follow the [migration guide](#) to migrate your existing jobs to AWS Glue version 3.0 or create new jobs with AWS Glue version 3.0.

Auto Scaling is available for AWS Glue jobs with both G1.X and G2.X worker types. Standard DPUs are not supported.

Topics

- [Enabling Auto Scaling in AWS Glue Studio \(p. 222\)](#)
- [Enabling Auto Scaling with the AWS CLI or SDK \(p. 223\)](#)
- [Monitoring Auto Scaling with Amazon CloudWatch Metrics \(p. 224\)](#)
- [Monitoring Auto Scaling with Spark UI \(p. 224\)](#)
- [Monitoring Auto Scaling job run DPU usage \(p. 224\)](#)
- [Limitations \(p. 225\)](#)

Enabling Auto Scaling in AWS Glue Studio

On the **Job details** tab in AWS Glue Studio, choose the type as **Spark** or **Spark Streaming**, and **Glue version** as **Glue 3.0**. Then a check box will show up below **Worker type**.

- Select the **Automatically scale the number of workers** option.
- Set the **Maximum number of workers** to define the maximum number of workers that can be vended to the job run.

Visual | Script | **Job details** | Runs | Schedules

Type
The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

Glue version [Info](#)
Glue 3.0 - Supports spark 3.1, Scala 2, Python 3

Language
Python 3

Worker type
Set the type of predefined worker that is allowed when a job runs.

G.1X

Automatically scale the number of workers
AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run.
Requires Glue 3.0 or later.

Maximum number of workers
The number of workers you want AWS Glue to allocate to this job.

10

The maximums are 299 for G.1X and 149 for G.2X, and the minimum is 2.

Enabling Auto Scaling with the AWS CLI or SDK

To enable Auto Scaling From the AWS CLI for your job run, run `start-job-run` with the following configuration:

```
{  
    "JobName": "<your job name>",  
    "Arguments": {  
        "--enable-auto-scaling": "true"  
    },  
    "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs  
    "NumberOfWorkers": 20, // represents Maximum number of workers  
    ...other job run configurations...  
}
```

Once at ETL job run is finished, you can also call `get-job-run` to check the actual resource usage of the job run in DPU-seconds. Note: the new field **DPUSeconds** will only show up for your batch jobs on AWS Glue 3.0 enabled with Auto Scaling. This field is not supported for streaming jobs.

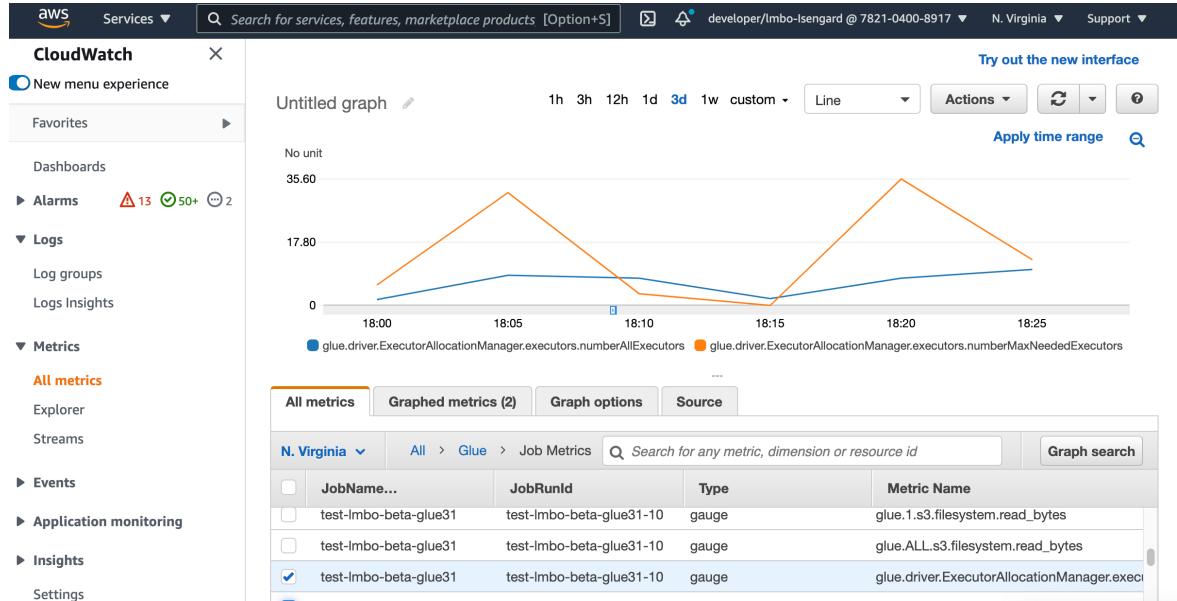
```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://glue.us-east-1.amazonaws.com --region us-east-1  
{  
    "JobRun": {  
        ...  
        "GlueVersion": "3.0",  
        "DPUSeconds": 386.0  
    }  
}
```

You can also configure job runs with Auto Scaling using the [AWS Glue SDK](#) with the same configuration.

Monitoring Auto Scaling with Amazon CloudWatch Metrics

The CloudWatch executor metrics are available for your AWS Glue 3.0 jobs if you enable Auto Scaling. The metrics can be used to monitor the demand and optimized usage of executors in their Spark applications enabled with Auto Scaling. For more information, see [Monitoring AWS Glue Using Amazon CloudWatch Metrics \(p. 331\)](#).

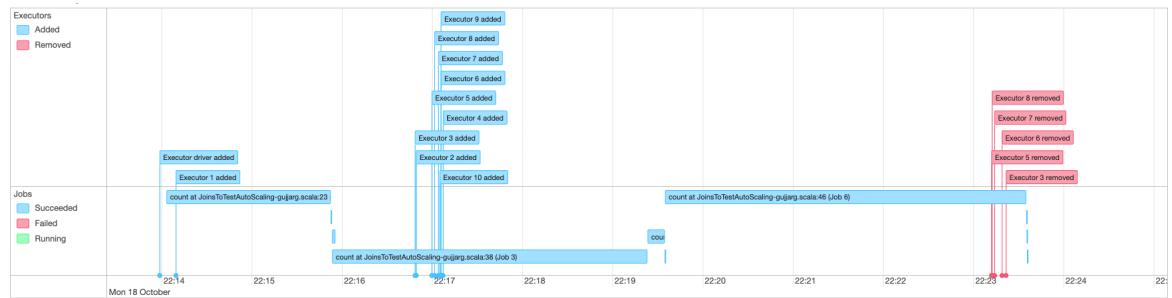
- `glue.driver.ExecutorAllocationManager.executors.numberAllExecutors`
- `glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors`



For more details on these metrics, see [Monitoring for DPU Capacity Planning \(p. 365\)](#).

Monitoring Auto Scaling with Spark UI

With Auto Scaling enabled, you can also monitor executors being added and removed with dynamic scale-up and scale-down based on the demand in your AWS Glue jobs using the Glue Spark UI. For more information, see [Enabling the Apache Spark Web UI for AWS Glue Jobs \(p. 319\)](#).



Monitoring Auto Scaling job run DPU usage

You may use the [AWS Glue Studio Job run view](#) to check the DPU usage of your Auto Scaling jobs.

1. Choose **Monitoring** from the AWS Glue Studio navigation pane. The Monitoring page appears.

2. Scroll down to the Job runs chart.
3. Navigate to the job run you are interested and scroll to the DPU hours column to check the usage for the specific job run.

Limitations

AWS Glue streaming Auto Scaling currently doesn't support a streaming DataFrame join with a static DataFrame created outside of `ForEachBatch`. A static DataFrame created inside the `ForEachBatch` will work as expected.

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 \(p. 227\)](#).

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 [Starting Jobs and Crawlers Using Triggers \(p. 296\)](#).

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	
<code>@params</code>	Parameters from the ETL job that the script requires.	

Annotation	Usage	
@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 \(p. 533\)](#).

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 \(p. 228\)](#).

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 \(p. 533\)](#).

- 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 \(p. 198\)](#).

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 \(p. 56\)](#).
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 \(p. 197\)](#).

For step-by-step guidance, see the **Add job** tutorial in the AWS Glue console.

Working with MongoDB Connections in ETL Jobs

You can create a connection for MongoDB and then use that connection in your AWS Glue job. The connection `url`, `username` and `password` are stored in the MongoDB connection. Other options can be specified in your ETL job script using the `additionalOptions` parameter of `glueContext.getCatalogSource`. The other options can include:

- `database`: (Required) The MongoDB database to read from.
- `collection`: (Required) The MongoDB collection to read from.

By placing the `database` and `collection` information inside the ETL job script, you can use the same connection for in multiple jobs.

1. Create an AWS Glue Data Catalog connection for the MongoDB data source. See "[connectionType": "mongodb](#)" for a description of the connection parameters. You can create the connection using the console, APIs or CLI.
2. Create a database in the AWS Glue Data Catalog to store the table definitions for your MongoDB data. See [Defining a Database in Your Data Catalog \(p. 101\)](#) for more information.
3. Create a crawler that crawls the data in the MongoDB using the information in the connection to connect to the MongoDB. The crawler creates the tables in the AWS Glue Data Catalog that describe the tables in the MongoDB database that you use in your job. See [Defining Crawlers \(p. 125\)](#) for more information.
4. Create a job with a custom script. You can create the job using the console, APIs or CLI. For more information, see [Adding Jobs in AWS Glue](#).
5. Choose the data targets for 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.
6. Customize the job-processing environment by providing arguments for your job and generated script.

Here is an example of creating a `DynamicFrame` from the MongoDB database based on the table structure defined in the Data Catalog. The code uses `additionalOptions` to provide the additional data source information:

Scala

```
val resultFrame: DynamicFrame = glueContext.getCatalogSource(  
    database = catalogDB,  
    tableName = catalogTable,  
    additionalOptions = JsonOptions(Map("database" -> DATABASE_NAME,  
        "collection" -> COLLECTION_NAME))
```

```
    ).getDynamicFrame()
```

Python

```
glue_context.create_dynamic_frame_from_catalog(  
    database = catalogDB,  
    table_name = catalogTable,  
    additional_options = {"database": "database_name",  
    "collection": "collection_name"})
```

7. Run the job, either on-demand or through a trigger.

Interactive applications and script development

Data engineers can author AWS Glue jobs faster and more easily than before using interactive sessions in AWS Glue.

Topics

- [Overview of AWS Glue Interactive Sessions \(p. 230\)](#)
- [Getting started with AWS Glue interactive sessions \(p. 231\)](#)
- [Configuring AWS Glue Interactive Sessions for Jupyter and AWS Glue Studio notebooks \(p. 241\)](#)
- [Interactive Sessions with IAM \(p. 245\)](#)
- [Converting a script or notebook into a Glue job \(p. 249\)](#)
- [AWS Glue Interactive Sessions for streaming \(p. 250\)](#)

Overview of AWS Glue Interactive Sessions

With AWS Glue interactive sessions, you can rapidly build, test, and run data preparation and analytics applications. Interactive Sessions provides a programmatic and visual interface for building and testing extract, transform, and load (ETL) scripts for data preparation. Interactive sessions run Apache Spark analytics applications and provide on-demand access to a remote Spark runtime environment. AWS Glue transparently manages serverless Spark for these interactive sessions.

Unlike AWS Glue development endpoints, AWS Glue interactive sessions are serverless with no infrastructure to manage. You can start interactive sessions very quickly. Interactive sessions have a 1-minute billing minimum with cost-control features. This reduces the cost of developing data preparation applications.

Because interactive sessions are flexible, you can build and test applications from the environment of your choice. You can create and work with interactive sessions through the AWS Command Line Interface and the API. You can use Jupyter-compatible notebooks to visually author and test your notebook scripts. Interactive sessions provide an open-source Jupyter kernel that integrates almost anywhere that Jupyter does, including integrating with IDEs such as PyCharm, IntelliJ, and VS Code. This enables you to author code in your local environment and run it seamlessly on the interactive sessions backend.

Using the Interactive Sessions API, customers can programmatically run applications that use Apache Spark analytics without having to manage Spark infrastructure. You can run one or more Spark statements within a single interactive session.

Interactive sessions therefore provide a faster, cheaper, more-flexible way to build and run data preparation and analytics applications. To learn how to use interactive sessions, see the documentation in this section.

Getting started with AWS Glue interactive sessions

These sections describe how to run AWS Glue interactive sessions locally.

Prerequisites for setting up Interactive Sessions locally

The following are prerequisites for installing interactive sessions:

- Python 3.6 or later
- See sections below for MacOS/Linux and Windows instructions.

MacOS/Linux Instructions

Installing Jupyter and AWS Glue Interactive Sessions Jupyter Kernels

1. Install `jupyter` `boto3` and `aws-glue-sessions` with pip. Jupyter Lab is also compatible and can be installed instead.

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

2. The following commands use pip to identify the installation location for `aws-glue-sessions`. The associated botocore then installs the Jupyter kernels.

```
SITE_PACKAGES=$(pip3 show aws-glue-sessions | grep Location | awk '{print $2}')
jupyter kernelspec install $SITE_PACKAGES/aws_glue_interactive_sessions_kernel/
glue_pyspark
jupyter kernelspec install $SITE_PACKAGES/aws_glue_interactive_sessions_kernel/
glue_spark
```

Configuring session credentials and region

AWS Glue Interactive Sessions requires the same IAM permissions as AWS Glue Jobs and Dev Endpoints. Specify the role used with Interactive Sessions in one of two ways:

1. With the `%iam_role` and `%region` magics
2. With an additional line in `~/.aws/config`

Configuring a session role with magic

In the first cell, type `%iam_role <YourGlueServiceRole>` in the first cell executed.

Configuring a Session role with `~/.aws/config`

AWS Glue Service Role for Interactive Sessions can either be specified in the notebook itself or stored alongside the AWS CLI config. If you have a role you typically use with AWS Glue Jobs this will be that role. If you do not have a role you use for AWS Glue jobs, please follow this guide, [Setting up IAM permissions for AWS Glue](#), to set one up.

To set this role as the default role for Interactive Sessions:

1. With a text editor, open `~/.aws/credentials`.
2. Look for the profile you use for AWS Glue. If you don't use a profile, use the `[Default]` profile.
3. Add a line in the profile for the role you intend to use like
`glue_role_arn=<AWSGlueServiceRole>`.

4. [Optional]: If your profile does not have a default region set, I recommend adding one with `region=us-east-1`, replacing `us-east-1` with your desired region.
5. Save the config.

For more information, see [Interactive Sessions with IAM \(p. 245\)](#).

Running Jupyter Notebook

To run Jupyter notebook, complete the following steps.

1. Run the following command to launch Jupyter Notebook.

```
jupyter notebook
```

2. Choose **New**, and then choose one of the AWS Glue kernels to begin coding against AWS Glue.

Windows Instructions

Installing Jupyter and AWS Glue Interactive Sessions kernels

1. Use `pip` to install Jupyter. Jupyter Lab is also compatible and can be installed instead.

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

2. (Optional) Run the following command to list the installed packages. If `jupyter` and `aws-glue-sessions` were successfully installed, you should see a long list of packages, including `jupyter 1.0.0` (or later).

```
pip3 list
```

3. Install the sessions kernels into Jupyter by running the following commands. These commands will look up the installation location for `aws-glue-sessions` from pip and install the Jupyter kernels therein.

- a. Change the directory to the `aws-glue-sessions` install directory within python's `site-packages` directory.

```
pip3 show aws-glue-sessions | Select-String Location | ConvertFrom-String.p2
```

- b. Windows PowerShell:

```
cd <site-packages Location>\aws_glue_interactive_sessions_kernel\
```

- c. Install the AWS Glue PySpark and AWS Glue Scala kernels.

```
jupyter-kernelspec install glue_pyspark
```

```
jupyter-kernelspec install glue_spark
```

Configuring session credentials and region

AWS Glue Interactive Sessions requires the same IAM permissions as AWS Glue Jobs and Dev Endpoints. Specify the role used with Interactive Sessions in one of two ways:

1. With the `%iam_role` and `%region` magics
2. With an additional line in `~/.aws/config`

Configuring a session role with magic

In the first cell, type `%iam_role <YourGlueServiceRole>` in the first cell executed.

Configuring a Session role with `~/.aws/config`

AWS Glue Service Role for Interactive Sessions can either be specified in the notebook itself or stored alongside the AWS CLI config. If you have a role you typically use with AWS Glue Jobs this will be that role. If you do not have a role you use for AWS Glue jobs, please follow this guide, [Setting up IAM permissions for AWS Glue](#), to set one up.

To set this role as the default role for Interactive Sessions:

1. With a text editor, open `~/.aws/credentials`.
2. Look for the profile you use for AWS Glue. If you don't use a profile, use the `[Default]` profile.
3. Add a line in the profile for the role you intend to use like
`glue_role_arn=<AWSGlueServiceRole>`.
4. [Optional]: If your profile does not have a default region set, I recommend adding one with
`region=us-east-1`, replacing `us-east-1` with your desired region.
5. Save the config.

For more information, see [Interactive Sessions with IAM \(p. 245\)](#).

Running Jupyter

To run Jupyter Notebook, complete the following steps.

1. Run the following command to launch Jupyter Notebook.

```
jupyter notebook
```

2. Choose **New**, and then choose one of the AWS Glue kernels to begin coding against AWS Glue.

Upgrading from the Interactive Sessions preview

The kernel was upgraded with new names when it was released with version 0.27. To clean up preview versions of the kernels run the following from a terminal or PowerShell.

Note

If you are a part of any other AWS Glue preview that requires a custom service model, this will remove it.

```
# Remove Old Glue Kernels
jupyter kernelspec remove glue_python_kernel
jupyter kernelspec remove glue_scala_kernel

# Remove Custom Model
cd ~/.aws/models
rm -rf glue/
```

Using Interactive Sessions with AWS Glue Studio notebook

See [Getting started with notebooks in AWS Glue Studio](#).

Using Interactive Sessions with SageMaker Notebooks

To run use Interactive Sessions with SageMaker Notebooks follow the steps to create a new lifecycle configuration and SageMaker Notebook instance.

Installing on an existing SageMaker Notebook instance

Note

Installing Interactive Sessions on an existing SageMaker Notebook instance is not currently supported. Although lifecycle configuration can be run on an existing SageMaker Notebook instance, installing Interactive Sessions on an existing SageMaker Notebook instance will modify the kernelspec manager class which may cause issues with other workloads.

Create a new lifecycle configuration and SageMaker notebook instance with Interactive Sessions

SageMaker Console setup

1. Create the Lifecycle Configuration.
 - a. Open the [Lifecycle configurations](#) page on the SageMaker console.

▼ Notebook

Notebook instances

Lifecycle configurations

Git repositories

- b. Choose **Create Configuration**.
- c. In the **Name** field, type 'AWSGlueInteractiveSessionsPreview'.

The screenshot shows the 'Create lifecycle configuration' page in the AWS SageMaker console. In the 'Scripts' section, the 'Create notebook' tab is selected. The code in the text area is:

```

1 #!/bin/bash
2 set -ex
3 sudo -u ec2-user -i <<'EOF'
4
5 ANACONDA_DIR=/home/ec2-user/anaconda3
6
7 # Create & Activate Conda Env
8 echo "Creating glue_pyspark conda enviornment"
9 conda create --name glue_pyspark python=3.7 ipykernel jupyter nb_conda -y
10
11 echo "Activating glue_pyspark"
12 source activate glue_pyspark
13
14 # Install Glue Sessions to Env
15 echo "Installing AWS Glue Sessions with pip"
16 pip install aws-glue-sessions
17
18 # Add Service Model to botocore. This will be removed with next CM.
19 echo "Copying Service Model to botocore"
20 cp ${ANACONDA_DIR}/envs/glue_pyspark/lib/python3.7/site-packages/aws_glue_interactive_sessions_ker
21
22 # Clone glue_pyspark to glue_scala. This is required because I had to match kernel naming conventi
23 echo "Cloning glue_pyspark to glue_scala"
24 conda create --name glue_scala --clone glue_pyspark
25

```

At the bottom right of the page are 'Cancel' and 'Create configuration' buttons.

- d. In the **Scripts** section, choose the **Create notebook** tab. Copy and paste the following into the text box and choose **Create configuration**.

```

#!/bin/bash
set -ex
sudo -u ec2-user -i <<'EOF'

ANACONDA_DIR=/home/ec2-user/anaconda3

# Create and Activate Conda Env
echo "Creating glue_pyspark conda enviornment"
conda create --name glue_pyspark python=3.7 ipykernel jupyter nb_conda -y

echo "Activating glue_pyspark"
source activate glue_pyspark

# Install Glue Sessions to Env
echo "Installing AWS Glue Sessions with pip"
pip install aws-glue-sessions

```

```

# Add Service Model to botocore. This will be removed with next CM.
echo "Copying Service Model to botocore"
cp ${ANACONDA_DIR}/envs/glue_pyspark/lib/python3.7/site-packages/
aws_glue_interactive_sessions_kernel/service-2.json ${ANACONDA_DIR}/envs/
glue_pyspark/lib/python3.7/site-packages/botocore/data/glue/2017-03-31/
service-2.json

# Clone glue_pyspark to glue_scala. This is required because I had to match kernel
# naming conventions to their environments and couldn't have two kernels in one
# conda env.
echo "Cloning glue_pyspark to glue_scala"
conda create --name glue_scala --clone glue_pyspark

# Remove python3 kernel from glue_pyspark
rm -r ${ANACONDA_DIR}/envs/glue_pyspark/share/jupyter/kernels/python3
rm -r ${ANACONDA_DIR}/envs/glue_scala/share/jupyter/kernels/python3

# Copy kernels to Jupyter kernel env (Discoverable by conda_nb_kernel)
echo "Copying Glue PySpark Kernel"
cp -r ${ANACONDA_DIR}/envs/glue_pyspark/lib/python3.7/site-packages/
aws_glue_interactive_sessions_kernel/glue_python_kernel/ ${ANACONDA_DIR}/envs/
glue_pyspark/share/jupyter/kernels/glue_python_kernel/

echo "Copying Glue Scala Kernel"
mkdir ${ANACONDA_DIR}/envs/glue_scala/share/jupyter/kernels
cp -r ${ANACONDA_DIR}/envs/glue_scala/lib/python3.7/site-packages/
aws_glue_interactive_sessions_kernel/glue_scala_kernel/ ${ANACONDA_DIR}/envs/
glue_scala/share/jupyter/kernels/glue_scala_kernel/

echo "Changing Jupyter kernel manager from EnvironmentKernelSpecManager to
      CondaKernelSpecManager"
JUPYTER_CONFIG=/home/ec2-user/.jupyter/jupyter_notebook_config.py

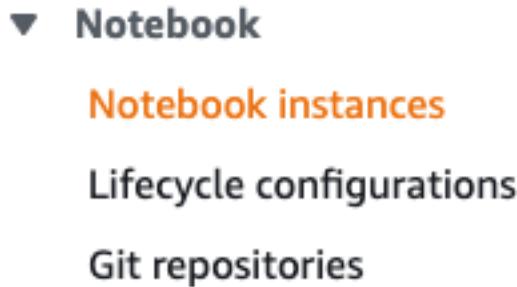
sed -i '/EnvironmentKernelSpecManager/ s/^/#/' ${JUPYTER_CONFIG}
echo "c.CondaKernelSpecManager.name_format='conda_{environment}'" >>
${JUPYTER_CONFIG}
echo "c.CondaKernelSpecManager.env_filter='anaconda3$|JupyterSystemEnv$|/R$'" >>
${JUPYTER_CONFIG}

EOF
restart jupyter-server

```

2. Create SageMaker Notebook instance with new lifecycle configuration.

- a. Choose **Notebook Instances**



- b. Choose **Create notebook instance**.
- c. Enter a name in the 'Notebook instance name' field.

The screenshot shows the 'Create notebook instance' configuration page. It includes sections for 'Notebook instance settings' (with fields for name, type, and identifier), 'Additional configuration' (with dropdowns for lifecycle configuration and volume size), and 'Permissions and encryption' (with IAM role selection and root access options). The 'Permissions and encryption' section is expanded.

Notebook instance settings

- Notebook instance name: My Glue Notebook
- Notebook instance type: ml.t2.medium
- Elastic Inference: none
- Platform identifier: notebook-al2-v1

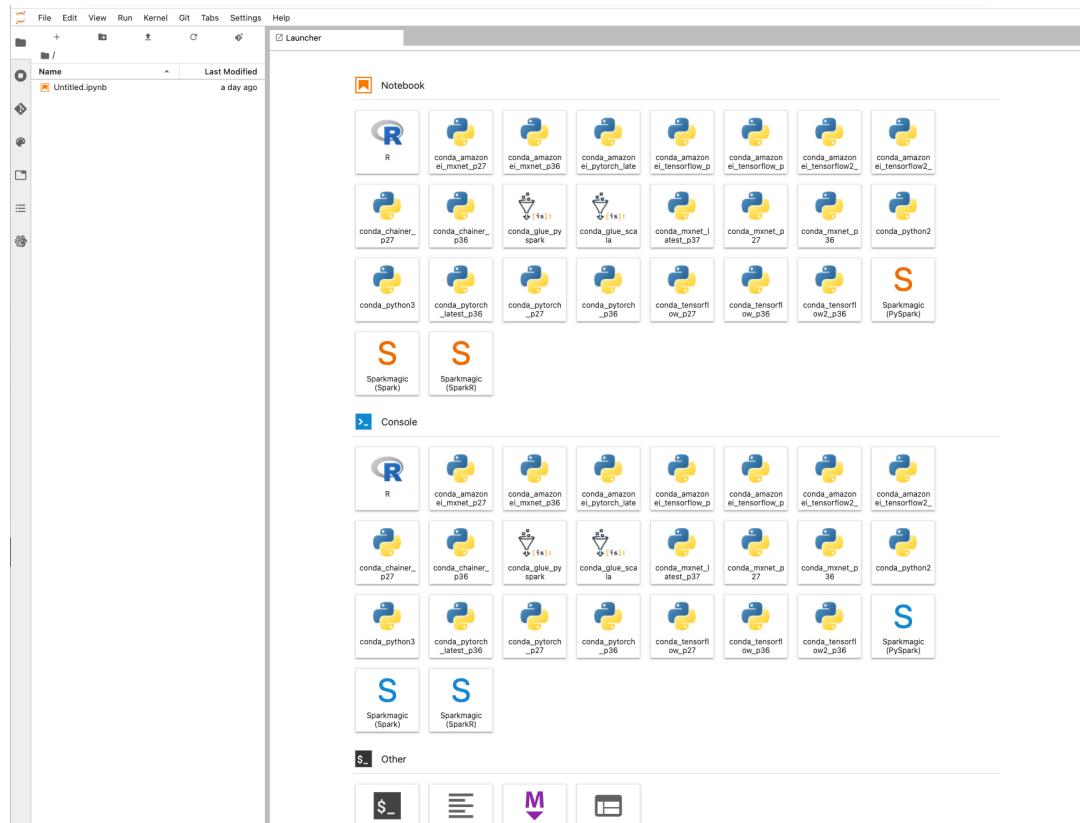
Additional configuration

- Lifecycle configuration - optional: AWSGlueInteractiveSessionsPreview
- Volume size in GB - optional: 5

Permissions and encryption

- IAM role: AWSGlueServiceSageMakerNotebookRole-viz_test
- Root access - optional:
 - Enable - Give users root access to the notebook (selected)
 - Disable - Don't give users root access to the notebook

- d. Choose 'notebook-al2-v1' as the **Platform identifier**.
 - e. Select **Additional configuration** to expose the **Lifecycle configuration - optional** drop-down menu.
 - f. Select **AWS GlueInteractiveSessionsPreview**.
 - g. In the **Permissions and encryption** section, choose an IAM role that has the permissions needed to run AWS Glue Interactive Sessions. For more information, see [Securing AWS Glue Interactive Sessions with IAM](#)
 - h. Complete selection of additional options as needed. When done, choose **Create notebook instance**.
3. Start an Interactive Session on your SageMaker Notebook instance.
- a. After the notebook has been provisioned, choose **Jupyter Lab** on the Notebook instances page.
 - b. Select the **conda_glue_pyspark** or **conda_glue_scala** icons in the **Notebook** section to create a new notebook.



4. Configure your AWS Glue session by running magics in the cells before your code. To find all available magics, run `%help` in your first cell. Run `%iam_role` if you have not configured it in `~/.aws/configure`.
5. For more information on configuring and using Interactive Sessions, see [Configuring AWS Glue Interactive Sessions](#).

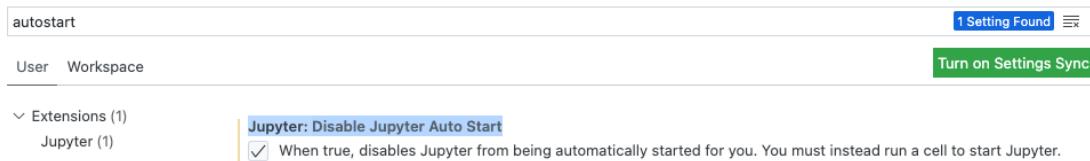
Using Interactive Sessions with Microsoft Visual Studio Code

Pre-requisites

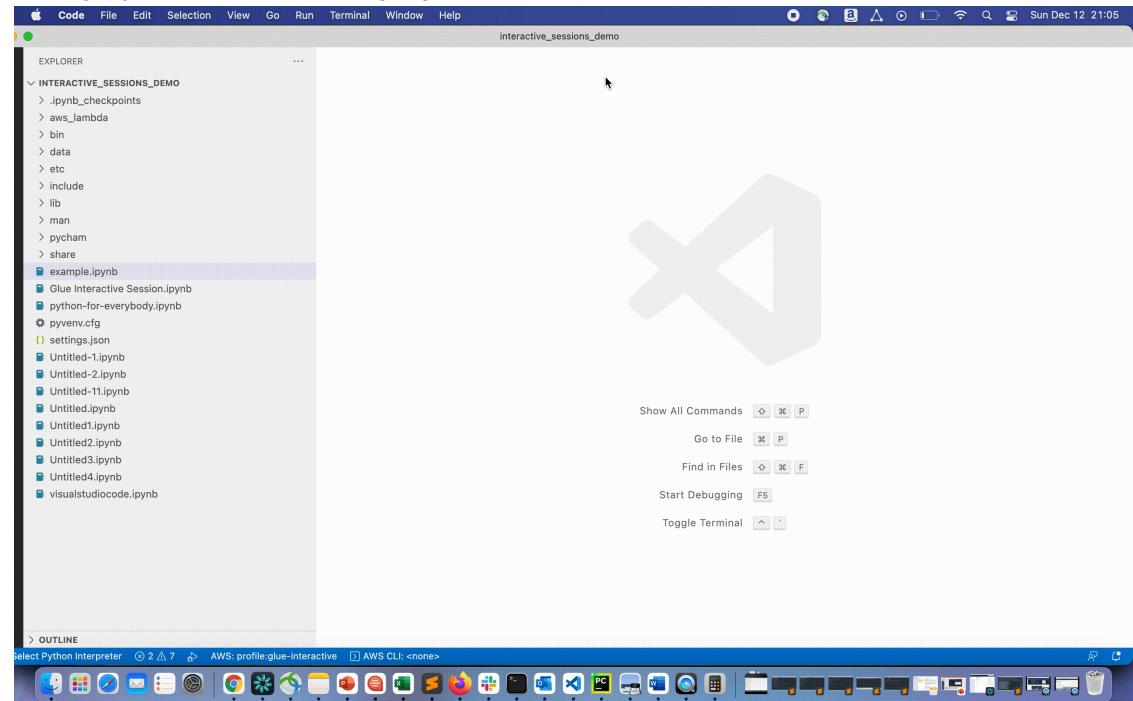
- Install AWS Glue Interactive Sessions and verify it works with Jupyter Notebook.
- Download and install Visual Studio Code with Jupyter. For details, see [Jupyter Notebook in VS Code](#).

1. Disable Jupyter AutoStart in VSCode.

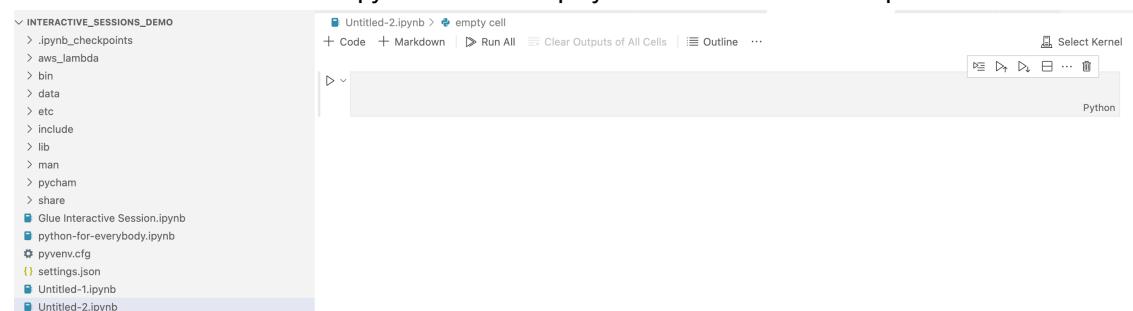
In Visual Studio Code Jupyter kernels will auto-start which will prevent your magics from taking effect as the session will already be started. To disable autostart Go to Code > Preferences > Settings > Extensions > Jupyter > Jupyter: Disable Jupyter Auto Start. Check the box labeled "When true, disables Jupyter from being automatically started for you. You must instead run a cell to start Jupyter."



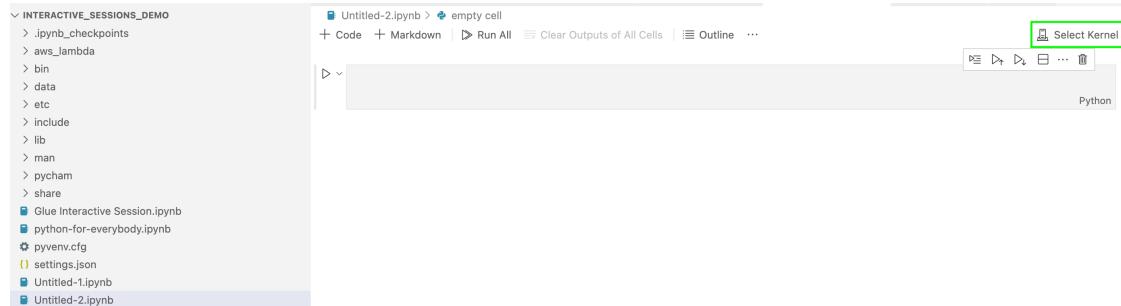
2. Go to File > New File > Save to save this file with name of your choice as an .ipynb extension or select **jupyter** under **select a language** and save the file.



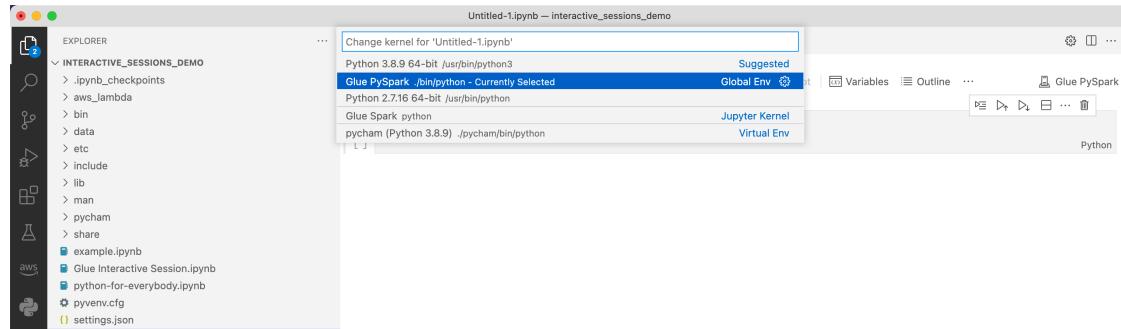
3. Double-click on the file. The Jupyter shell will display and a notebook will be opened.



4. When you first create a file, by default it has no kernel selected. Click on **Select Kernel** and a list of available kernels is displayed. Choose **Glue PySpark**.

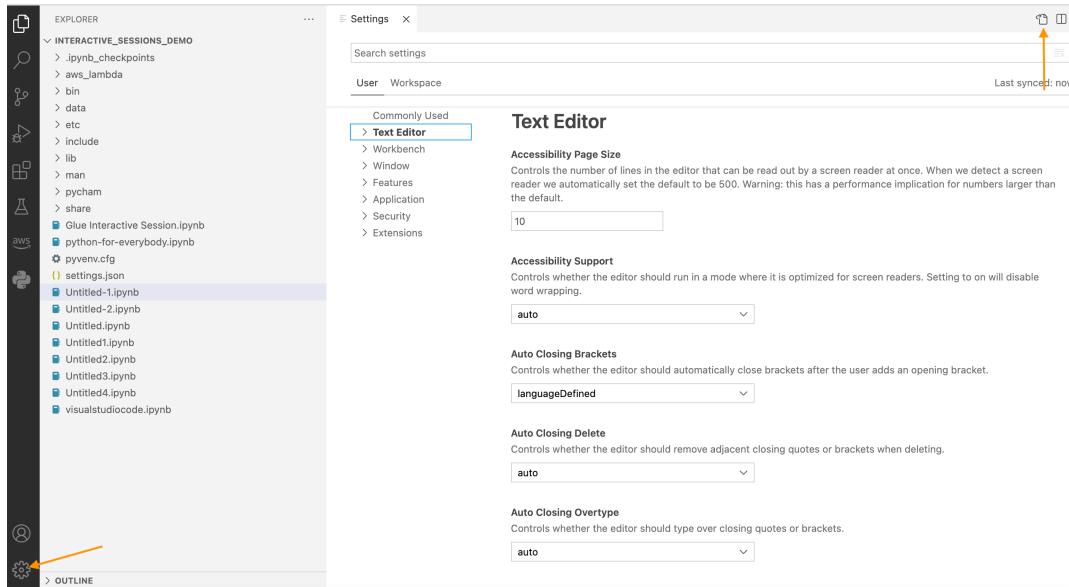


Choose the **Glue PySpark** or **Glue Spark** kernel (for Python and Scala respectively).



If you don't see **AWS Glue PySpark** and **AWS Glue Spark** kernels in the drop-down list, please ensure you have installed the AWS Glue kernel in the step above, or that your **python.pythonPath** in setting Visual Studio Code is correct. To validate the **python.pythonPath** see the steps below.

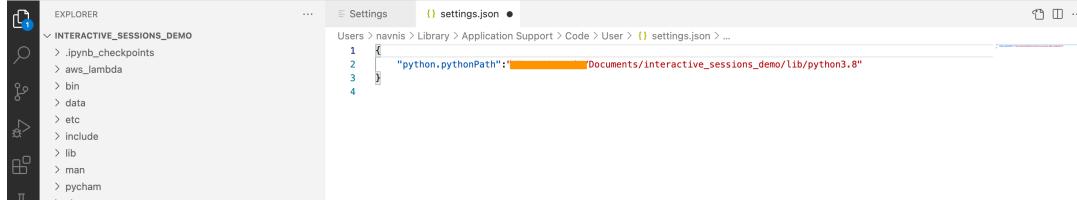
- In MS VS Code, navigate to Manage > Settings. Click on the open settings icon.



- python.pythonPath** should be pointing to your Python environment `python >location path<`. If you opened Visual Studio Code from a virtual environment with code `.,` it will point to the `pythonPath` of that virtual environment.
- If you don't see **python.pythonPath** please add it and restart Visual Studio Code. See examples below.

```
{  
    "python.pythonPath": "Python path of Python Virual environment"  
}
```

```
{  
    "python.pythonPath": "/Users/username/Documents/interactive_sessions_demo/lib/  
    python3.8"  
}
```



5. Create an AWS Glue Interactive Session. Proceed to create a session in the same manner as you did in Jupyter Notebook. Specify any magics at the top of your first cell and run a statement of code.

Configuring AWS Glue Interactive Sessions for Jupyter and AWS Glue Studio notebooks

Introduction to Jupyter Magics

Jupyter Magics are commands that can be run at the beginning of a cell or as a whole cell body. Magics start with % for line-magics and %% for cell-magics. Line-magics such as %region and %connections can be run with multiple magics in a cell, or with code included in the cell body like the following example.

```
%region us-east-2  
%connections my_rds_connection  
dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables',  
    table_name='sales_table')
```

Cell magics must use the entire cell and can have the command span multiple lines. An example of %sql is below.

```
%%sql  
select * from rds_tables.sales_table
```

Magics supported by AWS Glue Interactive Sessions for Jupyter

The following are magics that you can use with AWS Glue interactive sessions for Jupyter notebooks.

Sessions Magics

Name	Type	Description
%help	n/a	Return a list of descriptions and input types for all magic commands.
%profile	String	Specify a profile in your AWS configuration to use as the credentials provider.
%region	String	Specify the AWS Region; in which to initialize a session. Default from <code>~/.aws/configure</code> .
%idle_timeout	Int	The number of minutes of inactivity after which a session will timeout. The default idle timeout value is 2880 minutes (48 hours).
%session_id	String	Return the session ID for the running session. If a String is provided, this will be set as the session ID for the next running session.
%session_id_prefix	String	Define a string that will precede all session IDs in the format [session_id_prefix]-[session_id] . If a session ID is not provided, a random UUID will be generated.
%status		Return the status of the current AWS Glue session including its duration, configuration and executing user / role.
%stop_session		Stop the current session.
%list_sessions		Lists all currently running sessions by name and ID.
%stop_session		Stops the current session.
%stop_session	String	The version of Glue to be used by this session. Currently, the only valid options are 2.0 and 3.0. The default value is 2.0.
%streaming	String	Changes the session type to AWS Glue Streaming.
%etl	String	Changes the session type to AWS Glue ETL.

Glue Config Magics

Name	Type	Description
%%configure	Dictionary	Specify a JSON-formatted dictionary consisting of all configuration parameters for a session. Each parameter can be specified here or through individual magics.
%iam_role	String	Specify an IAM role ARN to execute your session with. Default from <code>~/.aws/configure</code>

Name	Type	Description
%number_of_workers	int	The number of workers of a defined worker_type that are allocated when a job runs. worker_type must be set too. The default number_of_workers is 5.
%worker_type	String	Standard, G.1X, or G.2X. number_of_workers must be set too. The default worker_type is G.1X.
%security_config	String	Define a Security Configuration to be used with this session.
%connections	List	Specify a comma-separated list of connections to use in the session.
%additional_python_modules	List	Comma separated list of additional Python modules to include in your cluster (can be from Pypi or S3).
%extra_py_files	List	Comma separated list of additional Python files from Amazon S3.
%extra_jars	List	Comma-separated list of additional jars to include in the cluster.

Action Magics

Name	Type	Description
%%sql	String	Run SQL code. All lines after the initial %%sql magic will be passed as part of the SQL code.

Naming sessions

\$GLU; Interactive Sessions are AWS resources and require a name. Names should be unique for each session and may be restricted by your IAM Administrators. For more information, see [Interactive Sessions with IAM \(p. 245\)](#). The Jupyter kernel automatically generates unique session names for you. However sessions can be named manually in two ways:

1. Using the AWS Command Line Interface config file located at `~/.aws/config`. See [Setting Up AWS Config with the AWS Command Line Interface](#).
2. Using the `%session_id_prefix` magics. See [Configuring AWS Glue Interactive Sessions for Jupyter and AWS Glue Studio notebooks \(p. 241\)](#).

A session name is generated as follows:

- When the prefix and session_id are provided: the session name will be `{prefix}-{UUID}`.
- When nothing is provided: the session name will be `{UUID}`.

Prefixing session names allows you to recognize your session when listing it in the AWS CLI or console.

Specifying an IAM role for Interactive Sessions

You must specify an AWS Identity and Access Management (IAM) role to use with AWS Glue ETL code that you run with interactive sessions.

The role requires the same IAM permissions as those required to run AWS Glue jobs. See [Create an IAM role for AWS Glue](#) for more information on creating a role for AWS Glue jobs and Interactive Sessions.

IAM roles can be specified in two ways:

- Using the AWS Command Line Interface config file located at `~/.aws/config` (Recommended). For more information, see [Configuring sessions with `~/.aws/config`](#).

Note

When the `%profile` magic is used, the configuration for `glue_iam_role` of that profile is honored.

- Using the `%iam_role` magic. For more information, see [Configuring AWS Glue Interactive Sessions for Jupyter and AWS Glue Studio notebooks \(p. 241\)](#).

Configuring sessions with Named Profiles

AWS Glue Interactive Sessions uses the same credentials as the AWS Command Line Interface or `boto3`. Interactive Sessions honors and works with Named profiles like the AWS CLI found in `~/ .aws/config` (Linux and MacOS) or `%USERPROFILE%\.aws\config` (Windows). For more information, see [Named profiles for the AWS Command Line Interface](#).

Interactive Sessions takes advantage of Named Profiles by allowing the AWS Glue Service Role and Session ID Prefix to be specified in a profile. To configure a profile role, add a line for the `iam_role` key and/or `session_id_prefix` to your named profile as shown below.

```
[default]
region=us-east-1
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXutnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRole>
session_id_prefix=<prefix_for_session_names>

[user1]
region=eu-west-1
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRoleUser1>
session_id_prefix=<prefix_for_session_names_for_user1>
```

If you have a custom method of generating credentials, you can also configure your profile to use the `credential_process` parameter in your `~/.aws/config` file. For example:

```
[profile developer]
region=us-east-1
credential_process = "/Users/Dave/generate_my_credentials.sh" --username helen
```

You can find more details about sourcing credentials through the `credential_process` parameter here: [Sourcing credentials with an external process](#).

If a region or `iam_role` are not set in the profile that you are using, you must specify them using the `%region` and `%iam_role` magics in the first cell that you run.

Interactive Sessions with IAM

These sections describe security considerations for AWS Glue Interactive Sessions.

Topics

- [IAM principals used with Interactive Sessions \(p. 245\)](#)
- [Setting up a client principal \(p. 245\)](#)
- [Setting up a runtime role \(p. 245\)](#)
- [Make your session private with TagOnCreate \(p. 246\)](#)
- [IAM policy considerations \(p. 249\)](#)

IAM principals used with Interactive Sessions

You use two IAM principals used with AWS Glue Interactive Sessions.

- **Client principal:** The client principal (either a user or a role) authorizes API operations for interactive sessions from an AWS Glue client that's configured with the principal's identity-based credentials. For example, this could be an IAM user whose credentials are used for the AWS Command Line Interface, or an AWS Glue client used by the interactive sessions Jupyter kernel. This could also be an IAM role principal that you typically use to access the AWS Glue console.
- **Runtime role:** The runtime role is an IAM role that the client principal passes to interactive sessions API operations. AWS Glue uses this role to run statements in your session. For example, this role could be the one used for running AWS Glue ETL jobs.

For more information, see [Setting up a runtime role \(p. 245\)](#).

Setting up a client principal

You must attach an identity policy to the client principal to allow it to call the Interactive Sessions API. This role must have `iam:PassRole` access to the execution role that you would pass to the Interactive Sessions API, such as `CreateSession`. For example, you can attach the **AWSGlueConsoleFullAccess** managed policy to an IAM user which allows all IAM users in your account with the same or similar policy attached to it to access all the sessions created in your account (such as runtime statement or cancel statement).

If you would like to protect your session and make it private only to the IAM user who created the session then you can use AWS Glue Interactive Session's Tag Based Authorization Control called TagOnCreate. For more information, see [Make your session private with TagOnCreate \(p. 246\)](#) on how an owner tag-based scoped down managed policy can make your session private with TagOnCreate. For more information on identity based policies, see [Using identity based policies](#).

Setting up a runtime role

You must pass an IAM role to the `CreateSession` API operation in order to allow AWS Glue to assume and run statements in interactive sessions. The role should have the same IAM permissions as those required to run a typical AWS Glue job. For example, you can create a service role using the **AWSGlueServiceRole** policy that allows AWS Glue to call AWS services on your behalf. If you use the AWS Glue console, it will automatically create a service role on your behalf or use an existing one. You can also create your own IAM role and attach your own IAM policy to allow similar permissions.

If you would like to protect your session and make it private only to the IAM user who created the session then you can use AWS Glue Interactive Session's Tag Based Authorization Control called TagOnCreate. For more information, see [Make your session private with TagOnCreate \(p. 246\)](#) on how an owner tag-based scoped down managed policy can make your session private with TagOnCreate.

For more information on identity based policies, see [Using identity based policies](#). If you are creating the execution role by yourself from the IAM console and you want to make your service private with TagOnCreate feature then follow the steps below.

1. Create an IAM role with role type set to Glue.
2. Attach this AWS Glue managed policy: `AwsGlueSessionUserRestrictedServiceRole`
3. Prefix the role name with the policy name `AwsGlueSessionUserRestrictedServiceRole`. For example, you can create a role with name `AwsGlueSessionUserRestrictedServiceRole-myrole` and attach AWS Glue managed policy `AwsGlueSessionUserRestrictedServiceRole`.
4. Attach a trust policy like following to allow AWS Glue to assume the role:

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

For an interactive sessions Jupyter kernel, you can specify the `iam_role` key in your AWS Command Line Interface profile. For more information, see [Configuring sessions with `~/.aws/configure`](#). If you're interacting with interactive sessions using an AWS Glue notebook, then you can pass the execution role in the `%iam_role` magic in the first cell that you run.

Make your session private with TagOnCreate

AWS Glue Interactive Sessions (IS) supports tagging and Tag Based Authorization (TBAC) for Interactive Sessions as a named resource. In addition to TBAC using TagResource and UntagResource APIs, AWS Glue Interactive Sessions supports the TagOnCreate feature to 'tag' a session with a given tag only during session creation with CreateSession operation. This also means those tags will be removed on DeleteSession, aka UntagOnDelete.

TagOnCreate offers a powerful security mechanism to make your session private to the creator of the session. For example, you can attach an IAM policy with "owner" RequestTag and value of `#{aws:userId}` to a client principal (such as an IAM user) in order to allow creating a session only if an "owner" tag with matching value of the callers userId is provided as userId tag in CreateSession request. This policy allows AWS Glue Interactive Sessions to create a session resource and tag the session with the userId tag only during session creation time. In addition to it you can scope down the access (like running statements) to your session only to the creator (aka owner tag with value `#{aws:userId}`) of the session by attaching an IAM policy with "owner" ResourceTag to the execution role you passed in during CreateSession.

In order to make it easier for you to use TagOnCreate feature to make a session private to the session creator, AWS Glue provides specialized managed policies and service roles. For example, you can attach the **AWSGlueSessionUserRestrictedPolicy** to each of the IAM users in your account to restrict them creating a session only with a owner tag with a value matching their own `#{aws:userId}`. For more information, see [Using identity based policies](#). This policy scopes down the access to a session only to the creator aka the `#{aws:userId}` of the IAM user who created the session with owner tag baring their own `#{aws:userId}`. If you have created the execution role yourself using IAM console by following the steps in Setting up a Runtime Role, then in addition to attaching `AwsGlueSessionUserRestrictedPolicy`

managed policy also attach the following inline policy to each of the IAM users in your account to allow `iam:PassRole` for the execution role you created earlier.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedServiceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

If you want to create a AWS Glue Interactive Session using an IAM AssumeRole principal (that is, using credential vended by assuming an IAM role) and you want to make the session private to the creator, then instead of using the **AWSGlueSessionUserRestrictedPolicy** and **AWSGlueSessionUserRestrictedServiceRole** you need to use policies similar to the **AWSGlueSessionUserRestrictedNotebookPolicy** and **AWSGlueSessionUserRestrictedNotebookServiceRole** respectively. These policies allow AWS Glue to use `${aws:PrincipalTag}` to extract the owner tag value. This requires you to pass a userId tag with value `${aws:userId}` as SessionTag in the assume role credential. See [ID session tags](#). If you are using an Amazon EC2 instance with an instance profile vending the credential and you want to create a session or interact with the session from within the Amazon EC2 instance , then you would require to pass a userId tag with value `${aws:userId}` as SessionTag in the assume role credential.

For example, If you are creating a session using an IAM AssumeRole principal credential and you want to make your service private with TagOnCreate feature then follow the steps below.

1. Create a runtime role yourself from the IAM console. Please attach this AWS Glue managed policy `AwsGlueSessionUserRestrictedNotebookServiceRole` and prefix the role name with the policy name `AwsGlueSessionUserRestrictedNotebookServiceRole`. For example, you can create a role with name `AwsGlueSessionUserRestrictedNotebookServiceRole-myrole` and attach AWS Glue managed policy `AwsGlueSessionUserRestrictedNotebookServiceRole`.
2. Attach a trust policy like below to allow AWS Glue to assume the above role.

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

}

3. Create another role named with a prefix `AwsGlueSessionUserRestrictedNotebookPolicy` and attach the AWS Glue managed policy `AwsGlueSessionUserRestrictedNotebookPolicy` to make the session private. In addition to the managed policy please attach the following inline policy to allow `iam:PassRole` to the role you created in step 1.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedNotebookServiceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        ]
    }
}
```

4. Attach a trust policy like following to the above IAM AWS Glue to assume the role.

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

Note

Optionally, you can use a single role (for example, notebook role) and attach both of the above managed policies `AwsGlueSessionUserRestrictedNotebookServiceRole` and `AwsGlueSessionUserRestrictedNotebookPolicy`. Also attach the additional inline policy to allow `iam:passrole` of your role to AWS Glue. And finally attach the above trust policy to allow `sts:AssumeRole` and `sts:TagSession`.

AWSGlueSessionUserRestrictedPolicy

The **AWSGlueSessionUserRestrictedPolicy** provides access to create a AWS Glue Interactive Session using the `CreateSession` API only if a tag key "owner" and value matching their AWS user ID is provided.

This identity policy is attached to the IAM user that invokes the CreateSession API. This policy also permits to interact with the AWS Glue Interactive Session resources that were created with a "owner" tag and value matching their AWS user id. This policy denies permission to change or remove "owner" tag from a AWS Glue session resource after the session is created.

AWSGlueSessionUserRestrictedServiceRole

The **AWSGlueSessionUserRestrictedServiceRole** provides full access to all AWS Glue resources except for sessions and allows users to create and use only the interactive sessions that are associated with the user. This policy also includes other permissions needed by AWS Glue to manage Glue resources in other AWS services. The policy also allows adding tags to AWS Glue resources in other AWS services.

AWSGlueSessionUserRestrictedNotebookPolicy

The **AWSGlueSessionUserRestrictedNotebookPolicy** provides access to create a AWS Glue Interactive Session from a notebook only if a tag key "owner" and value matching the AWS user id of the principal (IAM user or Role). For more information, see [Where you can use policy variables](#). This policy is attached to the principal (IAM User or role) that creates AWS Glue Interactive Session notebooks from AWS Glue Studio. This policy also permits sufficient access to the AWS Glue Studio notebook to interact with the AWS Glue Studio Interactive Session resources that are created with the "owner" tag value matching the AWS user ID of the principal. This policy denies permission to change or remove "owner" tag from a AWS Glue session resource after the session is created.

AWSGlueSessionUserRestrictedNotebookServiceRole

The **AWSGlueSessionUserRestrictedNotebookServiceRole** provides sufficient access to the AWS Glue Studio notebook to interact with the AWS Glue Interactive Session resources that are created with the "owner" tag value matching the AWS user ID of the principal (IAM user or role) of the notebook creator. For more information, see [Where you can use policy variables](#). This service-role policy is attached to the role that is passed as magic to a notebook or passed as execution role to the CreateSession API. This policy also permits to create a AWS Glue Interactive Session from a notebook only if a tag key "owner" and value matching the AWS user ID of the principal. This policy denies permission to change or remove "owner" tag from an AWS Glue session resource after the session is created. This policy also includes permissions for writing and reading from Amazon S3 buckets, writing CloudWatch logs, creating and deleting tags for Amazon EC2 resources used by AWS Glue.

IAM policy considerations

Interactive sessions are IAM resources in AWS Glue. Because they are IAM resources, access and interaction to a session is governed by IAM policies. Based on the IAM policies attached to a client principal or execution role configured by an admin, a client principal (user or role) will be able to create new sessions and interact with its own sessions and other sessions.

If an admin has attached an IAM policy such as [AWSGlueConsoleFullAccess](#) (<https://docs.aws.amazon.com/glue/latest/dg/using-identity-based-policies.html>) or [AWSGlueServiceRole](#) (<https://docs.aws.amazon.com/glue/latest/dg/using-identity-based-policies.html>) that allows access to all AWS Glue resources in that account, a client principal will be able to collaborate with each other. For example, one IAM user will be able to interact with sessions that are created by other IAM users if policies allow this.

If you'd like to configure a policy tailored to your specific needs, see [IAM documentation about configuring resources for a policy](#). For example, in order to isolate sessions that belong to an IAM user, in order to isolate sessions that belong to an IAM user, you can use the TagOnCreate feature supported by Glue IS. See [Make your session private with TagOnCreate \(p. 246\)](#).

Converting a script or notebook into a Glue job

There are two ways you can convert a script or notebook into a AWS Glue job:

- Use **nbconvert** to convert your Jupyter .ipynb notebook document file into a .py file. For more information, see [nbconvert: Convert Notebooks to other formats](#).
- Upload the file to AWS Glue Studio Notebooks.
 - In the AWS Glue Studio console, choose **Jobs** from the navigation menu.
 - In the **Create job** section, choose **Jupyter Notebook**.
 - In the **Options** section, choose **Upload and edit an existing notebook**.
 - Select **Choose file** to upload an .ipynb file.

AWS Glue Interactive Sessions for streaming

Switching streaming session type

Use the AWS Glue Interactive Sessions configuration magic, %streaming, to define the job you are running and initialize a streaming interactive session.

Sampling input stream for interactive development

One tool we have derived to help enhance the interactive experience in AWS Glue Interactive Sessions is the addition of a new method under `GlueContext` to obtain a snapshot of a stream in a static `DynamicFrame`. `GlueContext` allows you to inspect, interact and implement your workflow.

In an interactive session, a `GlueContext` is provided by default and with this class instance, you will be able to locate this method `getSampleStreamingDynamicFrame`. Required arguments for this methods are:

- `dataFrame`: The Spark Streaming Dataframe
- `options`: See available options below

Available options include :

- **windowSize**: This is also called Microbatch Duration. This parameter will determine how long a streaming query will wait after previous batch was triggered. This parameter value must be smaller than `pollingTimeInMs`.
- **pollingTimeInMs**: The total length of time the method will run. It will fire off at least one micro batch to obtain sample records from the input stream.
- **recordPollingLimit**: This parameter helps you limit the total number of records you will poll from the stream.

(Optional) You can also use `writeStreamFunction` to apply this custom function to every record sampling function. See below for examples in Scala and Python.

Scala

```
val jsonString: String = s""{"pollingTimeInMs": "2000", "windowSize": "1
seconds"}"""
val dynFrame = glueContext.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF,
  JsonOptions(jsonString))
dynFrame.show()
```

Python

```
options = {
    "pollingTimeInMs": "5000",
    "windowSize": "10 seconds"
}
glue_context.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF, options, None)
```

Note

When the sampled DynFrame is empty, it could be caused by a few reasons:

- The Streaming source is set to “Latest” and no new data has been ingested during the sampling period.
- The polling time is not enough to process the records it ingested. Data won’t show up unless the whole batch has been processed.

Running streaming applications in Interactive Sessions

In AWS Glue Interactive Sessions, you can run a the AWS Glue streaming application like how you would create a streaming application in the AWS Glue Console. Since Interactive Sessions is session-based, encountering exceptions in the runtime does not cause the session to stop. We now have the added benefit of developing your batch function iteratively. For example:

```
def batch_function(data_frame, batch_id):
    log.info(data_frame.count())
    invalid_method_call()
glueContext.forEachBatch(frame=streaming_df, batch_function = batch_function, options = {**})
```

In the example above, we included an invalid usage of a method and unlike regular AWS Glue jobs which will exit the entire application, the user’s coding context and definitions are fully preserved and the session is still operational. There is no need to bootstrap a new cluster and rerun all the preceding transformation. This allows you to focus on quickly iterating your batch function implementations to obtain desirable outcomes.

It is important to note that Interactive Session evaluates each statement in a blocking manner so that the session will only execute one statement at a time. Since streaming queries are continuous and never ending, sessions with active streaming queries won’t be able to handle any follow up statements unless they are interrupted. You can issue the interruption command directly from Jupyter Notebook and our kernel will handle the cancellation for you.

Take the following sequence of statements which are waiting for execution as an example:

```
Statement 1:
val number = df.count()
#Spark Action with deterministic result
Result: 5

Statement 2:
streamingQuery.start().awaitTermination()
#Spark Streaming Query that will be executing continuously
Result: Constantly updated with each microbatch

Statement 3:
val number2 = df.count()
```

#This will not be executed as previous statement will be running indefinitely

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.

Note

Development endpoints are not supported for use with AWS Glue version 2.0 jobs. For more information, see [Running Spark ETL Jobs with Reduced Startup Times](#).

Managing your Development Environment

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 \(p. 252\)](#)
- [How AWS Glue Development Endpoints Work with SageMaker Notebooks \(p. 253\)](#)
- [Adding a Development Endpoint \(p. 254\)](#)
- [Viewing Development Endpoint Properties \(p. 256\)](#)
- [Accessing Your Development Endpoint \(p. 257\)](#)
- [Creating a Notebook Server Hosted on Amazon EC2 \(p. 258\)](#)
- [Tutorial Setup: Prerequisites for the Development Endpoint Tutorials \(p. 260\)](#)
- [Tutorial: Set Up a Local Apache Zeppelin Notebook to Test and Debug ETL Scripts \(p. 264\)](#)
- [Tutorial: Set Up an Apache Zeppelin Notebook Server on Amazon EC2 \(p. 267\)](#)
- [Tutorial: Set Up a Jupyter Notebook in JupyterLab to Test and Debug ETL Scripts \(p. 269\)](#)
- [Tutorial: Use a SageMaker Notebook with Your Development Endpoint \(p. 272\)](#)
- [Tutorial: Use a REPL Shell with Your Development Endpoint \(p. 274\)](#)
- [Tutorial: Set Up PyCharm Professional with a Development Endpoint \(p. 275\)](#)
- [Advanced Configuration: Sharing Development Endpoints among Multiple Users \(p. 281\)](#)

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 \(p. 287\)](#).
 - Create an 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 \(p. 293\)](#).
 - 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.

How AWS Glue Development Endpoints Work with SageMaker Notebooks

One of the common ways to access your development endpoints is to use [Jupyter](#) on SageMaker notebooks. The Jupyter notebook is an open-source web application which is widely used in visualization, analytics, machine learning, etc. An AWS Glue SageMaker notebook provides you a Jupyter notebook experience with AWS Glue development endpoints. In the AWS Glue SageMaker notebook, the Jupyter notebook environment is pre-configured with [SparkMagic](#), an open source Jupyter plugin to submit Spark jobs to a remote Spark cluster. [Apache Livy](#) is a service that allows interaction with a remote Spark cluster over a REST API. In the AWS Glue SageMaker notebook, SparkMagic is configured to call the REST API against a Livy server running on an AWS Glue development endpoint.

The following text flow explains how each component works:

AWS Glue SageMaker notebook: (Jupyter # SparkMagic) # (network) # AWS Glue development endpoint: (Apache Livy # Apache Spark)

Once you run your Spark script written in each paragraph on a Jupyter notebook, the Spark code is submitted to the Livy server via SparkMagic, then a Spark job named "livy-session-N" runs on the Spark cluster. This job is called a Livy session. The Spark job will run while the notebook session is alive. The Spark job will be terminated when you shutdown the Jupyter kernel from the notebook, or when the session is timed out. One Spark job is launched per notebook (.ipynb) file.

You can use a single AWS Glue development endpoint with multiple SageMaker notebook instances. You can create multiple notebook files in each SageMaker notebook instance. When you open an each notebook file and run the paragraphs, then a Livy session is launched per notebook file on the Spark cluster via SparkMagic. Each Livy session corresponds to single Spark job.

Default Behavior for AWS Glue Development Endpoints and SageMaker Notebooks

The Spark jobs run based on the [Spark configuration](#). There are multiple ways to set the Spark configuration (for example, Spark cluster configuration, SparkMagic's configuration, etc.).

By default, Spark allocates cluster resources to a Livy session based on the Spark cluster configuration. In the AWS Glue development endpoints, the cluster configuration depends on the worker type. Here's a table which explains the common configurations per worker type.

	Standard	G.1X	G.2X
<code>spark.driver.memory</code>	5G	10G	20G
<code>spark.executorMemory</code>	5G	10G	20G
<code>spark.executorcores</code>	4cores	8	16
<code>spark.dynamicAllocation.enabled</code>	TRUE	TRUE	TRUE

The maximum number of Spark executors is automatically calculated by combination of DPU (or `NumberOfWorkers`) and worker type.

	Standard	G.1X	G.2X
The number of max Spark executors	$(DPU - 1) * 2 - 1$	$(NumberOfWorkers - 1)$	$(NumberOfWorkers - 1)$

For example, if your development endpoint has 10 workers and the worker type is G.1X, then you will have 9 Spark executors and the entire cluster will have 90G of executor memory since each executor will have 10G of memory.

Regardless of the specified worker type, Spark dynamic resource allocation will be turned on. If a dataset is large enough, Spark may allocate all the executors to a single Livy session since `spark.dynamicAllocation.maxExecutors` is not set by default. This means that other Livy sessions on the same dev endpoint will wait to launch new executors. If the dataset is small, Spark will be able to allocate executors to multiple Livy sessions at the same time.

Note

For more information about how resources are allocated in different use cases and how you set a configuration to modify the behavior, see [Advanced Configuration: Sharing Development Endpoints among Multiple Users \(p. 281\)](#).

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.

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.

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.

AWS 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 \(p. 198\)](#).

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 \(p. 310\)](#).

Spark UI

Turns on 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 \(p. 320\)](#).

Use AWS Glue Data Catalog as the Hive metastore (under Catalog Options)

Allows 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 \(p. 198\)](#).

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" \(p. 257\)](#).

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 \(p. 16\)](#).

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" \(p. 287\)](#).

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” \(p. 287\)](#).

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 \(p. 28\)](#).

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 \(p. 39\)](#).

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 AWS KMS encryption of the Zeppelin notebook storage on Amazon S3. This field is optional. Allow 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 Store (Amazon EBS) Amazon 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 \(p. 16\)](#). 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 \(p. 39\)](#).

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 \(p. 16\)](#). 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 retrieve the connection details from the existing connection to use in configuring the Amazon VPC, Subnet, and Security groups parameters 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 \(p. 31\)](#).
 - 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 \(p. 32\)](#).
 - 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:

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:

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 \(p. 39\)](#).

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 \(p. 264\)](#)
- [Tutorial: Set Up an Apache Zeppelin Notebook Server on Amazon EC2 \(p. 267\)](#)
- [Tutorial: Use a REPL Shell with Your Development Endpoint \(p. 274\)](#)

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 \(p. 260\)](#).

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.

AWS Glue Version	Zeppelin Release	File To Download
0.9	0.7.3	zeppelin-0.7.3-bin-all.tgz
1.0 and later	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+lwKzRko
+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 network 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, again turn on the **spark** interpreter's **Connect to existing process** settings, and then save again.

Resetting the interpreter like this should restore the network 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 \(p. 260\)](#).

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 \(p. 17\)](#).

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 \(p. 28\)](#).
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 \(p. 287\)](#). 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: Set Up a Jupyter Notebook in JupyterLab to Test and Debug ETL Scripts

In this tutorial, you connect a Jupyter notebook in JupyterLab running on your local machine to a development endpoint. You do this so that you can interactively run, debug, and test AWS Glue extract, transform, and load (ETL) scripts before deploying them. This tutorial uses Secure Shell (SSH) port forwarding to connect your local machine to an AWS Glue development endpoint. For more information, see [Port forwarding](#) on Wikipedia.

This tutorial assumes that you have already taken the steps outlined in [Tutorial Prerequisites \(p. 260\)](#).

Step 1: Install JupyterLab and Sparkmagic

You can install JupyterLab by using `conda` or `pip`. `conda` is an open-source package management system and environment management system that runs on Windows, macOS, and Linux. `pip` is the package installer for Python.

If you're installing on macOS, you must have Xcode installed before you can install Sparkmagic.

1. Install JupyterLab, Sparkmagic, and the related extensions.

```
$ conda install -c conda-forge jupyterlab
$ pip install sparkmagic
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
```

2. Check the `sparkmagic` directory from `Location`.

```
$ pip show sparkmagic | grep Location
```

```
Location: /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages
```

3. Change your directory to the one returned for Location, and install the kernels for Scala and PySpark.

```
$ cd /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages  
$ jupyter-kernelspec install sparkmagic/kernels/sparkkernel  
$ jupyter-kernelspec install sparkmagic/kernels/pysparkkernel
```

4. Download a sample config file.

```
$ curl -o ~/.sparkmagic/config.json https://raw.githubusercontent.com/jupyter-incubator/sparkmagic/master/sparkmagic/example_config.json
```

In this configuration file, you can configure Spark-related parameters like `driverMemory` and `executorCores`.

Step 2: Start JupyterLab

When you start JupyterLab, your default web browser is automatically opened, and the URL `http://localhost:8888/lab/workspaces/{workspace_name}` is shown.

```
$ jupyter lab
```

Step 3: Initiate SSH Port Forwarding to Connect to Your Development Endpoint

Next, use SSH local port forwarding to forward a local port (here, 8998) to the remote destination that is defined by AWS Glue (169.254.76.1:8998).

1. Open a separate terminal window that gives you access to SSH. In Microsoft Windows, you can use the BASH shell provided by [Git for Windows](#), or you can install [Cygwin](#).
2. 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're forwarding a different port than 8998, replace 8998 with the port number that you're actually using locally. The address 169.254.76.1:8998 is the remote port and isn't 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 on the **Endpoint details** page.

```
ssh -i private-key-file-path -NTL 8998:169.254.76.1:8998 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)?
```

Enter **yes** and leave the terminal window open while you use JupyterLab.

- Check that SSH port forwarding is working with the development endpoint correctly.

```
$ curl localhost:8998/sessions
{"from":0,"total":0,"sessions":[]}
```

Step 4: Run a Simple Script Fragment in a Notebook Paragraph

Now your notebook in JupyterLab should work with your development endpoint. Enter the following script fragment into your notebook and run it.

- Check that Spark is running successfully. The following command instructs Spark to calculate 1 and then print the value.

```
spark.sql("select 1").show()
```

- Check if AWS Glue Data Catalog integration is working. The following command lists the tables in the Data Catalog.

```
spark.sql("show tables").show()
```

- Check that a simple script fragment that uses AWS Glue libraries works.

The following script uses the `persons_json` table 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.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext

# 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

- During the installation of JupyterLab, if your computer is behind a corporate proxy or firewall, you might encounter HTTP and SSL errors due to custom security profiles managed by corporate IT departments.

The following is an example of a typical error that occurs when conda can't connect to its own repositories:

```
CondaHTTPError: HTTP 000 CONNECTION FAILED for url <https://repo.anaconda.com/pkgs/main/win-64/current_repodata.json>
```

This might happen because your company can block connections to widely used repositories in Python and JavaScript communities. For more information, see [Installation Problems](#) on the JupyterLab website.

- If you encounter a *connection refused* error when trying to connect to your development endpoint, you might be using a development endpoint that is out of date. Try creating a new development endpoint and reconnecting.

Tutorial: Use a SageMaker Notebook with Your Development Endpoint

In AWS Glue, you can create a development endpoint and then create a 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 \(p. 260\)](#).

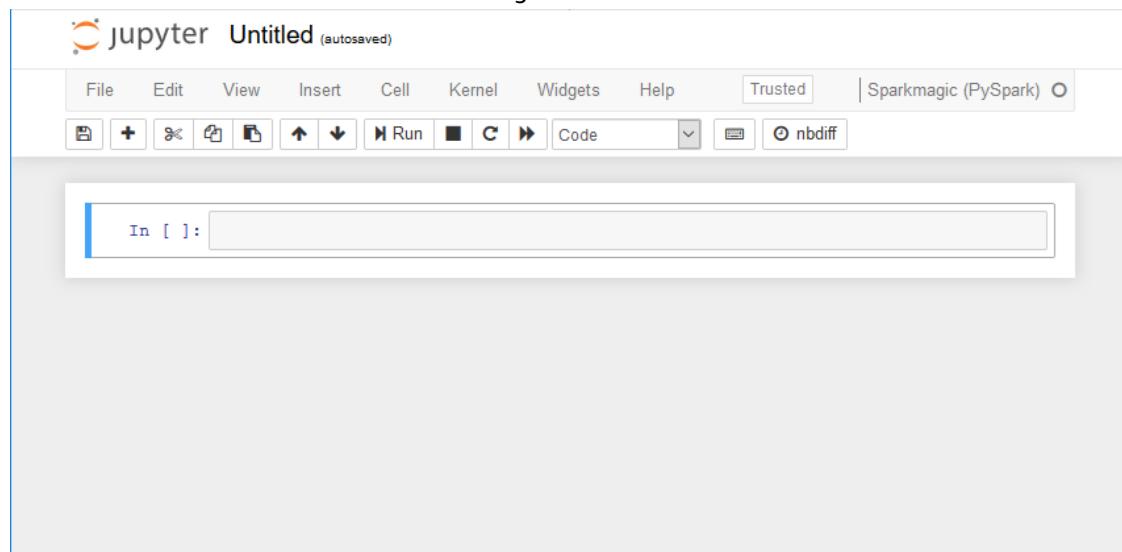
1. In the AWS Glue console, choose **Dev endpoints** to navigate to the development endpoints list.
2. Select the check box next to the name of a development endpoint that you want to use, and on the **Action** menu, choose **Create SageMaker notebook**.
3. Fill out the **Create and configure a notebook** 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 SageMaker Notebooks" \(p. 29\)](#).

- 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 \(p. 260\)](#).

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
```

```
/__/\_ \_\_ \_\_ /__/  
\_ \ \_ \_ \_ / \_ / \_ /  
/ \_ / . \_ / \_ , \_ / / \_ / \_ \_ /  
/ \_ /  
version 2.4.3
```

```
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 \(p. 260\)](#).

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 \(p. 275\)](#)
- [Deploying the Script to Your Development Endpoint \(p. 279\)](#)
- [Configuring a Remote Interpreter \(p. 280\)](#)
- [Running Your Script on the Development Endpoint \(p. 280\)](#)

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()
```

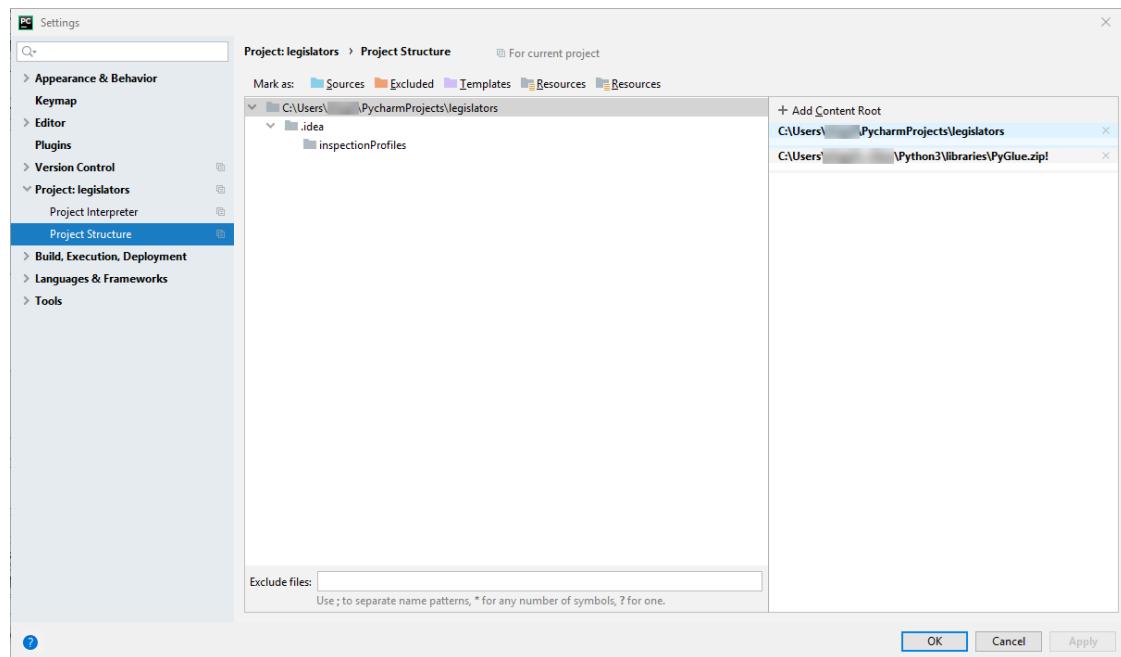
3. Do one of the following:

- For AWS 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 AWS Glue version 1.0 and later, 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**.

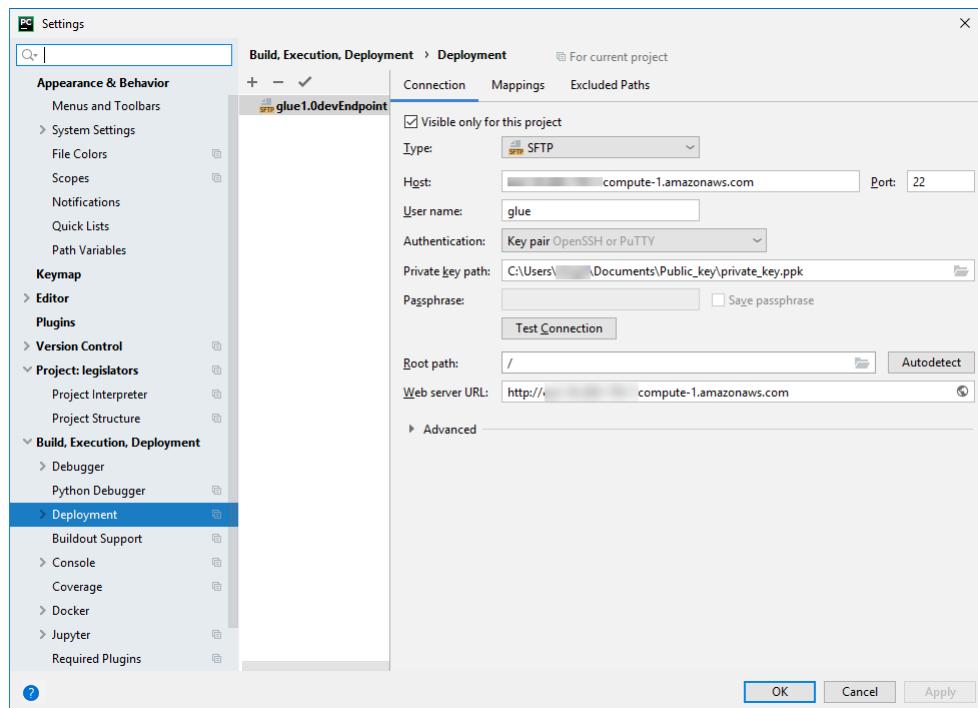
- Configure deployment options to upload the local script to your development endpoint using SFTP (this capability is available only in PyCharm Professional):

- 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:

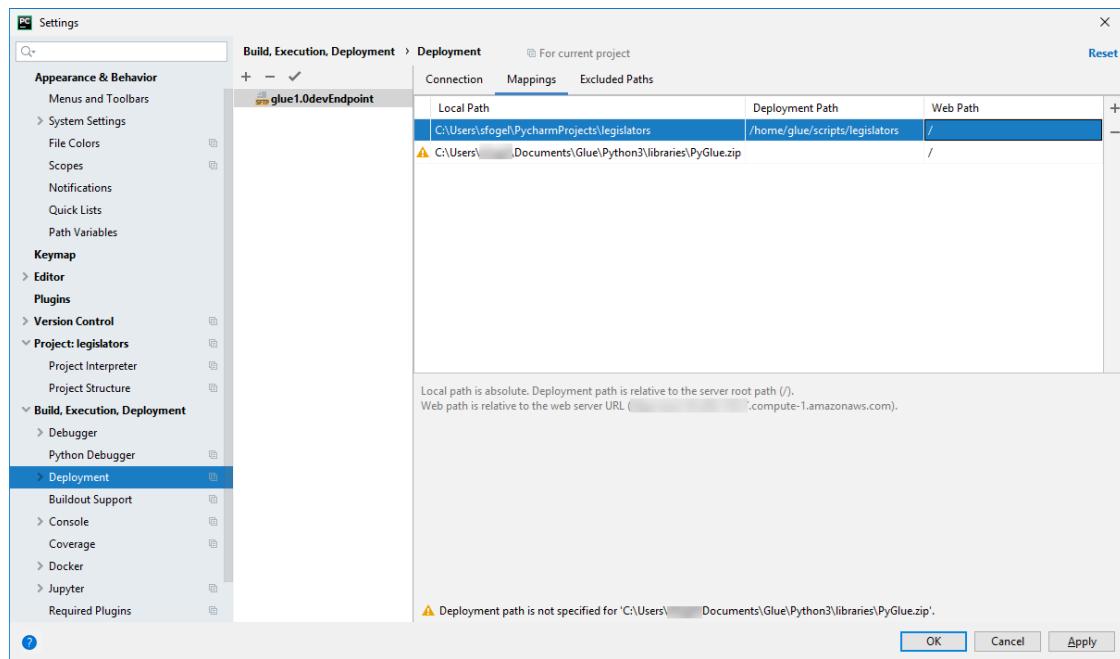


Again, leave the **Settings** dialog box open after you choose **Apply**.

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**.

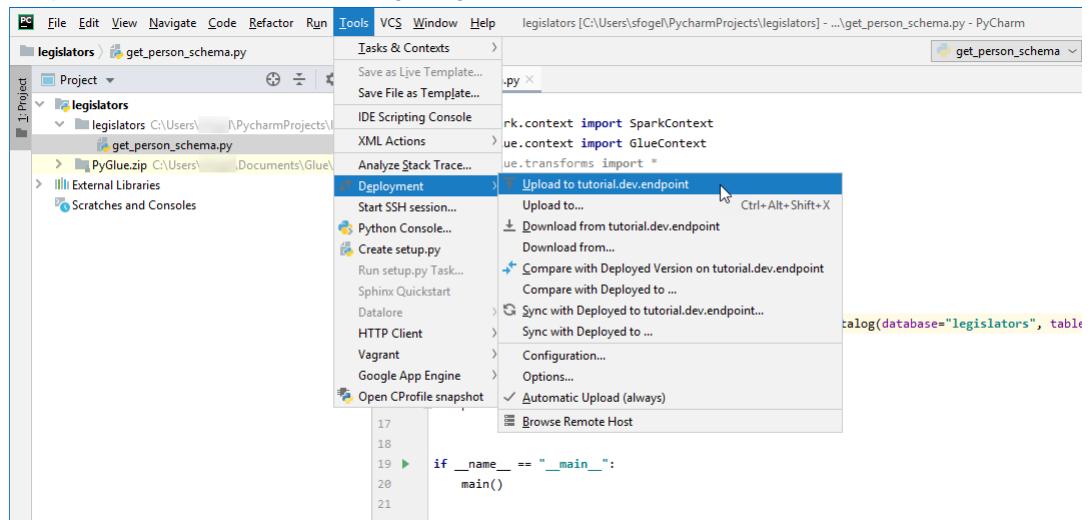
The **Settings** screen should now look something like the following:



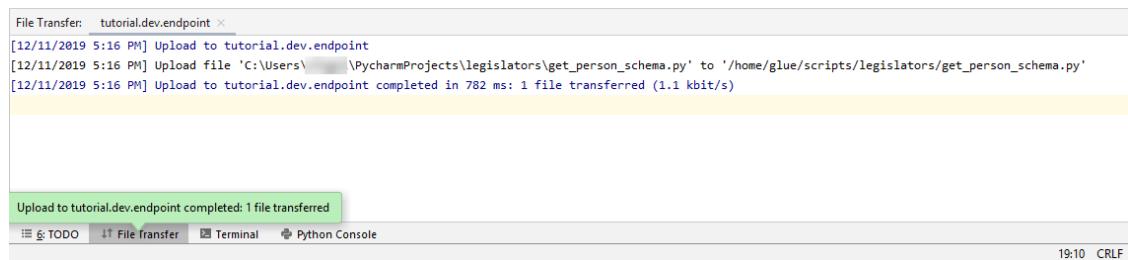
Choose **OK** to close the **Settings** dialog box.

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)**.

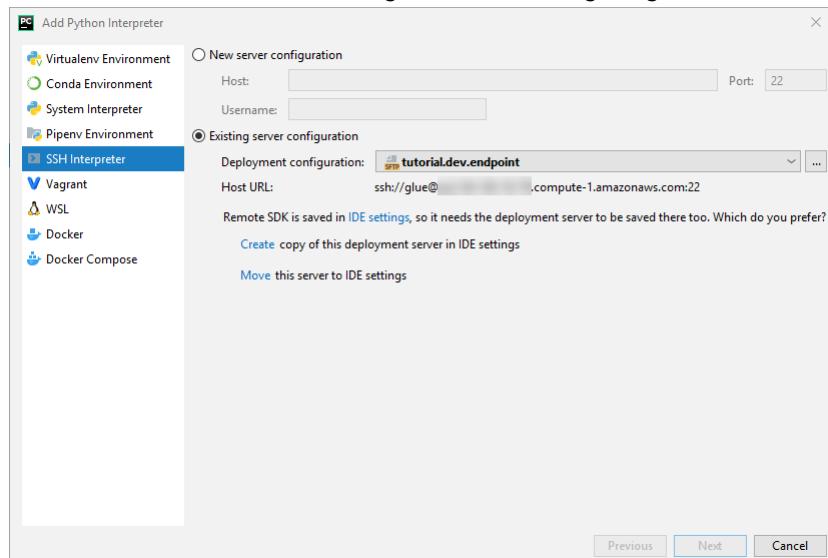
When this option is enabled, PyCharm automatically uploads changed files to the development endpoint.

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.

Advanced Configuration: Sharing Development Endpoints among Multiple Users

This section explains how you can take advantage of development endpoints with SageMaker notebooks in typical use cases to share development endpoints among multiple users.

Single-tenancy Configuration

In single tenant use-cases, to simplify the developer experience and to avoid contention for resources it is recommended that you have each developer use their own development endpoint sized for the project they are working on. This also simplifies the decisions related to worker type and DPU count leaving them up to the discretion of the developer and project they are working on.

You won't need to take care of resource allocation unless you run multiple notebook files concurrently. If you run code in multiple notebook files at the same time, multiple Livy sessions will be launched concurrently. To segregate Spark cluster configurations in order to run multiple Livy sessions at the same time, you can follow the steps which are introduced in multi tenant use-cases.

Multi-tenancy Configuration

Note

Please note, development endpoints are intended to emulate the AWS Glue ETL environment as a single-tenant environment. While multi-tenant use is possible, it is an advanced use-case and it is recommended most users maintain a pattern of single-tenancy for each development endpoint.

In multi tenant use-cases, you might need to take care of resource allocation. The key factor is the number of concurrent users who use a Jupyter notebook at the same time. If your team works in a "follow-the-sun" workflow and there is only one Jupyter user at each time zone, then the number of concurrent users is only one, so you won't need to be concerned with resource allocation. However, if your notebook is shared among multiple users and each user submits code in an ad-hoc basis, then you will need to consider the below points.

To segregate Spark cluster resources among multiple users, you can use SparkMagic configurations. There are two different ways to configure SparkMagic.

(A) Use the %%configure -f Directive

If you want to modify the configuration per Livy session from the notebook, you can run the `%%configure -f` directive on the notebook paragraph.

For example, if you want to run Spark application on 5 executors, you can run the following command on the notebook paragraph.

```
%%configure -f
{"numExecutors":5}
```

Then you will see only 5 executors running for the job on the Spark UI.

We recommend limiting the maximum number of executors for dynamic resource allocation.

```
%%configure -f
{"conf":{"spark.dynamicAllocation.maxExecutors":"5"}}
```

(B) Modify the SparkMagic Config File

SparkMagic works based on the [Livy API](#). SparkMagic creates Livy sessions with configurations such as `driverMemory`, `driverCores`, `executorMemory`, `executorCores`, `numExecutors`, `conf`, etc. Those are the key factors that determine how much resources are consumed from the entire Spark cluster. SparkMagic allows you to provide a config file to specify those parameters which are sent to Livy. You can see a sample config file in this [Github repository](#).

If you want to modify configuration across all the Livy sessions from a notebook, you can modify `/home/ec2-user/.sparkmagic/config.json` to add `session_config`.

To modify the config file on a SageMaker notebook instance, you can follow these steps.

1. Open a SageMaker notebook.
2. Open the Terminal kernel.
3. Run the following commands:

```
sh-4.2$ cd .sparkmagic
sh-4.2$ ls
```

```
config.json logs
sh-4.2$ sudo vim config.json
```

For example, you can add these lines to `/home/ec2-user/.sparkmagic/config.json` and restart the Jupyter kernel from the notebook.

```
"session_configs": {
    "conf": {
        "spark.dynamicAllocation.maxExecutors": "5"
    }
},
```

Guidelines and Best Practices

To avoid this kind of resource conflict, you can use some basic approaches like:

- Have a larger Spark cluster by increasing the `NumberOfWorkers` (scaling horizontally) and upgrading the `workerType` (scaling vertically)
- Allocate fewer resources per user (fewer resources per Livy session)

Your approach will depend on your use case. If you have a larger development endpoint, and there is not a huge amount of data, the possibility of a resource conflict will decrease significantly because Spark can allocate resources based on a dynamic allocation strategy.

As described above, the number of Spark executors can be automatically calculated based on a combination of DPU (or `NumberOfWorkers`) and worker type. Each Spark application launches one driver and multiple executors. To calculate you will need the `NumberOfWorkers = NumberOfExecutors + 1`. The matrix below explains how much capacity you need in your development endpoint based on the number of concurrent users.

Number of concurrent notebook users	Number of Spark executors you want to allocate per user	Total <code>NumberOfWorkers</code> for your dev endpoint
3	5	18
10	5	60
50	5	300

If you want to allocate fewer resources per user, the `spark.dynamicAllocation.maxExecutors` (or `numExecutors`) would be the easiest parameter to configure as a Livy session parameter. If you set the below configuration in `/home/ec2-user/.sparkmagic/config.json`, then SparkMagic will assign a maximum of 5 executors per Livy session. This will help segregating resources per Livy session.

```
"session_configs": {
    "conf": {
        "spark.dynamicAllocation.maxExecutors": "5"
    }
},
```

Suppose there is a dev endpoint with 18 workers (G.1X) and there are 3 concurrent notebook users at the same time. If your session config has `spark.dynamicAllocation.maxExecutors=5` then each

user can make use of 1 driver and 5 executors. There won't be any resource conflicts even when you run multiple notebook paragraphs at the same time.

Trade-offs

With this session config "spark.dynamicAllocation.maxExecutors": "5", you will be able to avoid resource conflict errors and you do not need to wait for resource allocation when there are concurrent user accesses. However, even when there are many free resources (for example, there are no other concurrent users), Spark cannot assign more than 5 executors for your Livy session.

Other Notes

It is a good practice to stop the Jupyter kernel when you stop using a notebook. This will free resources and other notebook users can use those resources immediately without waiting for kernel expiration (auto-shutdown).

Common Issues

Even when following the guidelines, you may experience certain issues.

Session not found

When you try to run a notebook paragraph even though your Livy session has been already terminated, you will see the below message. To activate the Livy session, you need to restart the Jupyter kernel by choosing **Kernel > Restart** in the Jupyter menu, then run the notebook paragraph again.

```
An error was encountered:  
Invalid status code '404' from http://localhost:8998/sessions/13 with error payload:  
"Session '13' not found."
```

Not enough YARN resources

When you try to run a notebook paragraph even though your Spark cluster does not have enough resources to start a new Livy session, you will see the below message. You can often avoid this issue by following the guidelines, however, there might be a possibility that you face this issue. To workaround the issue, you can check if there are any unneeded, active Livy sessions. If there are unneeded Livy sessions, you will need to terminate them to free the cluster resources. See the next section for details.

```
Warning: The Spark session does not have enough YARN resources to start.  
The code failed because of a fatal error:  
    Session 16 did not start up in 60 seconds..  
  
Some things to try:  
a) Make sure Spark has enough available resources for Jupyter to create a Spark context.  
b) Contact your Jupyter administrator to make sure the Spark magics library is configured correctly.  
c) Restart the kernel.
```

Monitoring and Debugging

This section describes techniques for monitoring resources and sessions.

Monitoring and Debugging Cluster Resource Allocation

You can watch the Spark UI to monitor how many resources are allocated per Livy session, and what are the effective Spark configurations on the job. To activate the Spark UI, see [Enabling the Apache Spark Web UI for Development Endpoints](#).

(Optional) If you need a real-time view of the Spark UI, you can configure an SSH tunnel against the Spark history server running on the Spark cluster.

```
ssh -i <private-key.pem> -N -L 8157:<development endpoint public address>:18080  
glue@<development endpoint public address>
```

You can then open <http://localhost:8157> on your browser to view the Spark UI.

Free Unneeded Livy Sessions

Review these procedures to shut down any unneeded Livy sessions from a notebook or a Spark cluster.

(a). Terminate Livy sessions from a notebook

You can shut down the kernel on a Jupyter notebook to terminate unneeded Livy sessions.

(b). Terminate Livy sessions from a Spark cluster

If there are unneeded Livy sessions which are still running, you can shut down the Livy sessions on the Spark cluster.

As a pre-requisite to perform this procedure, you need to configure your SSH public key for your development endpoint.

To log in to the Spark cluster, you can run the following command:

```
$ ssh -i <private-key.pem> glue@<development endpoint public address>
```

You can run the following command to see the active Livy sessions:

```
$ yarn application -list  
20/09/25 06:22:21 INFO client.RMProxy: Connecting to ResourceManager at  
ip-255-1-106-206.ec2.internal/172.38.106.206:8032  
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED,  
RUNNING]):2  
Application-Id Application-Name Application-Type User Queue State Final-State Progress  
Tracking-URL  
application_1601003432160_0005 livy-session-4 SPARK livy default RUNNING UNDEFINED 10%  
http://ip-255-1-4-130.ec2.internal:41867  
application_1601003432160_0004 livy-session-3 SPARK livy default RUNNING UNDEFINED 10%  
http://ip-255-1-179-185.ec2.internal:33727
```

You can then shut down the Livy session with the following command:

```
$ yarn application -kill application_1601003432160_0005  
20/09/25 06:23:38 INFO client.RMProxy: Connecting to ResourceManager at  
ip-255-1-106-206.ec2.internal/255.1.106.206:8032  
Killing application application_1601003432160_0005  
20/09/25 06:23:39 INFO impl.YarnClientImpl: Killed application  
application_1601003432160_0005
```

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 SageMaker notebooks and Apache Zeppelin notebook servers.

- SageMaker provides an integrated Jupyter authoring notebook instance. With AWS Glue, you create and manage SageMaker notebooks. You can also open SageMaker notebooks from the AWS Glue console.

In addition, you can use Apache Spark with SageMaker on AWS Glue development endpoints which support SageMaker (but not AWS Glue ETL jobs). SageMaker Spark is an open source Apache Spark library for SageMaker. For more information, 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 \(p. 293\)](#).

For more information about creating development endpoints, see [Viewing Development Endpoint Properties \(p. 256\)](#).

Important

Managing 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 \(p. 287\)](#)
- [Working with Notebooks on the AWS Glue Console \(p. 293\)](#)

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 \(p. 256\)](#).

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" \(p. 256\)](#).)
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 \(p. 291\)](#). 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:  
  
*****
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 SageMaker notebooks and Apache Zeppelin notebook servers. On the AWS Glue notebooks page, you can create 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 \(p. 258\)](#).

The **Notebooks** page on the AWS Glue console lists all the 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 SageMaker notebook that is attached to a development endpoint to another development endpoint as needed. The switch development endpoint action is only supported for SageMaker notebooks that were created after November 21, 2019.

To switch an 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 SageMaker notebook periodically checks whether it is connected to the attached development endpoint. If it isn't connected, the notebook tries to reconnect automatically.

SageMaker Notebooks on the AWS Glue Console

The following are some of the properties for SageMaker notebooks. The console displays some of these properties when you view the details of a notebook.

Important

AWS Glue only manages SageMaker notebooks in certain AWS Regions. For more information, see [Managing Notebooks \(p. 285\)](#).

Before you begin, ensure that you have permissions to manage 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 \(p. 17\)](#).

Notebook name

The unique name of the 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 SageMaker notebook.

This role has a trust relationship to 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 SageMaker Notebooks \(p. 30\)](#).

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.

Starting Jobs and Crawlers Using Triggers

In AWS Glue, you can create Data Catalog objects called triggers, which you can use to either manually or automatically start one or more crawlers or extract, transform, and load (ETL) jobs. Using triggers, you can design a chain of dependent jobs and crawlers.

Note

You can accomplish the same thing by defining *workflows*. Workflows are preferred for creating complex multi-job ETL operations. For more information, see [Performing Complex ETL Activities Using Blueprints and Workflows \(p. 373\)](#).

Topics

- [AWS Glue Triggers \(p. 296\)](#)
- [Adding Triggers \(p. 297\)](#)
- [Activating and Deactivating Triggers \(p. 298\)](#)

AWS Glue Triggers

When *fired*, a trigger can start specified jobs and crawlers. A trigger fires on demand, based on a schedule, or based on a combination of events.

Note

Only two crawlers can be activated by a single trigger. If you want to crawl multiple data stores, use multiple sources for each crawler instead of running multiple crawlers simultaneously.

A trigger can exist in one of several states. A trigger is either CREATED, ACTIVATED, or DEACTIVATED. There are also transitional states, such as ACTIVATING. To temporarily stop a trigger from firing, you can deactivate it. You can then reactivate it later.

There are three types of triggers:

Scheduled

A time-based trigger based on cron.

You can create a trigger for a set of jobs or crawlers based on a schedule. You can specify constraints, such as the frequency that the jobs or crawlers run, which days of the week they run, and at what time. These constraints are based on cron. When you're setting up a schedule for a trigger, 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 \(p. 301\)](#).

Conditional

A trigger that fires when a previous job or crawler or multiple jobs or crawlers satisfy a list of conditions.

When you create a conditional trigger, you specify a list of jobs and a list of crawlers to watch. For each watched job or crawler, you specify a status to watch for, such as succeeded, failed, timed out, and so on. The trigger fires if the watched jobs or crawlers end with the specified statuses. You can configure the trigger to fire when any or all of the watched events occur.

For example, you could configure a trigger T1 to start job J3 when both job J1 and job J2 successfully complete, and another trigger T2 to start job J4 if either job J1 or job J2 fails.

The following table lists the job and crawler completion states (events) that triggers watch for.

Job completion states	Crawler completion states
<ul style="list-style-type: none"> • SUCCEEDED • STOPPED • FAILED • TIMEOUT 	<ul style="list-style-type: none"> • SUCCEEDED • FAILED • CANCELLED

On-demand

A trigger that fires when you activate it. On-demand triggers never enter the ACTIVATED or DEACTIVATED state. They always remain in the CREATED state.

So that they are ready to fire as soon as they exist, you can set a flag to activate scheduled and conditional triggers when you create them.

Important

Jobs or crawlers that run as a result of other jobs or crawlers completing are referred to as *dependent*. Dependent jobs or crawlers are only started if the job or crawler that completes was started by a trigger. All jobs or crawlers in a dependency chain must be descendants of a single **scheduled or on-demand** trigger.

Passing Job Parameters with Triggers

A trigger can pass parameters to the jobs that it starts. Parameters include job arguments, timeout value, security configuration, and more. If the trigger starts multiple jobs, the parameters are passed to each job.

The following are the rules for job arguments passed by a trigger:

- If the key in the key-value pair matches a default job argument, the passed argument overrides the default argument. If the key doesn't match a default argument, then the argument is passed as an additional argument to the job.
- If the key in the key-value pair matches a non-overridable argument, the passed argument is ignored.

For more information, see [the section called “Triggers” \(p. 866\)](#) in the AWS Glue API.

Adding Triggers

You can add a trigger using the AWS Glue console, the AWS Command Line Interface (AWS CLI), or the AWS Glue API.

Note

Currently, the AWS Glue console supports only jobs, not crawlers, when working with triggers. You can use the AWS CLI or AWS Glue API to configure triggers with both jobs and crawlers.

To add a trigger (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, under **ETL**, choose **Triggers**. Then choose **Add trigger**.
3. 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 frequency and time.
- **Job events:** A conditional trigger. The trigger fires when any or all jobs in the list match their designated statuses. For the trigger to fire, the watched jobs must have been started by triggers. For any job you choose, you can only watch one job event (completion status).
- **On-demand:** The trigger fires when it is activated.

4. Complete the trigger wizard. On the **Review** page, you can activate **Schedule** and **Job events** (conditional) triggers immediately by selecting **Enable trigger on creation**.

To add a trigger (AWS CLI)

- Enter a command similar to the following.

```
aws glue create-trigger --name MyTrigger --type SCHEDULED --schedule "cron(0 12 * * ? *)" --actions CrawlerName=MyCrawler --start-on-creation
```

This command creates a schedule trigger named `MyTrigger`, which runs every day at 12:00pm UTC and starts a crawler named `MyCrawler`. The trigger is created in the activated state.

For more information, see [the section called “AWS Glue Triggers” \(p. 296\)](#).

Activating and Deactivating Triggers

You can activate or deactivate a trigger using the AWS Glue console, the AWS Command Line Interface (AWS CLI), or the AWS Glue API.

Note

Currently, the AWS Glue console supports only jobs, not crawlers, when working with triggers. You can use the AWS CLI or AWS Glue API to configure triggers with both jobs and crawlers.

To activate or deactivate a trigger (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, under **ETL**, choose **Triggers**.
3. Select the check box next to the desired trigger, and on the **Action** menu choose **Enable trigger** to activate the trigger or **Disable trigger** to deactivate the trigger.

To activate or deactivate a trigger (AWS CLI)

- Enter one of the following commands.

```
aws glue start-trigger --name MyTrigger
```

```
aws glue stop-trigger --name MyTrigger
```

Starting a trigger activates it, and stopping a trigger deactivates it. When you activate an on-demand trigger, it fires immediately.

For more information, see [the section called “AWS Glue Triggers” \(p. 296\)](#).

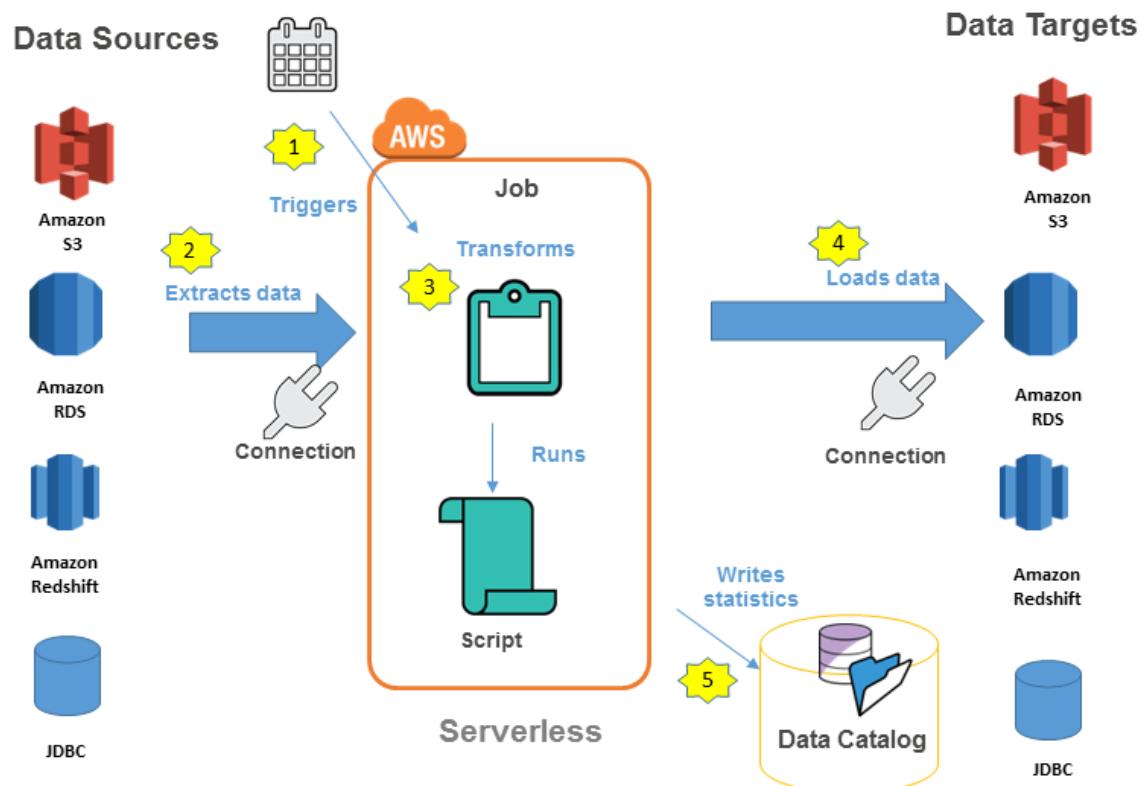
Running and Monitoring AWS Glue

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 \(p. 301\)](#)
- [Time-Based Schedules for Jobs and Crawlers \(p. 301\)](#)
- [Tracking Processed Data Using Job Bookmarks \(p. 303\)](#)
- [Workload Partitioning with Bounded Execution \(p. 309\)](#)
- [AWS Tags in AWS Glue \(p. 310\)](#)
- [Automating AWS Glue with CloudWatch Events \(p. 314\)](#)
- [Monitoring Jobs Using the Apache Spark Web UI \(p. 315\)](#)
- [AWS Glue Spark shuffle manager with Amazon S3 \(p. 325\)](#)
- [Monitoring with AWS Glue job run insights \(p. 327\)](#)
- [Monitoring with Amazon CloudWatch \(p. 331\)](#)
- [Job Monitoring and Debugging \(p. 349\)](#)
- [Logging AWS Glue API Calls with AWS CloudTrail \(p. 370\)](#)
- [AWS Glue Job Run Statuses \(p. 372\)](#)

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.

To learn more about configuring jobs and crawlers to run using a schedule, see [Starting Jobs and Crawlers Using Triggers \(p. 296\)](#).

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	, - * /
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

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
15	12	*	*	?	*	Run at 12:15 pm (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 pm (UTC) every 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
Version 1.0 and later	JSON, CSV, Apache Avro, XML, Parquet, ORC

For information about AWS Glue versions, see [Defining Job Properties for Spark Jobs \(p. 198\)](#).

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.
- AWS Glue doesn't support using case-sensitive columns as job bookmark keys.

Topics

- [Using Job Bookmarks in AWS Glue \(p. 304\)](#)
- [Using Job Bookmarks with the AWS Glue Generated Script \(p. 306\)](#)

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	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: <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

Job bookmark	Description
	<p>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> <p>The job bookmark state is not updated when this option set is specified.</p> <p>The sub-options are optional, however when used both the sub-options needs to be provided.</p>

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 \(p. 472\)](#).

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 AWS 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 \(Python: reset_job_bookmark\) \(p. 865\)](#) 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
```

When you rewind or reset a bookmark, AWS Glue does not clean the target files because there could be multiple targets and targets are not tracked with job bookmarks. Only source files are tracked with job bookmarks. You can create different output targets when rewinding and reprocessing the source files to avoid duplicate data in your output.

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 \(p. 989\)](#).

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 \(p. 573\)](#). For more information about PySpark extensions, see [AWS Glue PySpark Extensions Reference \(p. 553\)](#).

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.

In the AWS Glue service database, the bookmark states for all the transformations are stored together as key-value pairs:

```
{  
    "job_name" : ...,  
    "run_id": ...,  
    "run_number": ...,  
    "attempt_number": ...  
    "states": {  
        "transformation_ctx1" : {  
            bookmark_state1  
        },  
        "transformation_ctx2" : {  
            bookmark_state2  
        }  
    }  
}
```

The `transformation_ctx` serves as the key to search the bookmark state for a specific source in your script. For the bookmark to work properly, you should always keep the source and the associated `transformation_ctx` consistent. Changing the source property or renaming the `transformation_ctx` may make the previous bookmark invalid and the time stamp based filtering may not yield the correct result.

Best practices

The following are best practices for using job bookmarks with the AWS Glue generated script.

- *Always have job.init() in the beginning of the script and the job.commit() in the end of the script.* These two functions are used to initialize the bookmark service and update the state change to the service. Bookmarks won't work without calling them.
- *Do not change the data source property with the bookmark enabled.* For example, there is a datasource0 pointing to an Amazon S3 input path A, and the job has been reading from a source which has been running for several rounds with the bookmark enabled. If you change the input path of datasource0 to Amazon S3 path B without changing the transformation_ctx, the AWS Glue job will use the old bookmark state stored. That will result in missing or skipping files in the input path B as AWS Glue would assume that those files had been processed in previous runs.
- *Use a catalog table with bookmarks for better partition management.* Bookmarks works both for data sources from the Data Catalog or from options. However, it's difficult to remove/add new partitions with the from options approach. Using a catalog table with crawlers can provide better automation to track the newly added [partitions](#) and give you the flexibility to select particular partitions with a [pushdown predicate](#).
- *Use the AWS Glue Amazon S3 file lister for large datasets.* A bookmark will list all files under each input partition and do the filtering, so if there are too many files under a single partition the bookmark can run into driver OOM. Use the AWS Glue Amazon S3 file lister to avoid listing all files in memory at once.

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 \(p. 575\)](#) API, and the [DynamicFrameWriter Class \(p. 571\)](#) 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"]}, "jobBookmarkKeysSortOrder": "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()
```

You can specify `jobBookmarkKeys` and `jobBookmarkKeysSortOrder` in the following ways:

- `create_dynamic_frame.from_catalog` — Use `additional_options`.
- `create_dynamic_frame.from_options` — Use `connection_options`.

For more information about connection options related to job bookmarks, see [the section called “JDBC connections” \(p. 492\)](#).

Workload Partitioning with Bounded Execution

Errors in Spark applications commonly arise from inefficient Spark scripts, distributed in-memory execution of large-scale transformations, and dataset abnormalities. There are many reasons that may cause driver or executor out of memory issues, for example a data skew, listing too many objects, or large data shuffles. These issues often appear when you are processing huge amounts of backlog data with Spark.

AWS Glue allows you to solve OOM issues and make your ETL processing easier with workload partitioning. With workload partitioning enabled, each ETL job run only picks unprocessed data, with an upper bound on the dataset size or the number of files to be processed with this job run. Future job runs will process the remaining data. For example, if there are 1000 files need to be processed, you can set the number of files to be 500 and separate them into two job runs.

Workload partitioning is supported only for Amazon S3 data sources.

Enabling Workload Partitioning

You can enable bounded execution by manually setting the options in your script or by adding catalog table properties.

To enable workload partitioning with bounded execution in your script:

1. To avoid reprocessing data, enable job bookmarks in the new job or existing job. For more information, see [Tracking Processed Data Using Job Bookmarks](#).
2. Modify your script and set the bounded limit in the additional options in the AWS Glue `getSource` API. You should also set the transformation context for the job bookmark to store the `state` element. For example:

Python

```
glueContext.create_dynamic_frame.from_catalog(
    database = "database",
    tableName = "table_name",
    redshift_tmp_dir = "",
    transformation_ctx = "datasource0",
    additional_options = {
        "boundedFiles" : "500", # need to be string
        # "boundedSize" : "1000000000" unit is byte
    }
)
```

Scala

```
val datasource0 = glueContext.getCatalogSource(  
    database = "database", tableName = "table_name", redshiftTmpDir = "",  
    transformationContext = "datasource0",  
    additionalOptions = JsonOptions(  
        Map("boundedFiles" -> "500") // need to be string  
        // "boundedSize" -> "1000000000" unit is byte  
    )  
).getDynamicFrame()  
  
val connectionOptions = JsonOptions(  
    Map("paths" -> List(baseLocation), "boundedFiles" -> "30")  
)  
val source = glueContext.getSource("s3", connectionOptions, "datasource0", "")
```

To enable workload partitioning with bounded execution in your Data Catalog table:

1. Set the key-value pairs in the `parameters` field of your table structure in the Data Catalog. For more information, see [Viewing and Editing Table Details](#).
2. Set the upper limit for the dataset size or the number of files processed:
 - Set `boundedSize` to the target size of the dataset in bytes. The job run will stop after reaching the target size from the table.
 - Set `boundedFiles` to the target number of files. The job run will stop after processing the target number of files.

Note

You should only set one of `boundedSize` or `boundedFiles`, as only a single boundary is supported.

Setting up an AWS Glue Trigger to Automatically Run the Job

Once you have enabled bounded execution, you can set up an AWS Glue trigger to automatically run the job and incrementally load the data in sequential runs. Go to the AWS Glue Console and create a trigger, setup the schedule time, and attach to your job. Then it will automatically trigger the next job run and process the new batch of data.

You can also use AWS Glue workflows to orchestrate multiple jobs to process data from different partitions in parallel. For more information, see [AWS Glue Triggers](#) and [AWS Glue Workflows](#).

For more information on use cases and options, please refer to the blog [Optimizing Spark applications with workload partitioning in AWS Glue](#).

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 \(p. 63\)](#).

In AWS Glue, you can tag the following resources:

- Crawler
- Job
- Trigger
- Workflow
- Development endpoint
- Machine learning transform

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.

Tagging support for AWS Glue connections

You can restrict `CreateConnection`, `UpdateConnection`, `GetConnection` and, `DeleteConnection` action permission based on the resource tag. This enables you to implement the least privilege access control on AWS Glue jobs with JDBC data sources which need to fetch JDBC connection information from the Data Catalog.

Example usage

Create an AWS Glue connection with the tag `["connection-category", "dev-test"]`.

Specify the tag condition for the `GetConnection` action in the IAM policy.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "glue:GetConnection"  
    ],  
    "Resource": "*",  
    "Condition": {  
        "ForAnyValue:StringEquals": {  
            "aws:ResourceTag/tagKey": "dev-test"  
        }  
    }  
}
```

```
}
```

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": "*"
                }
              ]
            }
          }
        ],
        "Tags": [
          {
            "Key": "Name",
            "Value": "MyJobRole"
          }
        ]
      }
    },
    "MyJob": {
      "Type": "AWS::Glue::Job",
      "Properties": {
        "Command": {
          "Name": "glueetl",
          "ScriptLocation": "s3://aws-glue-scripts//prod-job1"
        },
        "Role": "arn:aws:iam::123456789012:role/MyJobRole"
      }
    }
  }
}
```

```
"DefaultArguments": {
    "--job-bookmark-option": "job-bookmark-enable"
},
"ExecutionProperty": {
    "MaxConcurrentRuns": 2
},
"MaxRetries": 0,
"Name": "cf-job1",
"Role": {
    "Ref": "MyJobRole",
    "Tags": {
        "key1": "value1",
        "key2": "value2"
    }
}
}
```

AWS CloudFormation YAML

```
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 \(p. 63\)](#).

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'])

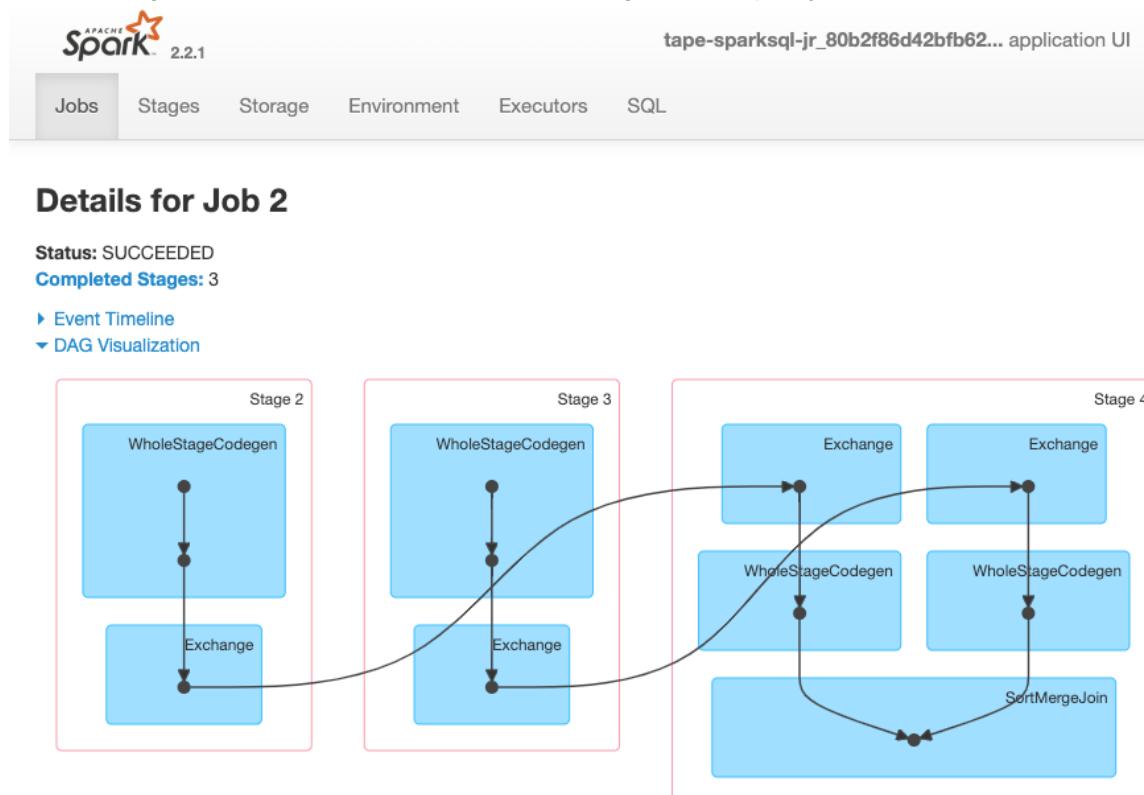
df_persons = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
persons.json")
df_memberships = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
memberships.json")

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/")

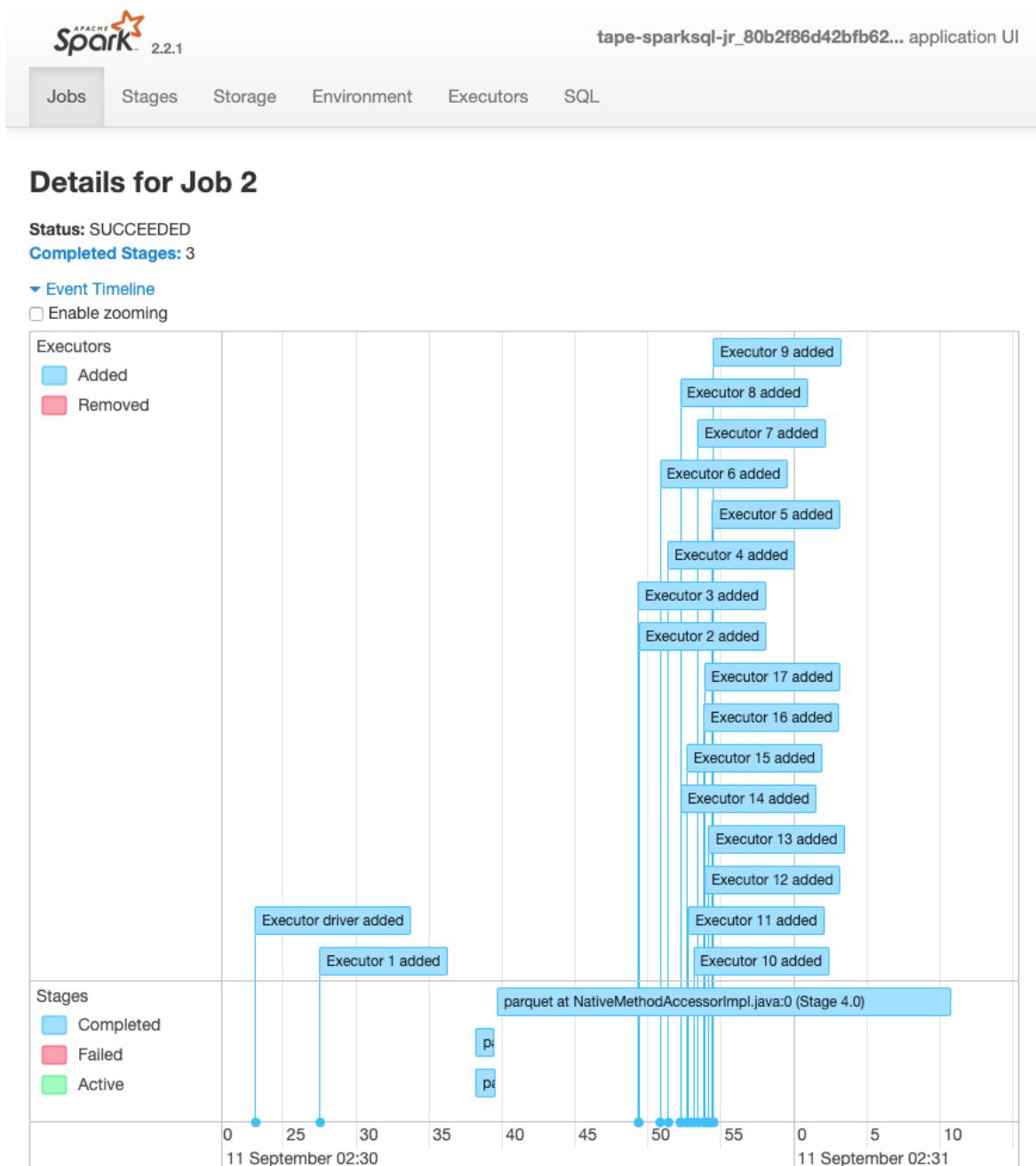
job.commit()

```

The following DAG visualization shows the different stages in this Spark job.



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

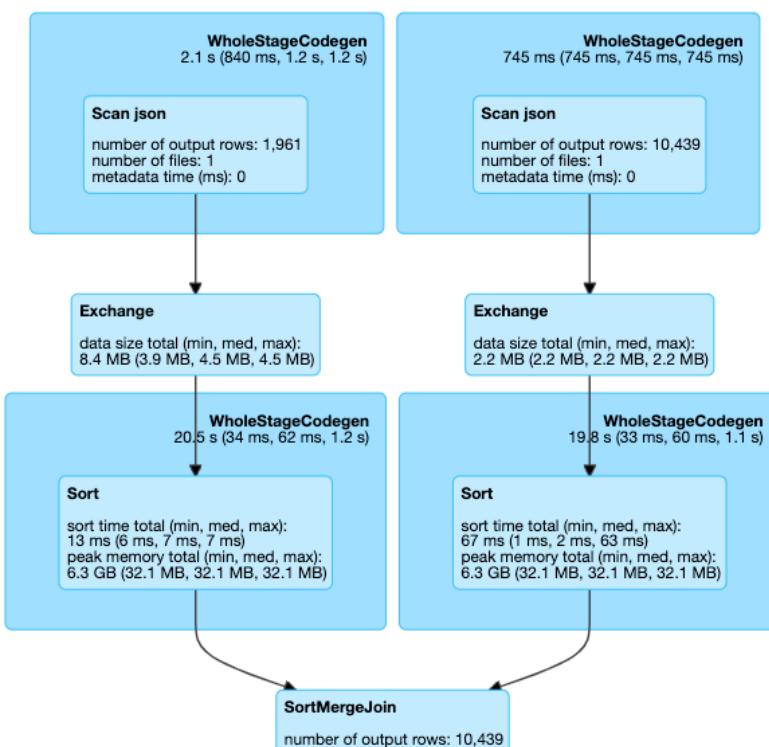


Details for Query 0

Submitted Time: 2019/09/11 02:30:37

Duration: 34 s

Succeeded Jobs: 2



▼ Details

```

== Parsed Logical Plan ==
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
Relation[area_id#45,end_date#46,legislative_period_id#47,on_behalf_of_id#48,organization_id#49,person_id#50,role#51,start_date#52] json

== Analyzed Logical Plan ==
birth_date: string, contact_details: array<struct<type:string,value:string>>, death_date: string, family_name: string, gender: string, given_name: string, id: string, identifiers: array<struct<identifier:string,scheme:string>>, image: string, images: array<struct<url:string>>, links: array<struct<note:string,url:string>>, name: string, other_names: array<struct<lang:string,name:string,note:string>>, sort_name: string, area_id: string, end_date: string, legislative_period_id: string, on_behalf_of_id: string, organization_id: string, person_id: string, role: string, start_date: string
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
Relation[area_id#45,end_date#46,legislative_period_id#47,on_behalf_of_id#48,organization_id#49,person_id#50,role#51,start_date#52] json
  
```

```

== Optimized Logical Plan ==
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
  
```

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 \(p. 319\)](#)
- [Enabling the Apache Spark Web UI for Development Endpoints \(p. 320\)](#)
- [Launching the Spark History Server \(p. 321\)](#)

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 \(Console\) \(p. 319\)](#)
- [Configuring the Spark UI \(AWS CLI\) \(p. 320\)](#)

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 \(Console\) \(p. 320\)](#)
- [Enabling the Spark UI \(AWS CLI\) \(p. 320\)](#)

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**.
6. 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 \(p. 321\)](#)
- [Launching the Spark History Server and Viewing the Spark UI Using Docker \(p. 324\)](#)

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

1. Choose one of the **Launch Stack** buttons in the following table. This launches the stack on the AWS CloudFormation console.

Region	Launch for Glue 1.0/2.0	Launch for Glue 3.0
US East (Ohio)	Launch Stack	Launch Stack
US East (N. Virginia)	Launch Stack	Launch Stack
US West (N. California)	Launch Stack	Launch Stack
US West (Oregon)	Launch Stack	Launch Stack
Africa (Cape Town)	Must first enable console access to the region. Launch Stack	Launch Stack
Asia Pacific (Hong Kong)	Must first enable console access to the region.	Launch Stack

Region	Launch for Glue 1.0/2.0	Launch for Glue 3.0
	Launch Stack	
Asia Pacific (Mumbai)	Launch Stack	Launch Stack
Asia Pacific (Osaka)	Launch Stack	Launch Stack
Asia Pacific (Seoul)	Launch Stack	Launch Stack
Asia Pacific (Singapore)	Launch Stack	Launch Stack
Asia Pacific (Sydney)	Launch Stack	Launch Stack
Asia Pacific (Tokyo)	Launch Stack	Launch Stack
Canada (Central)	Launch Stack	Launch Stack
Europe (Frankfurt)	Launch Stack	Launch Stack
Europe (Ireland)	Launch Stack	Launch Stack
Europe (London)	Launch Stack	Launch Stack
Europe (Milan)	Must first enable console access to the region. Launch Stack	Launch Stack
Europe (Paris)	Launch Stack	Launch Stack
Europe (Stockholm)	Launch Stack	Launch Stack
Middle East (Bahrain)	Must first enable console access to the region. Launch Stack	Launch Stack
South America (São Paulo)	Launch Stack	Launch Stack

2. On the **Specify template** page, choose **Next**.
3. On the **Specify stack details** page, enter the **Stack name**. Enter additional information under **Parameters**.

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** — Enter a 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.

c. **Choose Next.**

4. On the **Configure stack options** page, to use the current user credentials for determining how CloudFormation can create, modify, or delete resources in the stack, choose **Next**. You can also specify a role in the **Permissions** section to use instead of the current user permissions, and then 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. Determine if you want to use your user credentials or federated user credentials to access AWS.

- To use the current user credentials for accessing AWS, get the values to use for AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY in the docker run command. For more information, see [Managing access keys for IAM users](#) in the *IAM User Guide*.
- To use SAML 2.0 federated users for accessing AWS, get the values for AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN. For more information, see [Requesting temporary security credentials](#)

3. Determine the location of your event log directory, to use in the docker run command.
4. Build the Docker image using the files in the local directory, using the name glue/sparkui, and the tag latest.

```
$ docker build -t glue/sparkui:latest .
```

5. Create and start the docker container.

In the following commands, use the values obtained previously in steps 2 and 3.

- a. To create the docker container using your user credentials, use a command similar to the following

```
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"
```

- b. To create the docker container using temporary credentials, use org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider as the provider, and provide the credential values obtained in step 2. For more information, see [Using Session Credentials with TemporaryAWSCredentialsProvider](#) in the *Hadoop: Integration with Amazon Web Services* documentation.

```
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  
-Dspark.hadoop.fs.s3a.session.token=AWS_SESSION_TOKEN  
-  
Dspark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsP
```

```
-p 18080:18080 glue/sparkui.latest "/opt/spark/bin/spark-class  
org.apache.spark.deploy.history.HistoryServer"
```

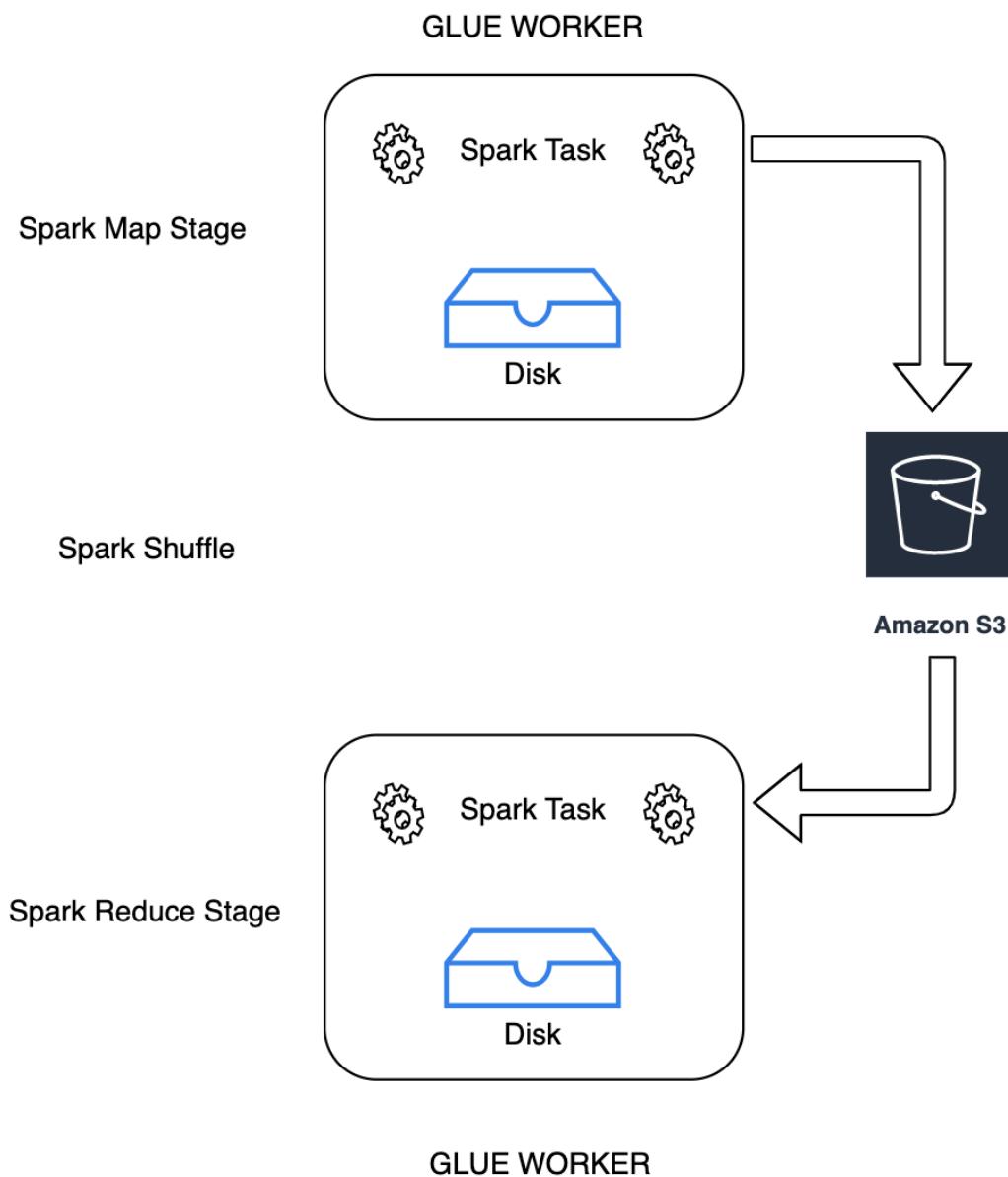
6. Open <http://localhost:18080> in your browser to view the Spark UI locally.

AWS Glue Spark shuffle manager with Amazon S3

Shuffling is an important step in a Spark job whenever data is rearranged between partitions. This is required because wide transformations such as `join`, `groupByKey`, `reduceByKey`, and `repartition` require information from other partitions to complete processing. Spark gathers the required data from each partition and combines it into a new partition. During a shuffle, data is written to disk and transferred across the network. As a result, the shuffle operation is bound to local disk capacity. Spark throws a `No space left on device` or `MetadataFetchFailedException` error when there is not enough disk space left on the executor and there is no recovery.

Solution

With AWS Glue 2.0, you can now use Amazon S3 to store Spark shuffle and spill data. Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. This solution disaggregates compute and storage for your Spark jobs, and gives complete elasticity and low-cost shuffle storage, allowing you to run your most shuffle-intensive workloads reliably.



We are introducing a new shuffle manager (called `GlueShuffleManager`) which will write and read shuffle files to/from Amazon S3. You can turn on Amazon S3 shuffling to run your AWS Glue jobs reliably without failures if they are known to be bound by the local disk capacity for large shuffle operations. In some cases, shuffling to Amazon S3 is marginally slower than local disk (or EBS) if you have a large number of small partitions or shuffle files written out to Amazon S3.

Using AWS Glue Spark shuffle manager from the AWS Console

To set up the AWS Glue Spark shuffle manager using the AWS Glue console or AWS Glue Studio when configuring a job: choose the `--write-shuffle-files-to-s3` job parameter to turn on Amazon S3 shuffling for the job.

Job parameters

Key	Value - optional	
<input type="text"/> --write-shuffle-files- <input type="button" value="X"/>	<input type="text"/> true <input type="button" value="X"/>	<input type="button" value="Remove"/>

You can add 49 more parameters.

Using AWS Glue Spark shuffle manager

The following job parameters turn on and tune the AWS Glue shuffle manager.

- `--write-shuffle-files-to-s3` — The main flag, which when true enables the AWS Glue Spark shuffle manager to use Amazon S3 buckets for writing and reading shuffle data. When false, or not specified the shuffle manager is not used.
- `--write-shuffle-spills-to-s3` — An optional flag that when true allows you to offload spill files to Amazon S3 buckets, which provides additional resiliency to your Spark job. This is only required for large workloads that spill a lot of data to disk. When false, no intermediate spill files are written. This flag is disabled by default.
- `--conf spark.shuffle.glue.s3ShuffleBucket=S3://<shuffle-bucket>` — Another optional flag that specifies the Amazon S3 bucket where you write the shuffle files. By default, `--TempDir/shuffle-data`.

AWS Glue supports all other shuffle related configurations provided by Spark.

Notes and limitations

The following are notes or limitations for the AWS Glue shuffle manager:

- AWS Glue currently supports the shuffle manager only on AWS Glue version 2.0, and Spark 2.4.3.
- Make sure the location of the shuffle bucket is in the same AWS Region in which the job runs.
- Set the Amazon S3 storage lifecycle policies on the prefix as the shuffle manager does not clean the files after the job is done.
- You can use this feature if your data is skewed.

Monitoring with AWS Glue job run insights

AWS Glue job run insights is a feature in AWS Glue that simplifies job debugging and optimization for your AWS Glue jobs. AWS Glue provides [Spark UI](#), and [CloudWatch logs and metrics](#) for monitoring your AWS Glue jobs. With this feature, you get this information about your AWS Glue job's execution:

- Line number of your AWS Glue job script that had a failure.
- Spark action that executed last in the Spark query plan just before the failure of your job.
- Spark exception events related to the failure presented in a time-ordered log stream.
- Root cause analysis and recommended action (such as tuning your script) to fix the issue.
- Common Spark events (log messages relating to a Spark action) with a recommended action that addresses the root cause.

All these insights are available to you using two new log streams in the CloudWatch logs for your AWS Glue jobs.

Requirements

The AWS Glue job run insights feature is available for AWS Glue version 2.0 and AWS Glue version 3.0. You can follow the [migration guide](#) for your existing jobs to upgrade them from older AWS Glue versions.

Enabling job run insights for an AWS Glue ETL job

You can enable job run insights through AWS Glue Studio or the CLI.

AWS Glue Studio

When creating a job via AWS Glue Studio, you can enable or disable job run insights under the **Job Details** tab. Check that the **Generate job insights** box is selected (enabled by default).

Requested number of workers
The number of workers you want AWS Glue to allocate to this job.

10

The maximums are 299 for G.1X and 149 for G.2X, and the minimum is 2.

Generate job insights
AWS Glue will analyze your job runs and provide insights on how to optimize your jobs and the reasons for job failures.

Command Line

If creating a job via the CLI, you can start a job run with a single new **job** parameter: `--enable-job-insights = true`.

By default, the job run insights log streams are created under the same default log group used by [AWS Glue continuous logging](#), that is, `/aws-glue/jobs/logs-v2/`. You may set up custom log group name, log filters and log group configurations using the same set of arguments for continuous logging. For more information, see [Enabling Continuous Logging for AWS Glue Jobs](#).

Accessing the Job Run Insights Log Streams in CloudWatch

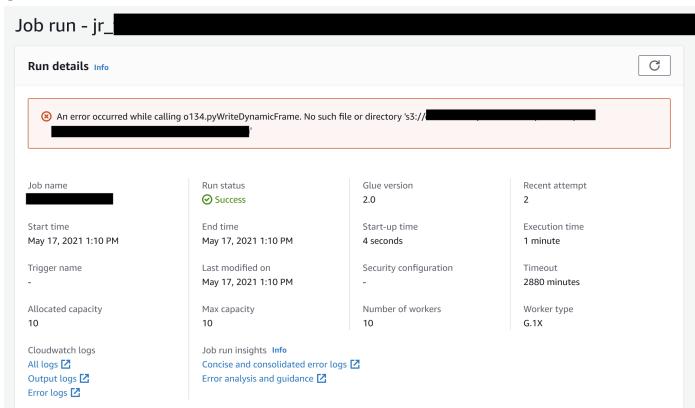
With the job run insights feature enabled, there may be two log streams created when a job run fails. When a job finishes successfully, neither of the streams are generated.

1. *Exception analysis log stream:* `<job-run-id>-job-insights-rca-driver`. This stream provides the following:
 - Line number of your AWS Glue job script that caused the failure.
 - Spark action that executed last in the Spark query plan (DAG).
 - Concise time-ordered events from the Spark driver and executors that are related to the exception. You can find details such as complete error messages, the failed Spark task and its executors ID that help you to focus on the specific executor's log stream for a deeper investigation if needed.
2. *Rule-based insights stream:*
 - Root cause analysis and recommendations on how to fix the errors (such as using a specific job parameter to optimize the performance).
 - Relevant Spark events serving as the basis for root cause analysis and a recommended action.

Note

The first stream will exist only if any exception Spark events are available for a failed job run, and the second stream will exist only if any insights are available for the failed job run. For example, if your job finishes successfully, neither of the streams will be generated; if your job fails but there isn't a service-defined rule that can match with your failure scenario, then only the first stream will be generated.

If the job is created from AWS Glue Studio, the links to the above streams are also available under the job run details tab (Job run insights) as "Concise and consolidated error logs" and "Error analysis and guidance".



Example for AWS Glue job run insights

In this section we present an example of how the job run insights feature can help you resolve an issue in your failed job. In this example, a user forgot to import the required module (`tensorflow`) in an AWS Glue job to analyze and build a machine learning model on their data.

```
1. import sys
2. from awsglue.transforms import *
3. from awsglue.utils import getResolvedOptions
4. from pyspark.context import SparkContext
5. from awsglue.context import GlueContext
6. from awsglue.job import Job
7. from pyspark.sql.types import *
8. from pyspark.sql.functions import udf,col
9.
10. args = getResolvedOptions(sys.argv, ['JOB_NAME'])
11.
12. sc = SparkContext()
13. glueContext = GlueContext(sc)
14. spark = glueContext.spark_session
15. job = Job(glueContext)
16. job.init(args['JOB_NAME'], args)
17.
18. data_set_1 = [1, 2, 3, 4]
19. data_set_2 = [5, 6, 7, 8]
20.
21. scoresDf = spark.createDataFrame(data_set_1, IntegerType())
22.
23. def data_multiplier_func(factor, data_vector):
24.     import tensorflow as tf
25.     with tf.compat.v1.Session() as sess:
26.         x1 = tf.constant(factor)
27.         x2 = tf.constant(data_vector)
28.         result = tf.multiply(x1, x2)
29.         return sess.run(result).tolist()
```

```

30.
31. data_multiplier_udf = udf(lambda x:data_multiplier_func(x, data_set_2),
   ArrayType(IntegerType(),False))
32. factoredDf = scoresDf.withColumn("final_value", data_multiplier_udf(col("value")))
33. print(factoredDf.collect())

```

Without the job run insights feature, as the job fails, you only see this message thrown by Spark:

An error occurred while calling o111.collectToPython. Traceback (most recent call last):

The message is ambiguous and limits your debugging experience. In this case, this feature provides with you additional insights in two CloudWatch log streams:

1. The job-insights-rca-driver log stream:

- Exception events:** This log stream provides you the Spark exception events related to the failure collected from the Spark driver and different distributed workers. These events help you understand the time-ordered propagation of the exception as faulty code executes across Spark tasks, executors, and stages distributed across the AWS Glue workers.
- Line numbers:** This log stream identifies line 21, which made the call to import the missing Python module that caused the failure; it also identifies line 24, the call to Spark Action collect(), as the last executed line in your script.

	Timestamp	Message
No older events at this moment. Retry		
▶	2022-01-31T06:07:04.750-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Failure Reason: Traceback (most recent call last):
▶	2022-01-31T06:07:04.870-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: -
▶	2022-01-31T06:07:04.888-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: -
▶	2022-01-31T06:07:04.940-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: -
▶	2022-01-31T06:07:04.998-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisStageFailed Failure Reason: Job a...
▶	2022-01-31T06:07:05.044-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisJobFailed Failure Reason: JobFail...
▼	2022-01-31T06:07:05.055-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Root Cause Analysis Result: line 24 in script jobInsightsDemo.py
	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Root Cause Analysis Result: line 24 in script jobInsightsDemo.py.	Copy
▶	2022-01-31T06:07:05.427-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueETLJobExceptionEvent Failure Reason: Traceback (most recent call last):
▼	2022-01-31T06:07:05.430-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Last Executed Line number from script jobInsightsDemo.py: 33
	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Last Executed Line number from script jobInsightsDemo.py: 33	Copy

2. The job-insights-rule-driver log stream:

- Root cause and recommendation:** In addition to the line number and last executed line number for the fault in your script, this log stream shows the root cause analysis and recommendation for you to follow the AWS Glue doc and set up the necessary job parameters in order to use an additional Python module in your AWS Glue job.
- Basis event:** This log stream also shows the Spark exception event that was evaluated with the service-defined rule to infer the root cause and provide a recommendation.

▼	2022-01-31T06:07:05.499-08:00	22/01/31 14:07:05 ERROR Analyzer: 2022-01-31 14:07:05,499 ERROR [pool-2-thread-1] app.GlueJobAnalyzerApp\$ (Logging.scala:logError(9)) - [Glue ...
22/01/31 14:07:05 ERROR Analyzer: 2022-01-31 14:07:05,499 ERROR [pool-2-thread-1] app.GlueJobAnalyzerApp\$ (Logging.scala:logError(9)) - [Glue Insights]		
{	"details": {	
	"time": 1643638025489,	
	"rootCauseAnalysis": "Module that is referenced in Glue job was not found.",	
	"action": "Include all modules used in Glue job, refer documentation on how to include external modules, https://aws.amazon.com/premiumsupport/knowledge-center/glue-version2-external-python-libraries/",	
	},	
	"cause": {	
	"module": "data_multiplier_func",	
	"issue": "ModuleNotFoundError: No module named 'tensorflow'",	
	"fileName": "jobInsightsDemo.py",	
	"lineNoFCode": 24	
	},	
	"basis": [
	{	
	"eventId": 1643638024940,	
	"failureReason": "Traceback (most recent call last):\n File \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(worker.py\\", line 377, in main\nprocess()\nFile \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(worker.py\\", line 372, in process\n self.serializer.dump_stream(func(split_index, iterator),\n outfile)\nFile \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(serializers.py\\", line 345, in dump_stream\n self.serializer.dump_stream(self._batched(iterator), stream)\nFile \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(serializers.py\\", line 141, in dump_stream\n for obj in iterator:\n File \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(serializers.py\\", line 334, in _batched\n for item in iterator:\n File \\"<string>\", line 1, in <lambda>\n File \\"/opt/amazon/spark/python/lib/pyspark.zip/pyspark(worker.py\\", line 85, in wrapper\n return lambda *args, **kwargs\\n File \\"/tmp/jobInsightsDemo.py\\", line 31, in <lambda>\n File \\"/tmp/jobInsightsDemo.py\\", line 24, in data_multiplier_func\nModuleNotFoundError: No module named 'tensorflow'",	
	{"declaringClass": "data_multiplier_func",	
	"methodName": "ModuleNotFoundError: No module named 'tensorflow'",	
	"fileName": "/tmp/jobInsightsDemo.py",	
	"lineNumber": 24	
	},	
	"stackTrace": [
	{	
	"declaringClass": "data_multiplier_func",	
	"methodName": "ModuleNotFoundError: No module named 'tensorflow'",	
	"fileName": "/tmp/jobInsightsDemo.py",	
	"lineNumber": 24	

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 \(p. 332\)](#).

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 \(p. 345\)](#).

Topics

- [Monitoring AWS Glue Using Amazon CloudWatch Metrics \(p. 331\)](#)
- [Setting Up Amazon CloudWatch Alarms on AWS Glue Job Profiles \(p. 345\)](#)
- [Continuous Logging for AWS Glue Jobs \(p. 345\)](#)

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 \(p. 216\)](#).

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>	<p>The number of bytes 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: Bytes</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none">• Bytes read.• Job progress.• JDBC data sources.

Metric	Description
	<ul style="list-style-type: none"> • 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.
<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.

Metric	Description
<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.
<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>

Metric	Description
<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.
<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>

Metric	Description
<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, 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 to avoid hot keys. • Pre-filter data before joins or groupBy operations.
<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.

Metric	Description
glue.driver.BlockManager.disk.diskSpaceUsed	<p>The average of 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> <code>glue.executorId.jvm.heap.used</code> <code>glue.ALL.jvm.heap.used</code>	<p>The number of memory bytes used by the JVM heap for the driver, the executor identified by <i>executorId</i>, 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: 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
<code>glue.driver.s3.filesystem.write_bytes</code> <code>glue.executorId.s3.filesystem.write_bytes</code> <code>glue.ALL.s3.filesystem.write_bytes</code>	<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.
<code>glue.driver.streaming.numRecords</code>	<p>The number of records that are received in a micro-batch. This metric is only available for AWS Glue streaming jobs with AWS Glue version 2.0 and above.</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, Maximum, Minimum, Average, Percentile</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Records read. • Job progress.

Metric	Description
<code>glue.driver.streaming.batchProcessingTime</code>	<p>The time it takes to process the batches in milliseconds. This metric is only available for AWS Glue streaming jobs with AWS Glue version 2.0 and above.</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, Maximum, Minimum, Average, Percentile</p> <p>Unit: Count</p> <p>Can be used to monitor:</p> <ul style="list-style-type: none"> • Job progress. • Script performance.
<code>glue.driver.system.cpuSystemLoad</code> <code>glue.executorId.system.cpuSystemLoad</code> <code>glue.ALL.system.cpuSystemLoad</code>	<p>The fraction of CPU system load used (scale: 0-1) by the driver, an executor identified by <code>executorId</code>, 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 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
JobName	This dimension filters for metrics of all job runs of a specific AWS Glue job.
JobRunID	This dimension filters for metrics of a specific AWS Glue job run by a JobRun ID, or ALL.
Type	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 \(p. 349\)](#).

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

For information about how continuous logging is supported in AWS Glue version 2.0, see [Running Spark ETL Jobs with Reduced Startup Times](#).

You can restrict access to CloudWatch log groups or streams for IAM roles to read the logs. For more details on restricting access, see [Using identity-based policies \(IAM policies\) for CloudWatch Logs](#) in the CloudWatch documentation.

Topics

- [Enabling Continuous Logging for AWS Glue Jobs \(p. 346\)](#)
- [Viewing Continuous Logging for AWS Glue Jobs \(p. 349\)](#)

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 Amazon 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 \(p. 346\)](#)
- [Logging Application-Specific Messages Using the Custom Script Logger \(p. 347\)](#)
- [Enabling the Progress Bar to Show Job Progress \(p. 348\)](#)
- [Security Configuration with Continuous Logging \(p. 348\)](#)

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**, expand the **Monitoring options** section.
5. Select **Continuous logging** to use it for this job.
6. Under **Log filtering**, 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. Expand the **Monitoring options** section.
6. Select **Continuous logging** to use it for this job.
7. Under **Log filtering**, 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. Under the heading **Monitoring options**, choose **Continuous logging**.
5. Under **Log filtering**, 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 \(p. 472\)](#).

```
'--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 Amazon 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 Amazon 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 AWS 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) /  
totalNumOfTasksInThisStage]
```

Security Configuration with Continuous Logging

If a security configuration is enabled for CloudWatch logs, AWS Glue will create a log group named as follows for continuous logs:

```
<Log-Group-Name>-<Security-Configuration-Name>
```

The default and custom log groups will be as follows:

- The default continuous log group will be `/aws-glue/jobs/logs-v2-<Security-Configuration-Name>`
- The custom continuous log group will be `<custom-log-group-name>-<Security-Configuration-Name>`

You need to add the `logs:AssociateKmsKey` to your IAM role permissions, if you enable a security configuration with CloudWatch Logs. If that permission is not included, continuous logging will be disabled. Also, to configure the encryption for the CloudWatch Logs, follow the instructions at [Encrypt Log Data in CloudWatch Logs Using AWS Key Management Service](#) in the *Amazon CloudWatch Logs User Guide*.

For more information on creating security configurations, see [Working with Security Configurations on the AWS Glue Console \(p. 53\)](#).

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 for Spark Jobs \(p. 198\)](#) or [Special Parameters Used by AWS Glue \(p. 472\)](#).
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 \(p. 350\)](#)
- [Debugging Demanding Stages and Straggler Tasks \(p. 357\)](#)
- [Monitoring the Progress of Multiple Jobs \(p. 361\)](#)
- [Monitoring for DPU Capacity Planning \(p. 365\)](#)

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 \(p. 350\)](#)
- [Debugging an Executor OOM Exception \(p. 353\)](#)

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 \(p. 340\)](#) crosses the safe threshold of 50 percent usage quickly. On the other hand, the [average memory usage \(p. 340\)](#) 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
jr_5731b225...	-	Failed	Command failed with exit code 1	1 mins	2880 mins

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

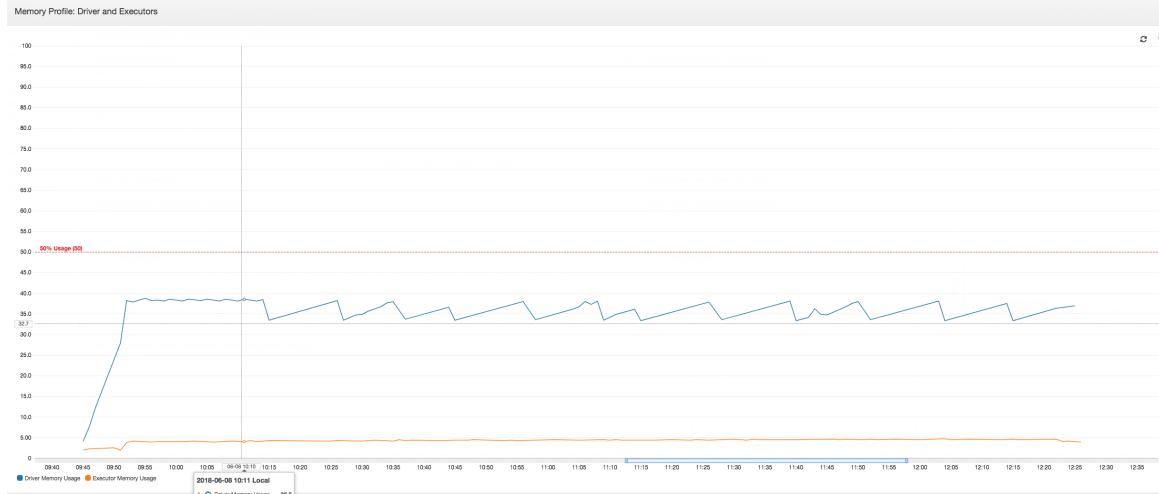
You can fix the processing of the multiple files by using the *grouping* feature in AWS Glue. Grouping is automatically enabled when you use dynamic frames and when the input dataset has a large number of files (more than 50,000). Grouping allows you to coalesce multiple files together into a group, 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 \(p. 511\)](#).

To check the memory profile of the AWS Glue job, profile the following code with grouping enabled:

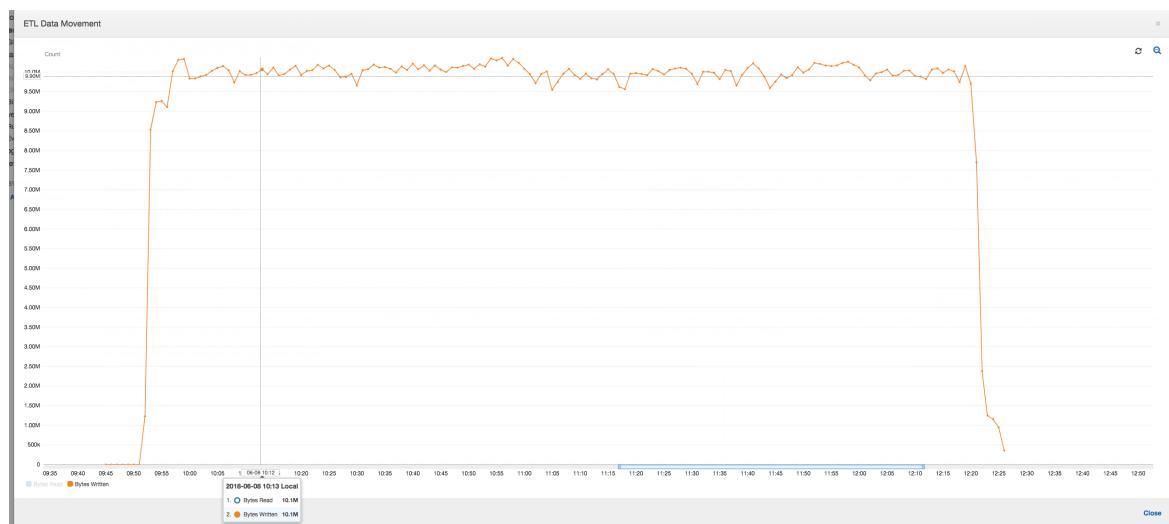
```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"], "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3", connection_options = {"path": output_path}, format = "parquet", transformation_ctx = "datasink")
```

You can monitor the memory profile and the ETL data movement in the AWS Glue job profile.

The driver runs 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 \(p. 342\)](#) and [written \(p. 343\)](#) 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.



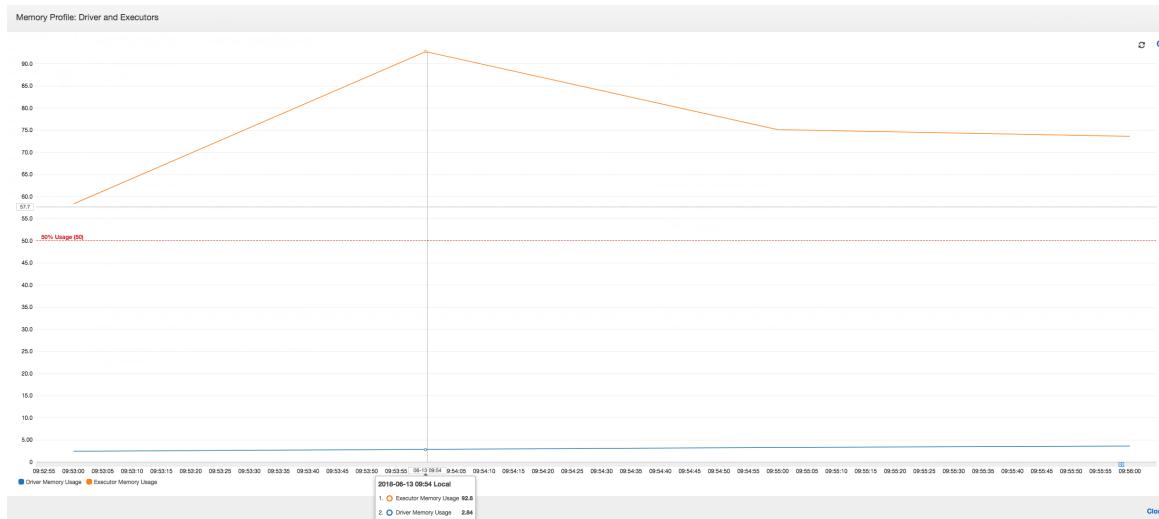
Debugging an Executor OOM Exception

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.

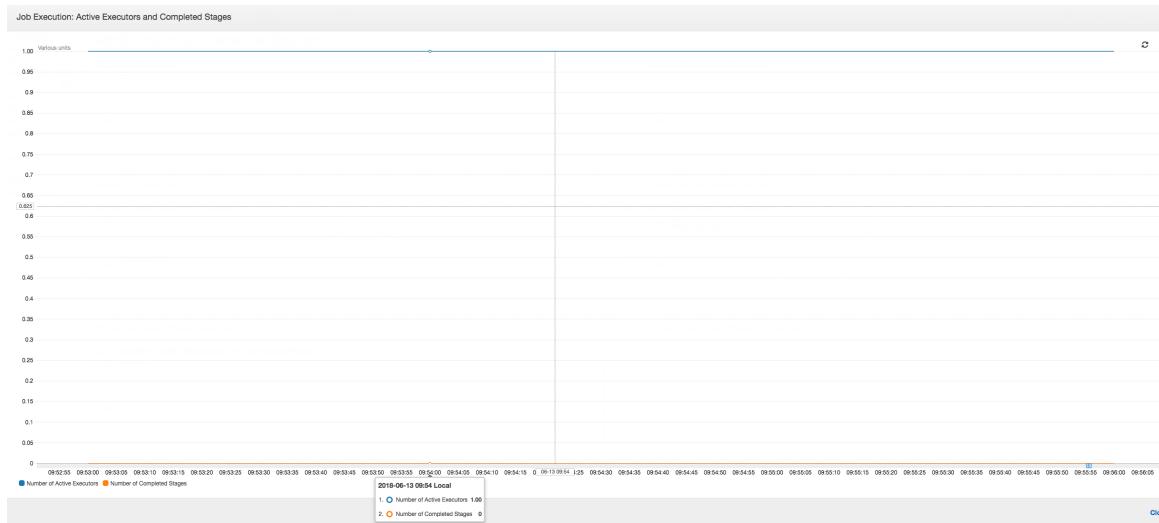
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

Visualize the Profiled Metrics on the AWS Glue Console

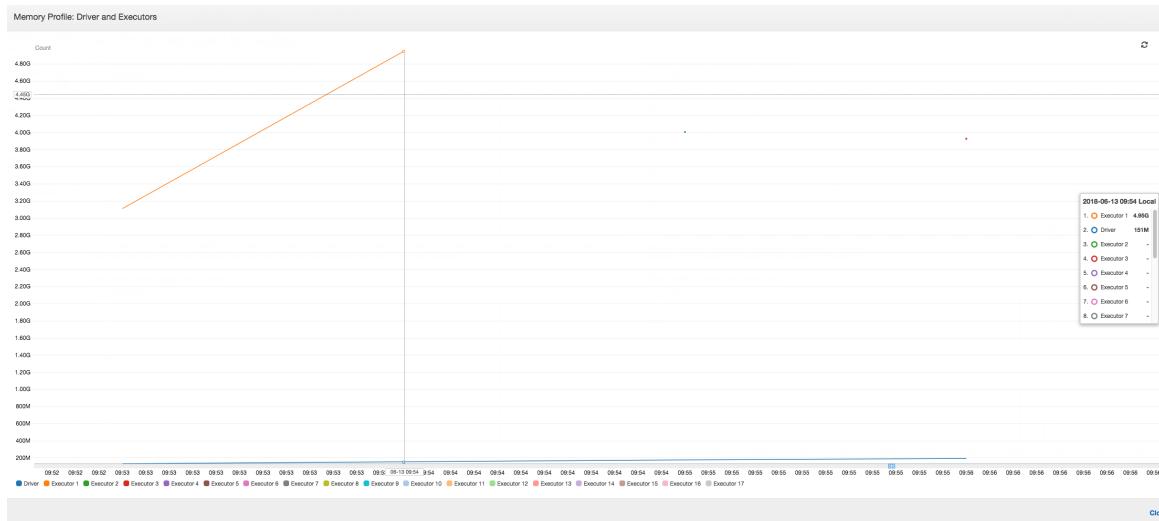
If the slope of the memory usage graph is positive and crosses 50 percent, then if the job fails before the next metric is emitted, then memory exhaustion is a good candidate for the cause. The following graph shows that within a minute of execution, the [average memory usage \(p. 340\)](#) across all executors spikes up quickly above 50 percent. The usage reaches up to 92 percent and the container running the executor is stopped by Apache Hadoop YARN.



As the following graph shows, there is always a [single executor \(p. 338\)](#) running until the job fails. This is because a new executor is launched to replace the stopped 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 \(p. 341\)](#) 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.

History	Details	Script	Metrics
Showing: 1 - 15			
Run ID	Retry attempt	Run status	Error
y_467d27235b5d34900d9e26...	-	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.
y_d10a929898758881528db...	-	Succeeded	
y_dcd807823082bef919162...	-	Succeeded	
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.

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 stopped 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

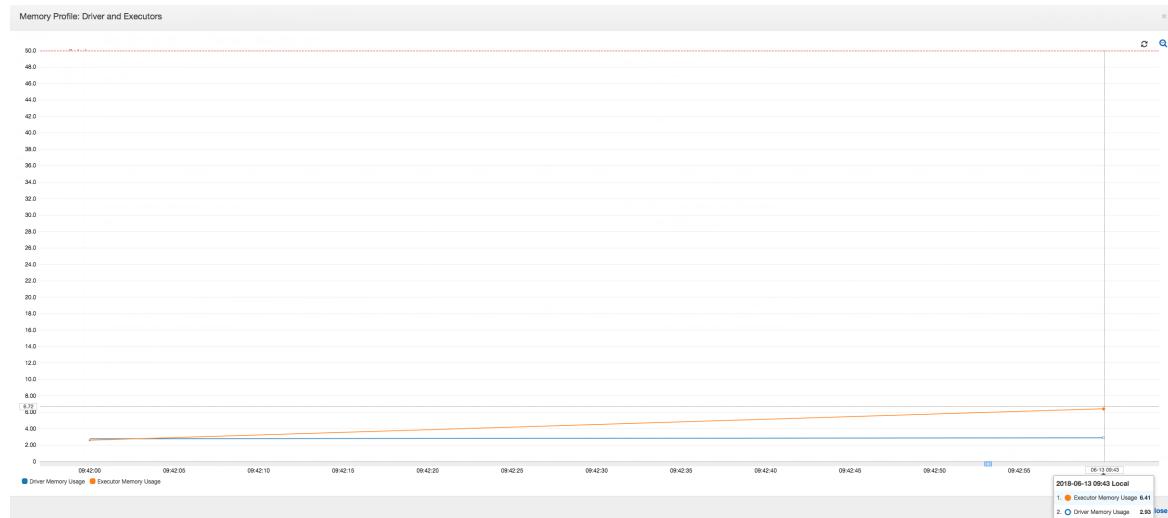
The executor ran out of memory while reading the JDBC table because the default configuration for the Spark JDBC fetch size is zero. This means that the JDBC driver on the Spark executor tries to fetch the 34 million rows from the database together and cache them, even though Spark streams through the 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 \(p. 340\)](#) 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)
```

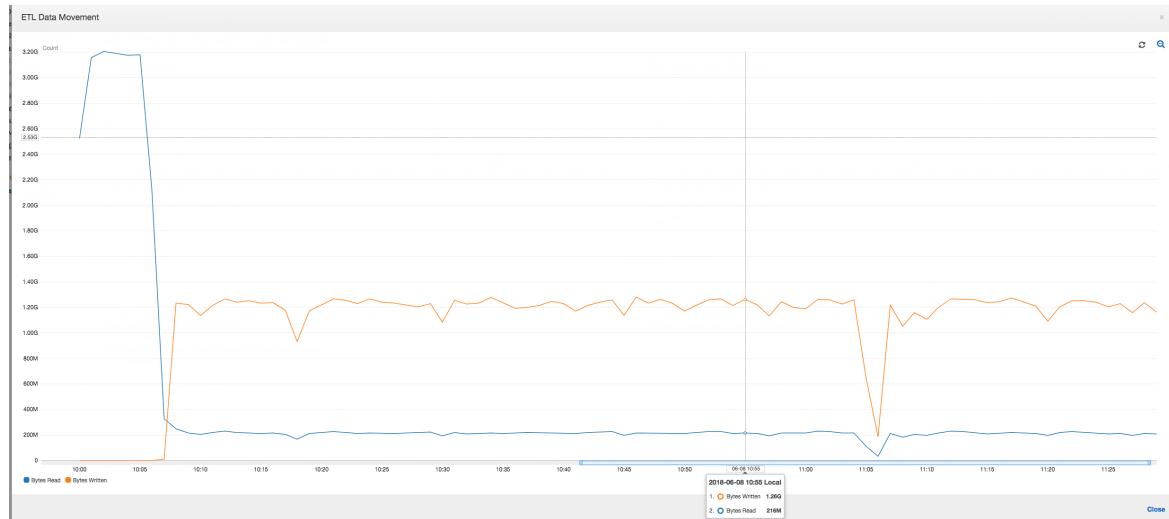
Visualize the Profiled Metrics on the AWS Glue Console

You can profile your job to examine four different sets of metrics:

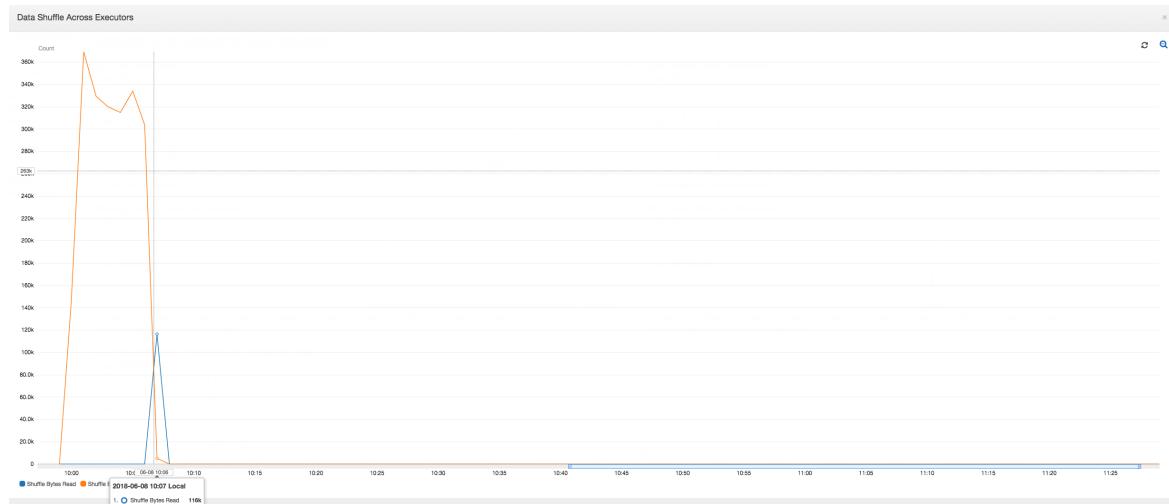
- ETL data movement
- Data shuffle across executors

- Job execution
- Memory profile

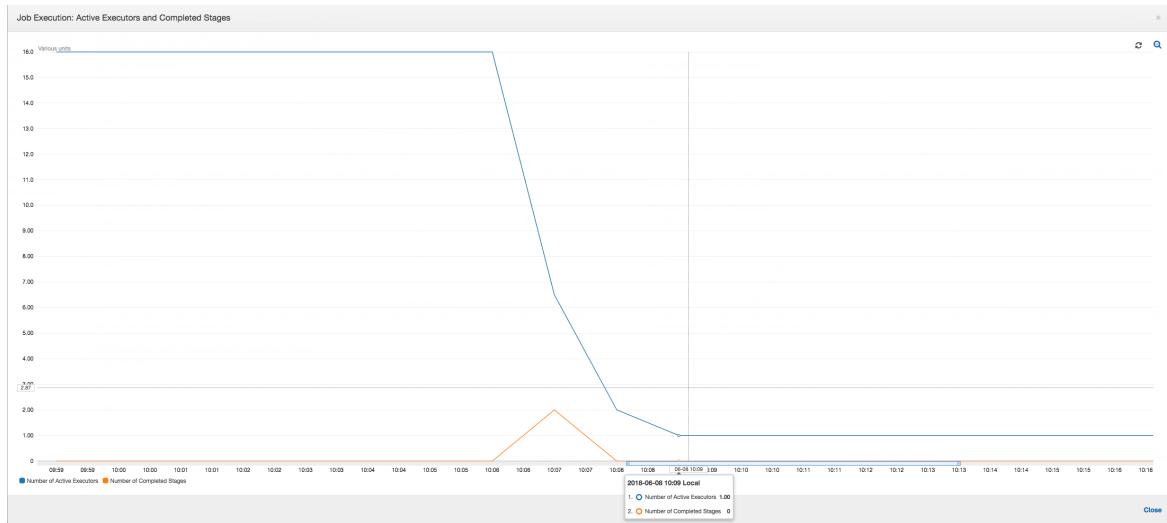
ETL data movement: In the **ETL Data Movement** profile, the bytes are [read \(p. 342\)](#) fairly quickly by all the executors in the first stage that completes within the first six minutes. However, the total job execution time is around one hour, mostly consisting of the data [writes \(p. 343\)](#).



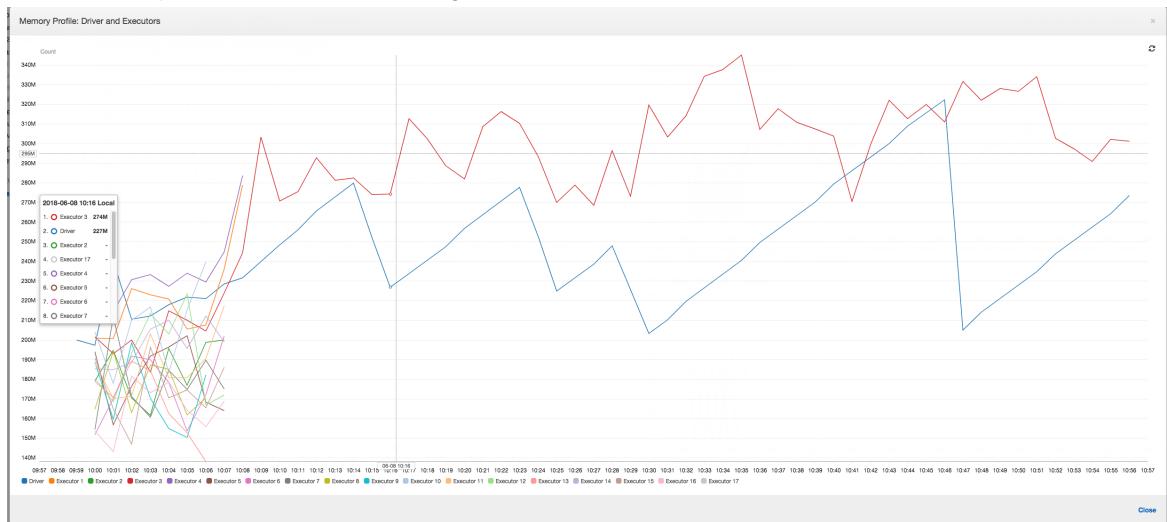
Data shuffle across executors: The number of bytes [read \(p. 336\)](#) and [written \(p. 336\)](#) during shuffling also shows a spike before Stage 2 ends, as indicated by the **Job Execution** and **Data Shuffle** metrics. After the data shuffles from all executors, the reads and writes proceed from 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 \(p. 341\)](#) 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.



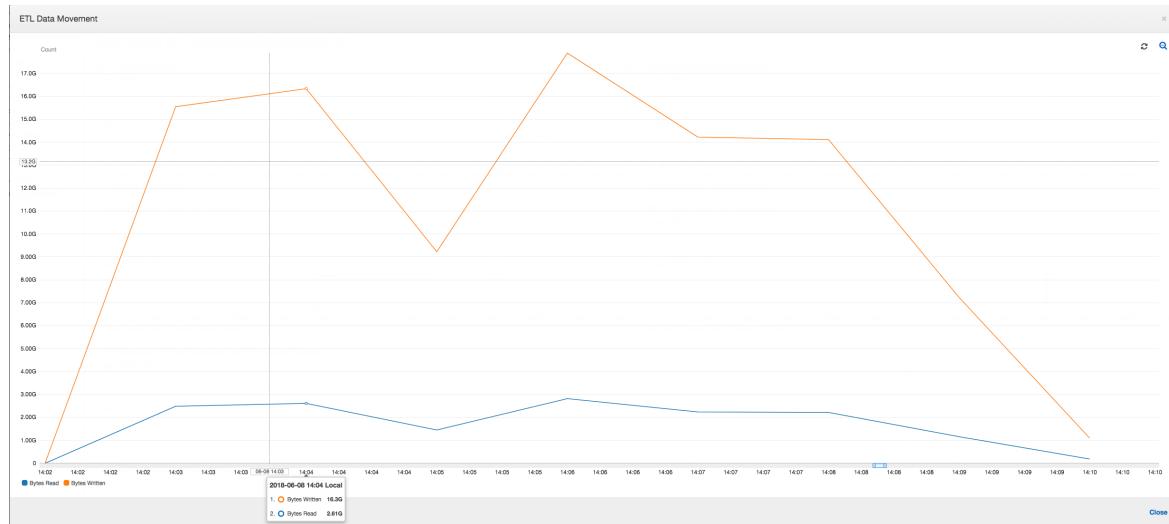
Fix Straggling Executors Using Grouping

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 \(p. 511\)](#).

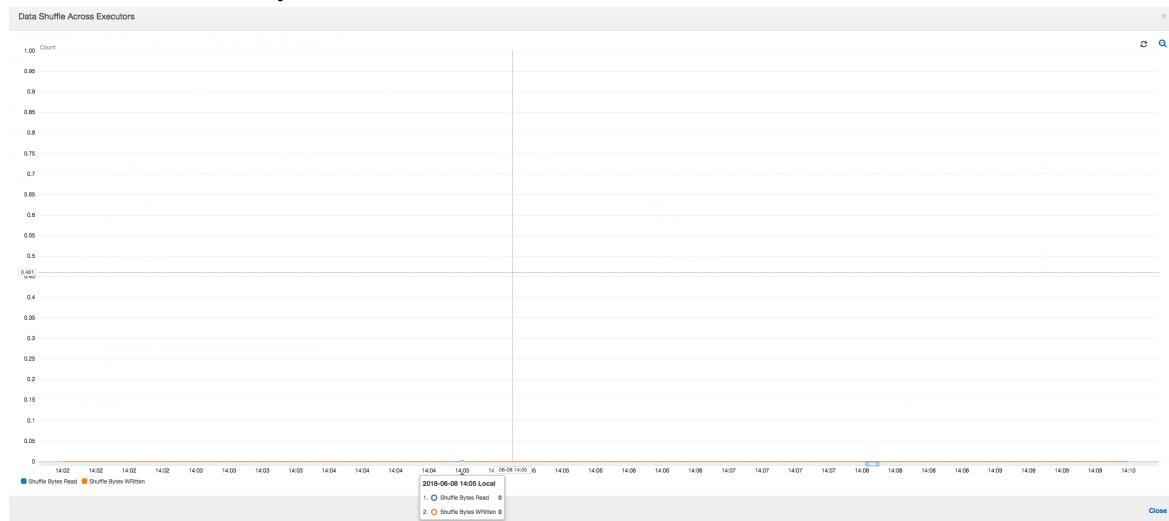
To check the ETL data movements in the AWS Glue job, profile the following code with grouping enabled:

```
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.



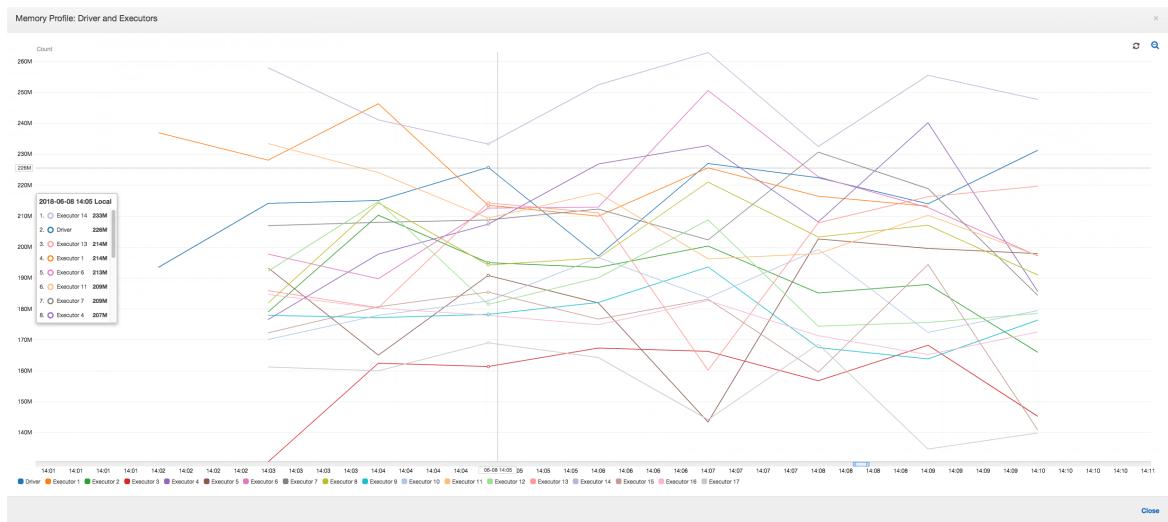
Data shuffle across executors: As the input files are coalesced during the reads using the grouping feature, there is no costly data shuffle after the data reads.



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 \(p. 341\)](#) 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 \(p. 362\)](#)
- [Visualize the Profiled Metrics on the AWS Glue Console \(p. 362\)](#)
- [Fix the Processing of Files \(p. 364\)](#)

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.					
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:

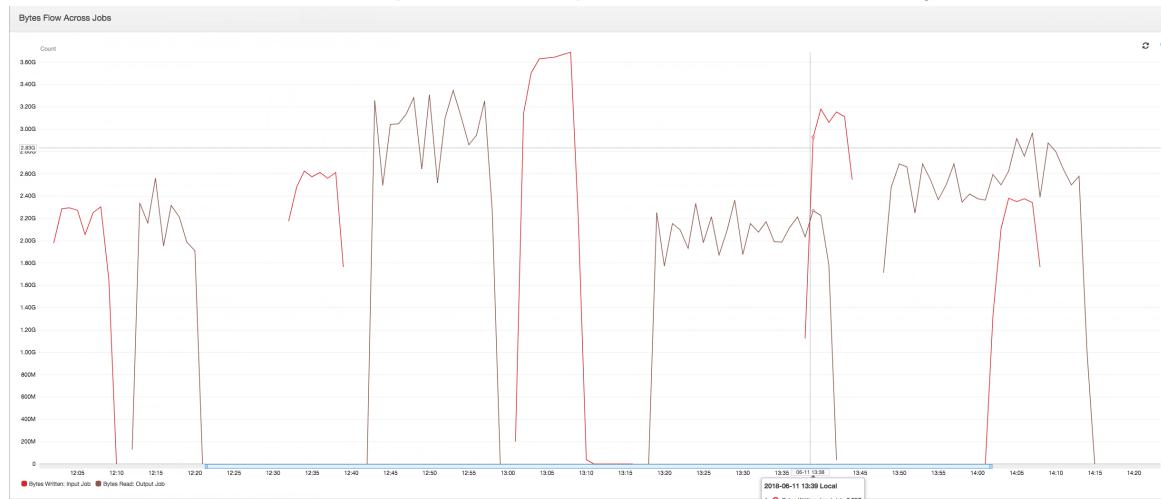
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
    connection_options = {"paths": ["s3://input_path"],
    "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = 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
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:30 PM UT...	11 June 2018 2:40 PM UT...
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:30 PM UT...	11 June 2018 2:10 PM UT...
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		7 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:30 PM UT...	11 June 2018 1:46 PM UT...
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		15 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:00 PM UT...	11 June 2018 1:16 PM UT...
<hr/>											

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
jrcce47fb1a561051fdccae95e...	-	Failed	Max conc...	Logs	Error logs	0 secs	2880 mins		e2e-bookmark-output	11 June 2018 2:11 PM UT...	
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		27 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:47 PM UT...	11 June 2018 2:15 PM UT...
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		24 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:17 PM UT...	11 June 2018 1:43 PM UT...
jrcce47fb1a561051fdccae95e...	-	Succeeded		Logs		17 mins	2880 mins		e2e-bookmark-output	11 June 2018 12:41 PM UT...	11 June 2018 12:59 PM UT...
<hr/>											

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

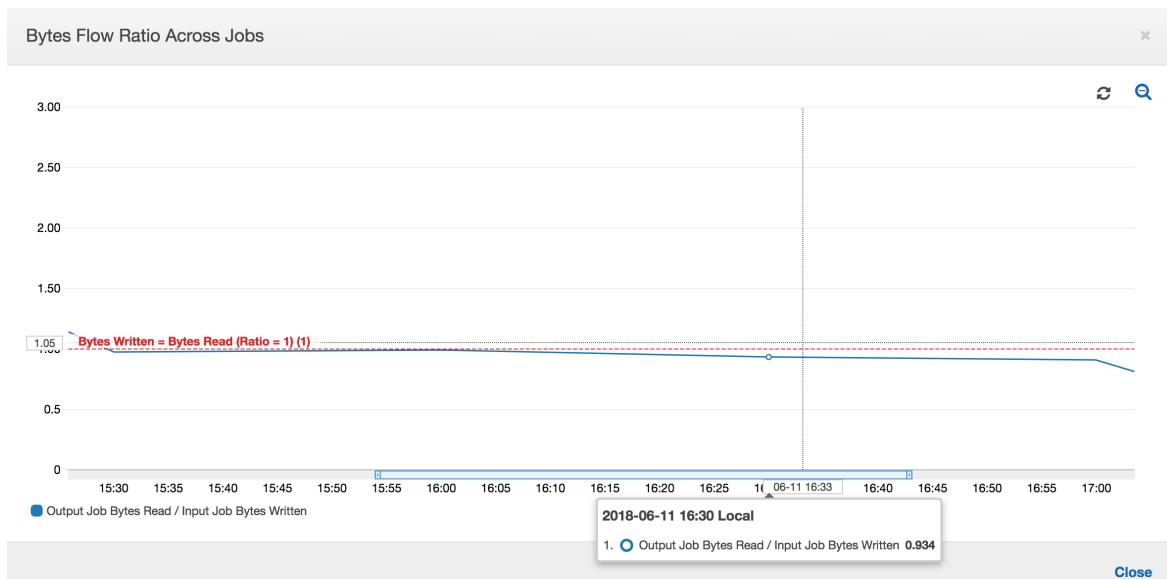
You should ensure that the Output job processes only the files that haven't been processed by previous job runs of the Output job. To do this, enable job bookmarks and set the transformation context in the Output job, as follows:

```
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")
```

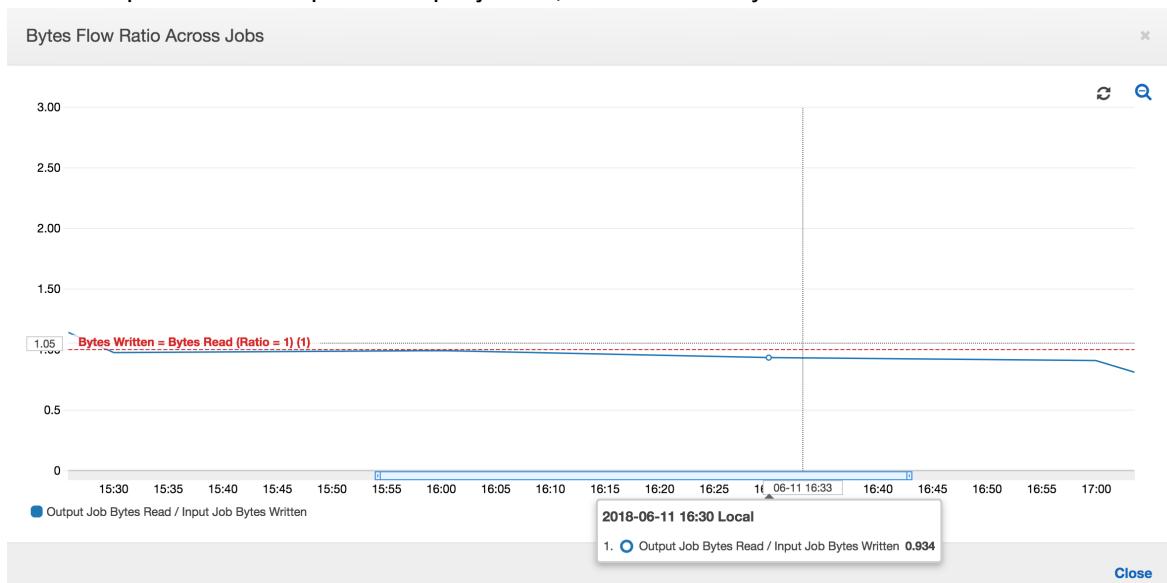
With job bookmarks enabled, the Output job doesn't reprocess the data in the staging location from all the previous job runs of the Input job. In the following image showing the data read and written, the area under the brown curve is fairly consistent and similar with the red curves.



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.

Note

This page is only applicable to AWS Glue versions 0.9 and 1.0. For more information on AWS Glue version 2.0 limitations on these metrics, see [AWS Glue Release Notes \(p. 1006\)](#).

Topics

- [Profiled Code \(p. 366\)](#)
- [Visualize the Profiled Metrics on the AWS Glue Console \(p. 366\)](#)
- [Determine the Optimal DPU Capacity \(p. 368\)](#)

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 run 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")
```

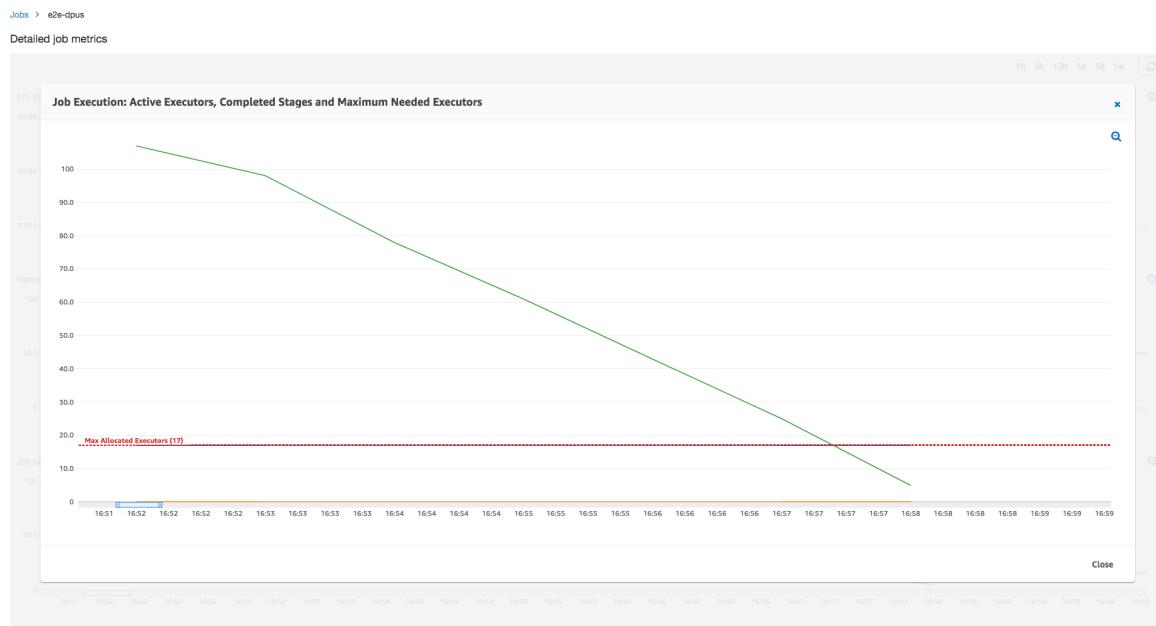
Visualize the Profiled Metrics on the AWS Glue Console

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 \(p. 338\)](#), the [number of completed stages \(p. 333\)](#), and the [number of maximum needed executors \(p. 339\)](#).

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.

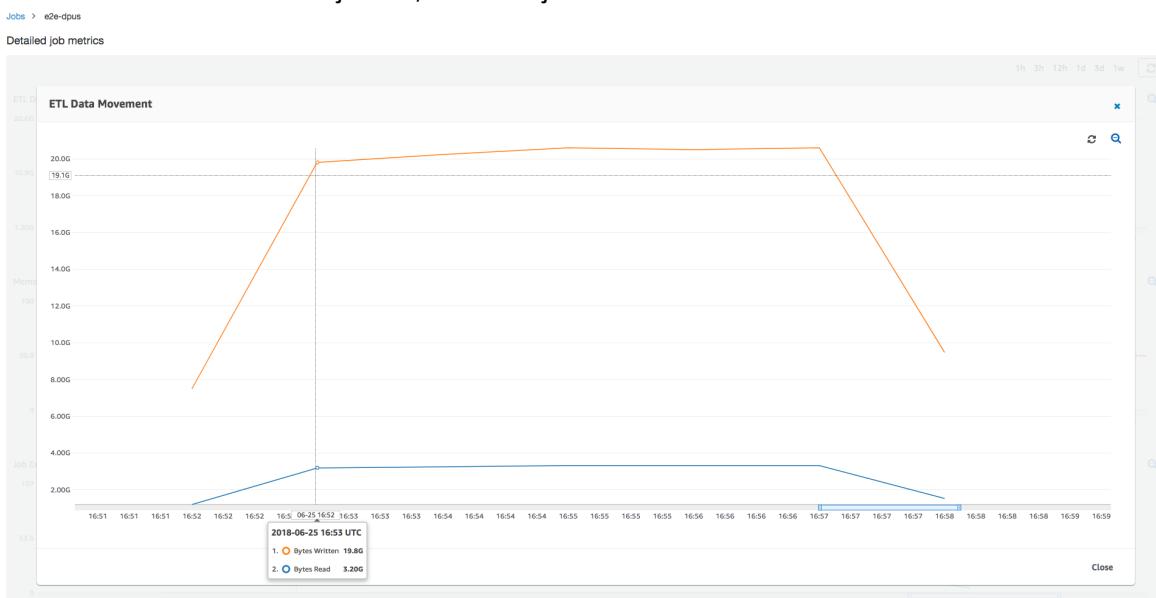
In contrast, the number of actively running executors measures how many executors are running active Apache Spark tasks. As the job progresses, the maximum needed executors can change and typically goes down towards the end of the job as the pending task queue diminishes.

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 management. Nine DPUs run two executors each and one executor is reserved for the Spark driver. The Spark driver runs inside the primary application. 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**.

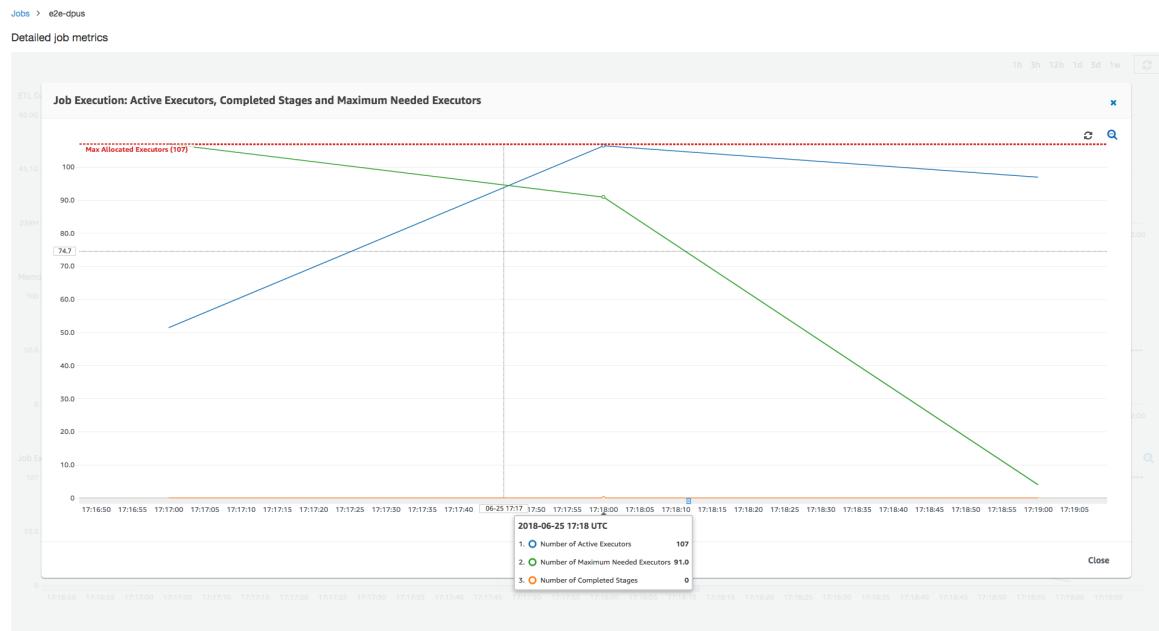


Looking at the Amazon S3 bytes [read \(p. 342\)](#) and [written \(p. 343\)](#), notice that the job spends all six minutes streaming in data from Amazon S3 and writing it out in parallel. All the cores on the allocated

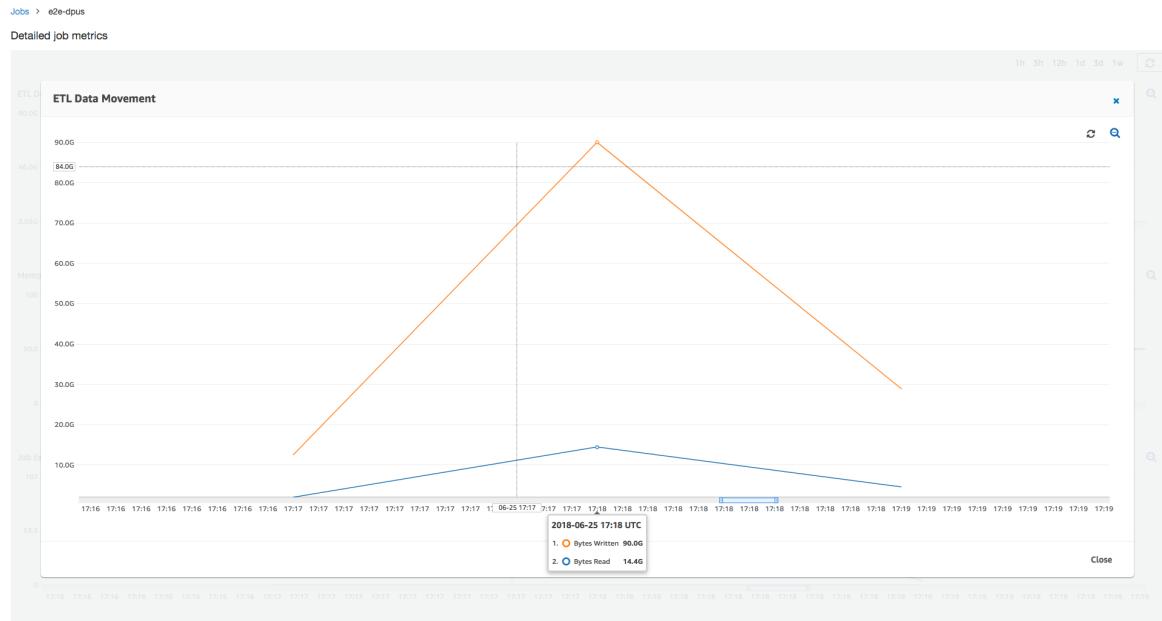
DPU are reading and writing to Amazon S3. The maximum number of needed executors being 107 also matches the number of files in the input Amazon S3 path—428. Each executor can launch four Spark tasks to process four input files (JSON gzipped).

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.

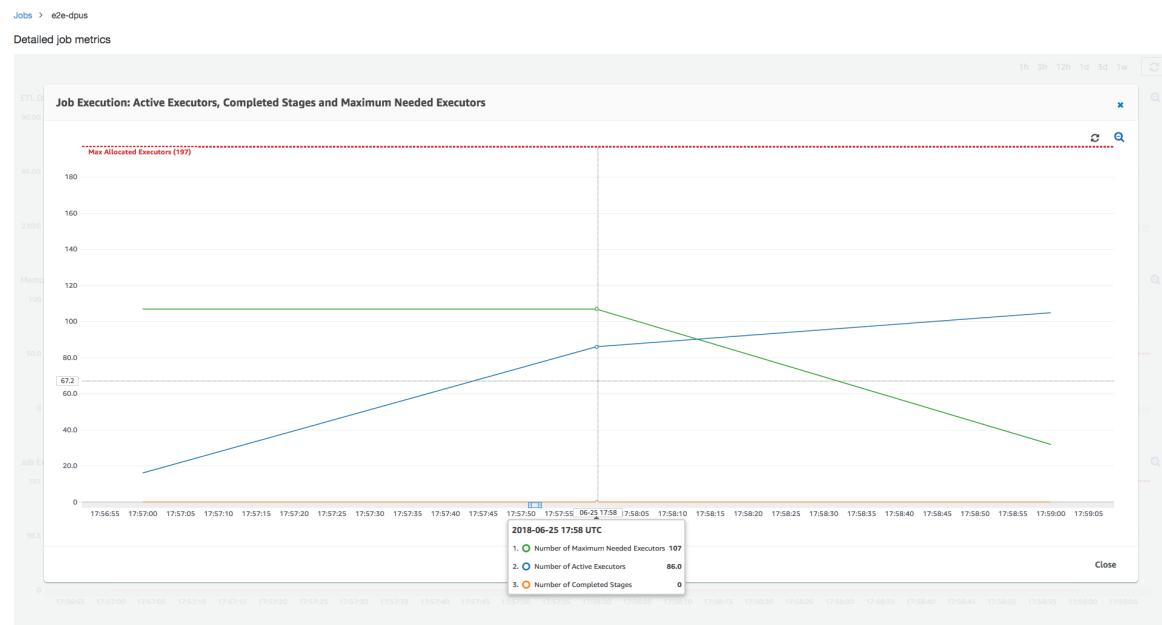


As the above image shows, the total number of active executors reaches the maximum allocated—107 executors. Similarly, the maximum needed executors is never above the maximum allocated executors. The maximum needed executors number is computed from the actively running and pending task counts, so it 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.



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 \(p. 701\)](#). 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"
}

```

AWS Glue Job Run Statuses

You can view the status of an AWS Glue extract, transform, and load (ETL) job while it is running or after it has stopped. You can view the status using the AWS Glue console, the AWS Command Line Interface (AWS CLI), or the [GetJobRun action](#) in the AWS Glue API.

Possible job run statuses are STARTING, RUNNING, STOPPING, STOPPED, SUCCEEDED, FAILED, ERROR, and TIMEOUT. The following table lists the statuses that indicate abnormal job termination.

Job run status	Description
FAILED	The job exceeded its maximum allowed concurrent runs, or terminated with an unknown exit code.
ERROR	A workflow, schedule trigger, or event trigger attempted to run a deleted job.
TIMEOUT	The job run time exceeded its specified timeout value.

Performing Complex ETL Activities Using Blueprints and Workflows in AWS Glue

Some of your organization's complex extract, transform, and load (ETL) processes might best be implemented by using multiple, dependent AWS Glue jobs and crawlers. Using AWS Glue *workflows*, you can design a complex multi-job, multi-crawler ETL process that AWS Glue can run and track as single entity. After you create a workflow and specify the jobs, crawlers, and triggers in the workflow, you can run the workflow on demand or on a schedule.

Your organization might have a set of similar ETL use cases that could benefit from being able to parametrize a single workflow to handle them all. To address this need, AWS Glue enables you to define *blueprints*, which you can use to generate workflows. A blueprint accepts parameters, so that from a single blueprint, a data analyst can create different workflows to handle similar ETL use cases. After you create a blueprint, you can reuse it for different departments, teams, and projects.

Topics

- [Overview of Workflows in AWS Glue \(p. 373\)](#)
- [Overview of Blueprints in AWS Glue \(p. 376\)](#)
- [Developing Blueprints in AWS Glue \(p. 378\)](#)
- [Registering a Blueprint in AWS Glue \(p. 394\)](#)
- [Viewing Blueprints in AWS Glue \(p. 396\)](#)
- [Updating a Blueprint in AWS Glue \(p. 397\)](#)
- [Creating a Workflow from a Blueprint in AWS Glue \(p. 398\)](#)
- [Viewing Blueprint Runs in AWS Glue \(p. 400\)](#)
- [Creating and Building Out a Workflow Manually in AWS Glue \(p. 400\)](#)
- [Starting an AWS Glue Workflow with an Amazon EventBridge Event \(p. 404\)](#)
- [Viewing the EventBridge Events That Started a Workflow \(p. 408\)](#)
- [Running and Monitoring a Workflow in AWS Glue \(p. 409\)](#)
- [Stopping a Workflow Run \(p. 411\)](#)
- [Repairing and Resuming a Workflow Run \(p. 411\)](#)
- [Getting and Setting Workflow Run Properties in AWS Glue \(p. 416\)](#)
- [Querying Workflows Using the AWS Glue API \(p. 417\)](#)
- [Blueprint and Workflow Restrictions in AWS Glue \(p. 420\)](#)
- [Troubleshooting Blueprint errors in AWS Glue \(p. 420\)](#)
- [Permissions for Personas and Roles for AWS Glue Blueprints \(p. 424\)](#)

Overview of Workflows in AWS Glue

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 jobs and crawlers. As a workflow runs each component, it records execution progress and status. This provides 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.

You can create a workflow from an AWS Glue blueprint, or you can manually build a workflow a component at a time using the AWS Management Console or the AWS Glue API. For more information about blueprints, see the section called “[Overview of Blueprints](#)” (p. 376).

Triggers within workflows can start both jobs and crawlers and can be fired when jobs or crawlers complete. By using triggers, you can create large chains of interdependent jobs and crawlers. In addition to triggers within a workflow that define job and crawler dependencies, each workflow has a *start trigger*. There are three types of start triggers:

- **Schedule** – The workflow is started according to a schedule that you define. The schedule can be daily, weekly, monthly, and so on, or can be a custom schedule based on a cron expression.
- **On demand** – The workflow is started manually from the AWS Glue console, API, or AWS CLI.
- **EventBridge event** – The workflow is started upon the occurrence of a single Amazon EventBridge event or a batch of Amazon EventBridge events. With this trigger type, AWS Glue can be an event consumer in an event-driven architecture. Any EventBridge event type can start a workflow. A common use case is the arrival of a new object in an Amazon S3 bucket (the S3 PutObject operation).

Starting a workflow with a batch of events means waiting until a specified number of events have been received or until a specified amount of time has passed. When you create the EventBridge event trigger, you can optionally specify batch conditions. If you specify batch conditions, you must specify the batch size (number of events), and can optionally specify a batch window (number of seconds). The default and maximum batch window is 900 seconds (15 minutes). The batch condition that is met first starts the workflow. The batch window starts when the first event arrives. If you don't specify batch conditions when creating a trigger, the batch size defaults to 1.

When the workflow starts, the batch conditions are reset and the event trigger begins watching for the next batch condition to be met to start the workflow again.

The following table shows how batch size and batch window operate together to trigger a workflow.

Batch size	Batch window	Resulting triggering condition
10		The workflow is triggered upon the arrival of 10 EventBridge events, or 15 minutes after the arrival of the first event, whichever occurs first. (If windows size isn't specified, it defaults to 15 minutes.)
10	2 mins	The workflow is triggered upon the arrival of 10 EventBridge events, or 2 minutes after the arrival of the first event, whichever occurs first.
1		The workflow is triggered upon the arrival of the first event. Window size is irrelevant. The batch size defaults to 1 if you don't specify batch conditions when you create the EventBridge event trigger.

The `GetWorkflowRun` API operation returns the batch condition that triggered the workflow.

Regardless of how a workflow is started, you can specify the maximum number of concurrent workflow runs when you create the workflow.

If an event or batch of events starts a workflow run that eventually fails, that event or batch of events is no longer considered for starting a workflow run. A new workflow run is started only when the next event or batch of events arrives.

Important

Limit the total number of jobs, crawlers, and triggers within a workflow to 100 or less. If you include more than 100, you might get errors when trying to resume or stop workflow runs.

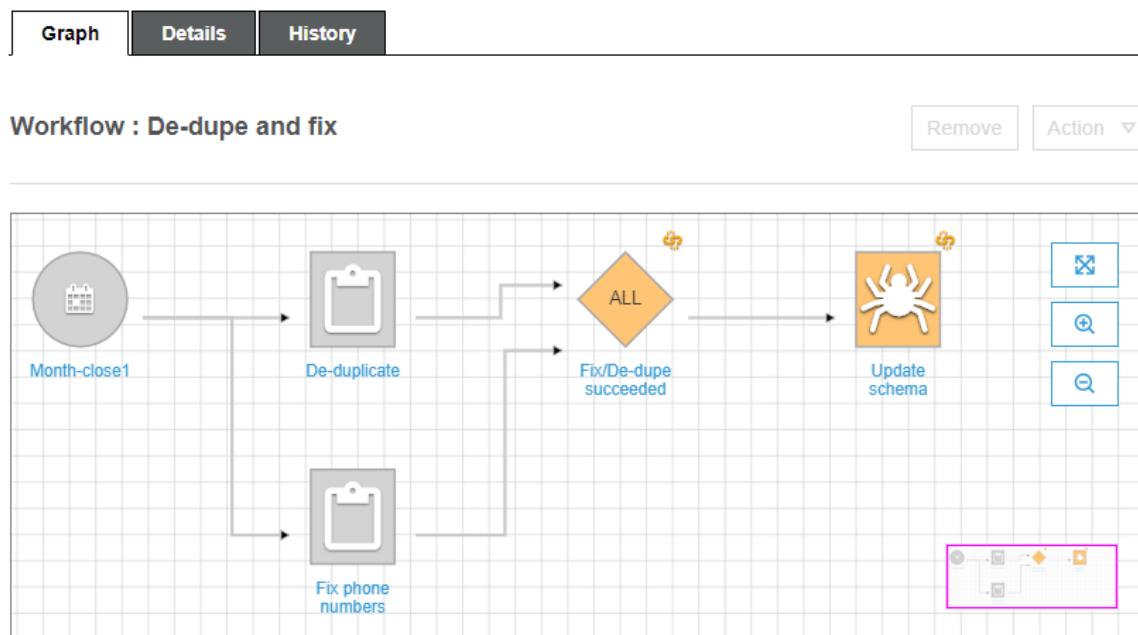
A workflow run will not be started if it would exceed the concurrency limit set for the workflow, even though the event condition is met. It's advisable to adjust workflow concurrency limits based on the expected event volume. AWS Glue does not retry workflow runs that fail due to exceeded concurrency limits. Likewise, it's advisable to adjust concurrency limits for jobs and crawlers within workflows based on expected event volume.

Workflow Run Properties

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.

Workflow Graph

The following image shows the graph of a very basic workflow on the AWS Glue console. Your workflow could have dozens of components.



This workflow is started by a schedule trigger, `Month-close1`, which starts two jobs, `De-duplicate` and `Fix phone numbers`. Upon successful completion of both jobs, an event trigger, `Fix/De-dupe succeeded`, starts a crawler, `Update schema`.

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 runtime 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 \(p. 417\)](#).

See Also

- the section called “[Creating a Workflow from a Blueprint](#)” (p. 398)
- the section called “[Creating and Building Out a Workflow Manually](#)” (p. 400)
- [Workflows \(p. 921\)](#) (for the workflows API)

Overview of Blueprints in AWS Glue

AWS Glue blueprints provide a way to create and share AWS Glue workflows. When there is a complex ETL process that could be used for similar use cases, rather than creating an AWS Glue workflow for each use case, you can create a single blueprint.

The blueprint specifies the jobs and crawlers to include in a workflow, and specifies parameters that the workflow user supplies when they run the blueprint to create a workflow. The use of parameters enables a single blueprint to generate workflows for the various similar use cases. For more information about workflows, see [Overview of Workflows in AWS Glue \(p. 373\)](#).

The following are example use cases for blueprints:

- You want to partition an existing dataset. The input parameters to the blueprint are Amazon Simple Storage Service (Amazon S3) source and target paths and a list of partition columns.
- You want to snapshot an Amazon DynamoDB table into a SQL data store like Amazon Redshift. The input parameters to the blueprint are the DynamoDB table name and an AWS Glue connection, which designates an Amazon Redshift cluster and destination database.
- You want to convert CSV data in multiple Amazon S3 paths to Parquet. You want the AWS Glue workflow to include a separate crawler and job for each path. The input parameters are the destination database in the AWS Glue Data Catalog and a comma-delimited list of Amazon S3 paths. Note that in this case, the number of crawlers and jobs that the workflow creates is variable.

Blueprint Components

A blueprint is a ZIP archive that contains the following components:

- A Python layout generator script

Contains a function that specifies the workflow *layout*—the crawlers and jobs to create for the workflow, the job and crawler properties, and the dependencies between the jobs and crawlers. The function accepts blueprint parameters and returns a workflow structure (JSON object) that AWS Glue uses to generate the workflow. Because you use a Python script to generate the workflow, you can add your own logic that is suitable for your use cases.

- A configuration file

Specifies the fully qualified name of the Python function that generates the workflow layout. Also specifies the names, data types, and other properties of all blueprint parameters used by the script.

- (Optional) ETL scripts and supporting files

As an advanced use case, you can parameterize the location of the ETL scripts that your jobs use. You can include job script files in the ZIP archive and specify a blueprint parameter for an Amazon S3 location where the scripts are to be copied to. The layout generator script can copy the ETL scripts to the designated location and specify that location as the job script location property. You can also include any libraries or other supporting files, provided that your script handles them.

Blueprint

Python Script

```
import sys
import os
import json
def generate_layout(use
    etl_job = Job(Name="
```

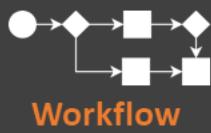
Config File

```
"layoutGenerator": "My
"parameterSpec" : {
"WorkflowName" : {
"type": "String"
"collection": false
```

Blueprint runs

When you create a workflow from a blueprint, AWS Glue runs the blueprint, which starts an asynchronous process to create the workflow and the jobs, crawlers, and triggers that the workflow encapsulates. AWS Glue uses the blueprint run to orchestrate the creation of the workflow and its components. You view the status of the creation process by viewing the blueprint run status. The blueprint run also stores the values that you supplied for the blueprint parameters.

Blueprint Run



Workflow

Name: MyWF
Role:CreateBP
Sources: 20

Parameter Values

You can view blueprint runs using the AWS Glue console or AWS Command Line Interface (AWS CLI). When viewing or troubleshooting a workflow, you can always return to the blueprint run to view the blueprint parameter values that were used to create the workflow.

Lifecycle of a Blueprint

Blueprints are developed, tested, registered with AWS Glue, and run to create workflows. There are typically three personas involved in the blueprint lifecycle.

Persona	Tasks
AWS Glue developer	<ul style="list-style-type: none">Writes the workflow layout script and creates the configuration file.Tests the blueprint locally using libraries provided by the AWS Glue service.Creates a ZIP archive of the script, configuration file, and supporting files and publishes the archive to a location in Amazon S3.Adds a bucket policy to the Amazon S3 bucket that grants read permissions on bucket objects to the AWS Glue administrator's AWS account.

Persona	Tasks
	<ul style="list-style-type: none">Grants IAM read permissions on the ZIP archive in Amazon S3 to the AWS Glue administrator.
AWS Glue administrator	<ul style="list-style-type: none"><i>Registers</i> the blueprint with AWS Glue. AWS Glue makes a copy of the ZIP archive into a reserved Amazon S3 location.Grants IAM permissions on the blueprint to data analysts.
Data analyst	<ul style="list-style-type: none">Runs the blueprint to create a workflow, and provides blueprint parameter values. Checks the blueprint run status to ensure that the workflow and workflow components were successfully generated.Runs and troubleshoots the workflow. Before running the workflow, can verify the workflow by viewing the workflow design graph on the AWS Glue console.

See Also

- [Developing Blueprints in AWS Glue \(p. 378\)](#)
- [Creating a Workflow from a Blueprint in AWS Glue \(p. 398\)](#)
- [Permissions for Personas and Roles for AWS Glue Blueprints \(p. 424\)](#)

Developing Blueprints in AWS Glue

As an AWS Glue developer, you can create and publish blueprints that data analysts can use to generate workflows.

Topics

- [Overview of Developing Blueprints \(p. 378\)](#)
- [Prerequisites for Developing Blueprints \(p. 379\)](#)
- [Writing the Blueprint Code \(p. 382\)](#)
- [Sample Blueprint Project \(p. 386\)](#)
- [Testing a Blueprint \(p. 389\)](#)
- [Publishing a Blueprint \(p. 390\)](#)
- [AWS Glue Blueprint Classes Reference \(p. 391\)](#)
- [Blueprint Samples \(p. 394\)](#)

See Also

- [Overview of Blueprints in AWS Glue \(p. 376\)](#)

Overview of Developing Blueprints

The first step in your development process is to identify a common use case that would benefit from a blueprint. A typical use case involves a recurring ETL problem that you believe should be solved in a general manner. Next, design a blueprint that implements the generalized use case, and define the blueprint input parameters that together can define a specific use case from the generalized use case.

A blueprint consists of a project that contains a blueprint parameter configuration file and a script that defines the *layout* of the workflow to generate. The layout defines the jobs and crawlers (or *entities* in blueprint script terminology) to create.

You do not directly specify any triggers in the layout script. Instead you write code to specify the dependencies between the jobs and crawlers that the script creates. AWS Glue generates the triggers based on your dependency specifications. The output of the layout script is a workflow object, which contains specifications for all workflow entities.

You build your workflow object using the following AWS Glue blueprint libraries:

- `awsglue.blueprint.base_resource` – A library of base resources used by the libraries.
- `awsglue.blueprint.workflow` – A library for defining a `Workflow` class.
- `awsglue.blueprint.job` – A library for defining a `Job` class.
- `awsglue.blueprint.crawler` – A library for defining a `Crawler` class.

The only other libraries that are supported for layout generation are those libraries that are available for the Python shell.

Before publishing your blueprint, you can use methods defined in the blueprint libraries to test the blueprint locally.

When you're ready to make the blueprint available to data analysts, you package the script, the parameter configuration file, and any supporting files, such as additional scripts and libraries, into a single deployable asset. You then upload the asset to Amazon S3 and ask an administrator to register it with AWS Glue.

For information about more sample blueprint projects, see [Sample Blueprint Project \(p. 386\)](#) and [Blueprint Samples \(p. 394\)](#).

Prerequisites for Developing Blueprints

To develop blueprints, you should be familiar with using AWS Glue and writing scripts for Apache Spark ETL jobs or Python shell jobs. In addition, you must complete the following setup tasks.

- Download four AWS Python libraries to use in your blueprint layout scripts.
- Set up the AWS SDKs.
- Set up the AWS CLI.

Download the Python Libraries

Download the following libraries from GitHub, and install them into your project:

- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/base_resource.py
- <https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/workflow.py>
- <https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/crawler.py>
- <https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/job.py>

Set Up the AWS Java SDK

For the AWS Java SDK, you must add a `.jar` file that includes the API for blueprints.

1. If you haven't already done so, set up the AWS SDK for Java.
 - For Java 1.x, follow the instructions in [Set up the AWS SDK for Java](#) in the *AWS SDK for Java Developer Guide*.
 - For Java 2.x, follow the instructions in [Setting up the AWS SDK for Java 2.x](#) in the *AWS SDK for Java 2.x Developer Guide*.
2. Download the client jar file that has access to the APIs for blueprints.
 - For Java 1.x: s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-preview/AWSGlueJavaClient-1.11.x.jar
 - For Java 2.x: s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-v2-preview/AwsJavaSdk-Glue-2.0.jar
3. Add the client jar to the front of the Java classpath to override the AWS Glue client provided by the AWS Java SDK.

```
export CLASSPATH=<path-to-preview-client-jar>:$CLASSPATH
```

4. (Optional) Test the SDK with the following Java application. The application should output an empty list.

Replace `accessKey` and `secretKey` with your credentials, and replace `us-east-1` with your Region.

```
import com.amazonaws.auth.AWS Credentials;
import com.amazonaws.auth.AWS CredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWS Credentials;
import com.amazonaws.services.glue.AWS Glue;
import com.amazonaws.services.glue.AWS Glue ClientBuilder;
import com.amazonaws.services.glue.model.ListBlueprintsRequest;

public class App{
    public static void main(String[] args) {
        AWS Credentials credentials = new BasicAWS Credentials("accessKey", "secretKey");
        AWS CredentialsProvider provider = new
        AWSStaticCredentialsProvider(credentials);
        AWS Glue glue = AWS Glue ClientBuilder.standard().withCredentials(provider)
            .withRegion("us-east-1").build();
        ListBlueprintsRequest request = new ListBlueprintsRequest().withMaxResults(2);
        System.out.println(glue.listBlueprints(request));
    }
}
```

Set Up the AWS Python SDK

The following steps assume that you have Python version 2.7 or later, or version 3.6 or later installed on your computer.

1. Download the following boto3 wheel file. If prompted to open or save, save the file. s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/boto3-1.17.31-py2.py3-none-any.whl
2. Download the following botocore wheel file: s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/botocore-1.20.31-py2.py3-none-any.whl
3. Check your Python version.

```
python --version
```

4. Depending on your Python version, enter the following commands (for Linux):

- For Python 2.7 or later.

```
python3 -m pip install --user virtualenv
source env/bin/activate
```

- For Python 3.6 or later.

```
python3 -m venv python-sdk-test
source python-sdk-test/bin/activate
```

5. Install the botocore wheel file.

```
python3 -m pip install <download-directory>/botocore-1.20.31-py2.py3-none-any.whl
```

6. Install the boto3 wheel file.

```
python3 -m pip install <download-directory>/boto3-1.17.31-py2.py3-none-any.whl
```

7. Configure your credentials and default region in the `~/.aws/credentials` and `~/.aws/config` files. For more information, see [Configuring the AWS CLI](#) in the *AWS Command Line Interface User Guide*.
8. (Optional) Test your setup. The following commands should return an empty list.

Replace `us-east-1` with your Region.

```
$ python
>>> import boto3
>>> glue = boto3.client('glue', 'us-east-1')
>>> glue.list_blueprints()
```

Set Up the Preview AWS CLI

1. If you haven't already done so, install and/or update the AWS Command Line Interface (AWS CLI) on your computer. The easiest way to do this is with `pip`, the Python installer utility:

```
pip install awscli --upgrade --user
```

You can find complete installation instructions for the AWS CLI here: [Installing the AWS Command Line Interface](#).

2. Download the AWS CLI wheel file from: `s3://awsglue-custom-blueprints-preview-artifacts/awscli-preview-build/awscli-1.19.31-py2.py3-none-any.whl`
3. Install the AWS CLI wheel file.

```
python3 -m pip install awscli-1.19.31-py2.py3-none-any.whl
```

4. Run the `aws configure` command. Configure your AWS credentials (including access key, and secret key) and AWS Region. You can find information on configuring the AWS CLI here: [Configuring the AWS CLI](#).
5. Test the AWS CLI. The following command should return an empty list.

Replace `us-east-1` with your Region.

```
aws glue list-blueprints --region us-east-1
```

Writing the Blueprint Code

Each blueprint project that you create must contain at a minimum the following files:

- A Python layout script that defines the workflow. The script contains a function that defines the entities (jobs and crawlers) in a workflow, and the dependencies between them.
- A configuration file, `blueprint.cfg`, which defines:
 - The full path of the workflow layout definition function.
 - The parameters that the blueprint accepts.

Topics

- [Creating the Blueprint Layout Script \(p. 382\)](#)
- [Creating the Configuration File \(p. 384\)](#)
- [Specifying Blueprint Parameters \(p. 385\)](#)

Creating the Blueprint Layout Script

The blueprint layout script must include a function that generates the entities in your workflow. You can name this function whatever you like. AWS Glue uses the configuration file to determine the fully qualified name of the function.

Your layout function does the following:

- (Optional) Instantiates the `Job` class to create `Job` objects, and passes arguments such as `Command` and `Role`. These are job properties that you would specify if you were creating the job using the AWS Glue console or API.
- (Optional) Instantiates the `Crawler` class to create `Crawler` objects, and passes `name`, `role`, and `target` arguments.
- To indicate dependencies between the objects (workflow entities), passes the `DependsOn` and `WaitForDependencies` additional arguments to `Job()` and `Crawler()`. These arguments are explained later in this section.
- Instantiates the `Workflow` class to create the workflow object that is returned to AWS Glue, passing a `Name` argument, an `Entities` argument, and an optional `OnSchedule` argument. The `Entities` argument specifies all of the jobs and crawlers to include in the workflow. To see how to construct an `Entities` object, see the sample project later in this section.
- Returns the `Workflow` object.

For definitions of the `Job`, `Crawler`, and `Workflow` classes, see [AWS Glue Blueprint Classes Reference \(p. 391\)](#).

The layout function must accept the following input arguments.

Argument	Description
<code>user_params</code>	Python dictionary of blueprint parameter names and values. For more information, see Specifying Blueprint Parameters (p. 385) .
<code>system_params</code>	Python dictionary containing two properties: <code>region</code> and <code>accountId</code> .

Here is a sample layout generator script in a file named `Layout.py`:

```

import argparse
import sys
import os
import json
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

def generate_layout(user_params, system_params):

    etl_job = Job(Name="{}_etl_job".format(user_params['WorkflowName']),
                  Command={
                      "Name": "glueetl",
                      "ScriptLocation": user_params['ScriptLocation'],
                      "PythonVersion": "2"
                  },
                  Role=user_params['PassRole'])
    post_process_job = Job(Name="{}_post_process".format(user_params['WorkflowName']),
                           Command={
                               "Name": "pythonshell",
                               "ScriptLocation": user_params['ScriptLocation'],
                               "PythonVersion": "2"
                           },
                           Role=user_params['PassRole'],
                           DependsOn={
                               etl_job: "SUCCEEDED"
                           },
                           WaitForDependencies="AND")
    sample_workflow = Workflow(Name=user_params['WorkflowName'],
                                Entities=Entities(Jobs=[etl_job, post_process_job]))
    return sample_workflow

```

The sample script imports the required blueprint libraries and includes a `generate_layout` function that generates a workflow with two jobs. This is a very simple script. A more complex script could employ additional logic and parameters to generate a workflow with many jobs and crawlers, or even a variable number of jobs and crawlers.

Using the `DependsOn` Argument

The `DependsOn` argument is a dictionary representation of a dependency that this entity has on other entities within the workflow. It has the following form:

```
DependsOn = {dependency1 : state, dependency2 : state, ...}
```

The keys in this dictionary represent the object reference, not the name, of the entity, while the values are strings that correspond to the state to watch for. AWS Glue infers the proper triggers. For the valid states, see [Condition Structure](#).

For example, a job might depend on the successful completion of a crawler. If you define a crawler object named `crawler2` as follows:

```
crawler2 = Crawler(Name="my_crawler", ...)
```

Then an object depending on `crawler2` would include a constructor argument such as:

```
DependsOn = {crawler2 : "SUCCEEDED"}
```

For example:

```
job1 = Job(Name="Job1", ..., DependsOn = {crawler2 : "SUCCEEDED", ...})
```

If `DependsOn` is omitted for an entity, that entity depends on the workflow start trigger.

Using the `WaitForDependencies` Argument

The `WaitForDependencies` argument defines whether a job or crawler entity should wait until *all* entities on which it depends complete or until *any* completes.

The allowable values are "AND" or "ANY".

Using the `OnSchedule` Argument

The `OnSchedule` argument for the `Workflow` class constructor is a cron expression that defines the starting trigger definition for a workflow.

If this argument is specified, AWS Glue creates a schedule trigger with the corresponding schedule. If it isn't specified, the starting trigger for the workflow is an on-demand trigger.

Creating the Configuration File

The blueprint configuration file is a required file that defines the script entry point for generating the workflow, and the parameters that the blueprint accepts. The file must be named `blueprint.cfg`.

Here is a sample configuration file.

```
{
    "layoutGenerator": "DemoBlueprintProject.Layout.generate_layout",
    "parameterSpec" : {
        "WorkflowName" : {
            "type": "String",
            "collection": false
        },
        "WorkerType" : {
            "type": "String",
            "collection": false,
            "allowedValues": ["G1.X", "G2.X"],
            "defaultValue": "G1.X"
        },
        "Dpu" : {
            "type" : "Integer",
            "allowedValues" : [2, 4, 6],
            "defaultValue" : 2
        },
        "DynamoDBTableName": {
            "type": "String",
            "collection" : false
        },
        "ScriptLocation" : {
            "type": "String",
            "collection": false
        }
    }
}
```

The `layoutGenerator` property specifies the fully qualified name of the function in the script that generates the layout.

The `parameterSpec` property specifies the parameters that this blueprint accepts. For more information, see [Specifying Blueprint Parameters \(p. 385\)](#).

Important

Your configuration file must include the workflow name as a blueprint parameter, or you must generate a unique workflow name in your layout script.

Specifying Blueprint Parameters

The configuration file contains blueprint parameter specifications in a `parameterSpec` JSON object. `parameterSpec` contains one or more parameter objects.

```
"parameterSpec": {
    "<parameter_name>": {
        "type": "<parameter-type>",
        "collection": true|false,
        "description": "<parameter-description>",
        "defaultValue": "<default value for the parameter if value not specified>"
        "allowedValues": "<list of allowed values>"
    },
    "<parameter_name>": {
        ...
    }
}
```

The following are the rules for coding each parameter object:

- The parameter name and type are mandatory. All other properties are optional.
- If you specify the `defaultValue` property, the parameter is optional. Otherwise the parameter is mandatory and the data analyst who is creating a workflow from the blueprint must provide a value for it.
- If you set the `collection` property to `true`, the parameter can take a collection of values. Collections can be of any data type.
- If you specify `allowedValues`, the AWS Glue console displays a dropdown list of values for the data analyst to choose from when creating a workflow from the blueprint.

The following are the permitted values for type:

Parameter Data Type	Notes
String	-
Integer	-
Double	-
Boolean	Possible values are <code>true</code> and <code>false</code> . Generates a check box on the Create a workflow from <blueprint> page on the AWS Glue console.
S3Uri	Complete Amazon S3 path, beginning with <code>s3://</code> . Generates a text field and Browse button on the Create a workflow from <blueprint> page.
S3Bucket	Amazon S3 bucket name only. Generates a bucket picker on the Create a workflow from <blueprint> page.
IAMRoleArn	Amazon Resource Name (ARN) of an AWS Identity and Access Management (IAM) role. Generates a role picker on the Create a workflow from <blueprint> page.
IAMRoleName	Name of an IAM role. Generates a role picker on the Create a workflow from <blueprint> page.

Sample Blueprint Project

Data format conversion is a frequent extract, transform, and load (ETL) use case. In typical analytic workloads, column-based file formats like Parquet or ORC are preferred over text formats like CSV or JSON. This sample blueprint enables you to convert data from CSV/JSON/etc. into Parquet for files on Amazon S3.

This blueprint takes a list of S3 paths defined by a blueprint parameter, converts the data to Parquet format, and writes it to the S3 location specified by another blueprint parameter. The layout script creates a crawler and job for each path. The layout script also uploads the ETL script in `Conversion.py` to an S3 bucket specified by another blueprint parameter. The layout script then specifies the uploaded script as the ETL script for each job. The ZIP archive for the project contains the layout script, the ETL script, and the blueprint configuration file.

For information about more sample blueprint projects, see [Blueprint Samples \(p. 394\)](#).

The following is the layout script, in the file `Layout.py`.

```
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *
import boto3

s3_client = boto3.client('s3')

# Ingesting all the S3 paths as Glue table in parquet format
def generate_layout(user_params, system_params):
    #Always give the full path for the file
    with open("ConversionBlueprint/Conversion.py", "rb") as f:
        s3_client.upload_fileobj(f, user_params['ScriptsBucket'], "Conversion.py")
    etlScriptLocation = "s3://{}//Conversion.py".format(user_params['ScriptsBucket'])
    crawlers = []
    jobs = []
    workflowName = user_params['WorkflowName']
    for path in user_params['S3Paths']:
        tablePrefix = "source_"
        crawler = Crawler(Name="{}_crawler".format(workflowName),
                           Role=user_params['PassRole'],
                           DatabaseName=user_params['TargetDatabase'],
                           TablePrefix=tablePrefix,
                           Targets= {"S3Targets": [{"Path": path}]})
        crawlers.append(crawler)
        transform_job = Job(Name="{}_transform_job".format(workflowName),
                            Command={"Name": "glueetl",
                                      "ScriptLocation": etlScriptLocation,
                                      "PythonVersion": "3"},
                            Role=user_params['PassRole'],
                            DefaultArguments={"--database_name":
                                              user_params['TargetDatabase'],
                                              "--table_prefix": tablePrefix,
                                              "--region_name": system_params['region'],
                                              "--output_path":
                                              user_params['TargetS3Location']},
                            DependsOn={crawler: "SUCCEEDED"},
                            WaitForDependencies="AND")
        jobs.append(transform_job)
    conversion_workflow = Workflow(Name=workflowName, Entities=Entities(Jobs=jobs,
                                                                       Crawlers=crawlers))
    return conversion_workflow
```

The following is the corresponding blueprint configuration file `blueprint.cfg`.

```
{
    "layoutGenerator": "ConversionBlueprint.Layout.generate_layout",
    "parameterSpec" : {
        "WorkflowName" : {
            "type": "String",
            "collection": false,
            "description": "Name for the workflow."
        },
        "S3Paths" : {
            "type": "S3Uri",
            "collection": true,
            "description": "List of Amazon S3 paths for data ingestion."
        },
        "PassRole" : {
            "type": "IAMRoleName",
            "collection": false,
            "description": "Choose an IAM role to be used in running the job/crawler"
        },
        "TargetDatabase": {
            "type": "String",
            "collection" : false,
            "description": "Choose a database in the Data Catalog."
        },
        "TargetS3Location": {
            "type": "S3Uri",
            "collection" : false,
            "description": "Choose an Amazon S3 output path: ex:s3://<target_path>/."
        },
        "ScriptsBucket": {
            "type": "S3Bucket",
            "collection": false,
            "description": "Provide an S3 bucket name(in the same AWS Region) to store the
scripts."
        }
    }
}
```

The following script in the file Conversion.py is the uploaded ETL script. Note that it preserves the partitioning scheme during conversion.

```
import sys
from pyspark.sql.functions import *
from pyspark.context import SparkContext
from awsglue.transforms import *
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
import boto3

args = getResolvedOptions(sys.argv, [
    'JOB_NAME',
    'region_name',
    'database_name',
    'table_prefix',
    'output_path'])
databaseName = args['database_name']
tablePrefix = args['table_prefix']
outputPath = args['output_path']

glue = boto3.client('glue', region_name=args['region_name'])

glue_context = GlueContext(SparkContext.getOrCreate())
spark = glue_context.spark_session
job = Job(glue_context)
```

```

job.init(args['JOB_NAME'], args)

def get_tables(database_name, table_prefix):
    tables = []
    paginator = glue.get_paginator('get_tables')
    for page in paginator.paginate(DatabaseName=database_name, Expression=table_prefix
+ "*"):
        tables.extend(page['TableList'])
    return tables

for table in get_tables(databaseName, tablePrefix):
    tableName = table['Name']
    partitionList = table['PartitionKeys']
    partitionKeys = []
    for partition in partitionList:
        partitionKeys.append(partition['Name'])

    # Create DynamicFrame from Catalog
    dyf = glue_context.create_dynamic_frame.from_catalog(
        name_space=databaseName,
        table_name=tableName,
        additional_options={
            'useS3ListImplementation': True
        },
        transformation_ctx='dyf'
    )

    # Resolve choice type with make_struct
    dyf = ResolveChoice.apply(
        frame=dyf,
        choice='make_struct',
        transformation_ctx='resolvechoice_' + tableName
    )

    # Drop null fields
    dyf = DropNullFields.apply(
        frame=dyf,
        transformation_ctx="dropnullfields_" + tableName
    )

    # Write DynamicFrame to S3 in glueparquet
    sink = glue_context.getSink(
        connection_type="s3",
        path=outputPath,
        enableUpdateCatalog=True,
        partitionKeys=partitionKeys
    )
    sink.setFormat("glueparquet")

    sink.setCatalogInfo(
        catalogDatabase=databaseName,
        catalogTableName=tableName[len(tablePrefix):]
    )
    sink.writeFrame(dyf)

job.commit()

```

Note

Only two Amazon S3 paths can be supplied as an input to the sample blueprint. This is because AWS Glue triggers are limited to invoking only two crawler actions.

Testing a Blueprint

While you develop your code, you should perform local testing to verify that the workflow layout is correct.

Local testing doesn't generate AWS Glue jobs, crawlers, or triggers. Instead, you run the layout script locally and use the `to_json()` and `validate()` methods to print objects and find errors. These methods are available in all three classes defined in the libraries.

There are two ways to handle the `user_params` and `system_params` arguments that AWS Glue passes to your layout function. Your test-bench code can create a dictionary of sample blueprint parameter values and pass that to the layout function as the `user_params` argument. Or, you can remove the references to `user_params` and replace them with hardcoded strings.

If your code makes use of the `region` and `accountId` properties in the `system_params` argument, you can pass in your own dictionary for `system_params`.

To test a blueprint

1. Start a Python interpreter in a directory with the libraries, or load the blueprint files and the supplied libraries into your preferred integrated development environment (IDE).
2. Ensure that your code imports the supplied libraries.
3. Add code to your layout function to call `validate()` or `to_json()` on any entity or on the `Workflow` object. For example, if your code creates a `Crawler` object named `mycrawler`, you can call `validate()` as follows.

```
mycrawler.validate()
```

You can print `mycrawler` as follows:

```
print(mycrawler.to_json())
```

If you call `to_json` on an object, there is no need to also call `validate()`, because `to_json()` calls `validate()`.

It is most useful to call these methods on the workflow object. Assuming that your script names the workflow object `my_workflow`, validate and print the workflow object as follows.

```
print(my_workflow.to_json())
```

For more information about `to_json()` and `validate()`, see [Class Methods \(p. 394\)](#).

You can also import `pprint` and pretty-print the workflow object, as shown in the example later in this section.

4. Run the code, fix errors, and finally remove any calls to `validate()` or `to_json()`.

Example

The following example shows how to construct a dictionary of sample blueprint parameters and pass it in as the `user_params` argument to layout function `generate_compaction_workflow`. It also shows how to pretty-print the generated workflow object.

```
from pprint import pprint
from awsglue.blueprint.workflow import *
```

```

from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

USER_PARAMS = {"WorkflowName": "compaction_workflow",
               "ScriptLocation": "s3://awsexamplebucket1/scripts/threaded-compaction.py",
               "PassRole": "arn:aws:iam::111122223333:role/GlueRole-ETL",
               "DatabaseName": "cloudtrial",
               "TableName": "ct_cLOUDTRAIL",
               "CoalesceFactor": 4,
               "MaxThreadWorkers": 200}

def generate_compaction_workflow(user_params: dict, system_params: dict) -> Workflow:
    compaction_job = Job(Name=f"{user_params['WorkflowName']}_etl_job",
                          Command={"Name": "glueetl",
                                    "ScriptLocation": user_params['ScriptLocation'],
                                    "PythonVersion": "3"},
                          Role="arn:aws:iam::111122223333:role/AWSGlueServiceRoleDefault",
                          DefaultArguments={"DatabaseName": user_params['DatabaseName'],
                                            "TableName": user_params['TableName'],
                                            "CoalesceFactor": user_params['CoalesceFactor'],
                                            "max_thread_workers":
                                            user_params['MaxThreadWorkers']})

    catalog_target = {"CatalogTargets": [{"DatabaseName": user_params['DatabaseName'],
                                         "Tables": [user_params['TableName']]}]}

    compacted_files_crawler = Crawler(Name=f"{user_params['WorkflowName']}_post_crawl",
                                        Targets = catalog_target,
                                        Role=user_params['PassRole'],
                                        DependsOn={compaction_job: "SUCCEEDED"},
                                        WaitForDependencies="AND",
                                        SchemaChangePolicy={"DeleteBehavior": "LOG"})

    compaction_workflow = Workflow(Name=user_params['WorkflowName'],
                                    Entities=Entities(Jobs=[compaction_job],
                                                       Crawlers=[compacted_files_crawler]))
    return compaction_workflow

generated = generate_compaction_workflow(user_params=USER_PARAMS, system_params={})
gen_dict = generated.to_json()

pprint(gen_dict)

```

Publishing a Blueprint

After you develop a blueprint, you must upload it to Amazon S3. You must have write permissions on the Amazon S3 bucket that you use to publish the blueprint. You must also make sure that the AWS Glue administrator, who will register the blueprint, has read access to the Amazon S3 bucket. For the suggested AWS Identity and Access Management (IAM) permissions policies for personas and roles for AWS Glue blueprints, see [Permissions for Personas and Roles for AWS Glue Blueprints \(p. 424\)](#).

To publish a blueprint

1. Create the necessary scripts, resources, and blueprint configuration file.
2. Add all files to a ZIP archive and upload the ZIP file to Amazon S3. Use an S3 bucket that is in the same Region as the Region in which users will register and run the blueprint.

You can create a ZIP file from the command line using the following command.

```
zip -r folder.zip folder
```

3. Add a bucket policy that grants read permissions to the AWS desired account. The following is a sample policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-blueprints/*"
        }
    ]
}
```

4. Grant the IAM s3:GetObject permission on the Amazon S3 bucket to the AWS Glue administrator or to whoever will be registering blueprints. For a sample policy to grant to administrators, see [AWS Glue Administrator Permissions for Blueprints \(p. 425\)](#).

After you have completed local testing of your blueprint, you may also want to test a blueprint on AWS Glue. To test a blueprint on AWS Glue, it must be registered. You can limit who sees the registered blueprint using IAM authorization, or by using separate testing accounts.

See Also:

- [Registering a Blueprint in AWS Glue \(p. 394\)](#)

AWS Glue Blueprint Classes Reference

The libraries for AWS Glue blueprints define three classes that you use in your workflow layout script: `Job`, `Crawler`, and `Workflow`.

Topics

- [Job Class \(p. 391\)](#)
- [Crawler Class \(p. 392\)](#)
- [Workflow Class \(p. 393\)](#)
- [Class Methods \(p. 394\)](#)

Job Class

The `Job` class represents an AWS Glue ETL job.

Mandatory Constructor Arguments

The following are mandatory constructor arguments for the `Job` class.

Argument Name	Type	Description
<code>Name</code>	<code>str</code>	Name to assign to the job. AWS Glue adds a randomly generated suffix to the name to distinguish the job from those created by other blueprint runs.

Argument Name	Type	Description
Role	str	Amazon Resource Name (ARN) of the role that the job should assume while executing.
Command	dict	Job command, as specified in the JobCommand Structure (p. 847) in the API documentation.

Optional Constructor Arguments

The following are optional constructor arguments for the `Job` class.

Argument Name	Type	Description
DependsOn	dict	List of workflow entities that the job depends on. For more information, see Using the DependsOn Argument (p. 383) .
WaitForDependencies	str	Indicates whether the job should wait until <i>all</i> entities on which it depends complete before executing or until <i>any</i> completes. For more information, see Using the WaitForDependencies Argument (p. 384) . Omit if the job depends on only one entity.
(Job properties)	-	Any of the job properties listed in Job Structure (p. 845) in the AWS Glue API documentation (except <code>CreatedOn</code> and <code>LastModifiedOn</code>).

Crawler Class

The `Crawler` class represents an AWS Glue crawler.

Mandatory Constructor Arguments

The following are mandatory constructor arguments for the `Crawler` class.

Argument Name	Type	Description
Name	str	Name to assign to the crawler. AWS Glue adds a randomly generated suffix to the name to distinguish the crawler from those created by other blueprint runs.
Role	str	ARN of the role that the crawler should assume while running.
Targets	dict	Collection of targets to crawl. Targets class constructor arguments are defined in the CrawlerTargets Structure (p. 794) in the API documentation. All Targets constructor arguments are optional, but you must pass at least one.

Optional Constructor Arguments

The following are optional constructor arguments for the `Crawler` class.

Argument Name	Type	Description
<code>DependsOn</code>	<code>dict</code>	List of workflow entities that the crawler depends on. For more information, see Using the DependsOn Argument (p. 383) .
<code>WaitForDependencies</code>	<code>str</code>	Indicates whether the crawler should wait until <i>all</i> entities on which it depends complete before running or until <i>any</i> completes. For more information, see Using the WaitForDependencies Argument (p. 384) . Omit if the crawler depends on only one entity.
(Crawler properties)	-	Any of the crawler properties listed in Crawler Structure (p. 792) in the AWS Glue API documentation, with the following exceptions:
		<ul style="list-style-type: none"> • <code>State</code> • <code>CrawlElapsedTime</code> • <code>CreationTime</code> • <code>LastUpdated</code> • <code>LastCrawl</code> • <code>Version</code>

Workflow Class

The `Workflow` class represents an AWS Glue workflow. The workflow layout script returns a `Workflow` object. AWS Glue creates a workflow based on this object.

Mandatory Constructor Arguments

The following are mandatory constructor arguments for the `Workflow` class.

Argument Name	Type	Description
<code>Name</code>	<code>str</code>	Name to assign to the workflow.
<code>Entities</code>	<code>Entities</code>	A collection of entities (jobs and crawlers) to include in the workflow. The <code>Entities</code> class constructor accepts a <code>Jobs</code> argument, which is a list of <code>Job</code> objects, and a <code>Crawlers</code> argument, which is a list of <code>Crawler</code> objects.

Optional Constructor Arguments

The following are optional constructor arguments for the `Workflow` class.

Argument Name	Type	Description
<code>Description</code>	<code>str</code>	See Workflow Structure (p. 924) .

Argument Name	Type	Description
DefaultRunProperties	dict	See Workflow Structure (p. 924) .
OnSchedule	str	A cron expression.

Class Methods

All three classes include the following methods.

validate()

Validates the properties of the object and if errors are found, outputs a message and exits.
Generates no output if there are no errors. For the `Workflow` class, calls itself on every entity in the workflow.

to_json()

Serializes the object to JSON. Also calls `validate()`. For the `Workflow` class, the JSON object includes job and crawler lists, and a list of triggers generated by the job and crawler dependency specifications.

Blueprint Samples

There are a number of sample blueprint projects available on the [AWS Glue blueprint Github repository](#). These samples are for reference only and are not intended for production use.

The titles of the sample projects are:

- Compaction: this blueprint creates a job that compacts input files into larger chunks based on desired file size.
- Conversion: this blueprint converts input files in various standard file formats into Apache Parquet format, which is optimized for analytic workloads.
- Crawling Amazon S3 locations: this blueprint crawls multiple Amazon S3 locations to add metadata tables to the Data Catalog.
- Custom connection to Data Catalog: this blueprint accesses data stores using AWS Glue custom connectors, reads the records, and populates the table definitions in the AWS Glue Data Catalog based on the record schema.
- Encoding: this blueprint converts your non-UTF files into UTF encoded files.
- Partitioning: this blueprint creates a partitioning job that places output files into partitions based on specific partition keys.
- Importing Amazon S3 data into a DynamoDB table: this blueprint imports data from Amazon S3 into a DynamoDB table.
- Standard table to governed: this blueprint imports an AWS Glue Data Catalog table into a Lake Formation table.

Registering a Blueprint in AWS Glue

After the AWS Glue developer has coded the blueprint and uploaded a ZIP archive to Amazon Simple Storage Service (Amazon S3), an AWS Glue administrator must register the blueprint. Registering the blueprint makes it available for use.

When you register a blueprint, AWS Glue copies the blueprint archive to a reserved Amazon S3 location. You can then delete the archive from the upload location.

To register a blueprint, you need read permissions on the Amazon S3 location that contains the uploaded archive. You also need the AWS Identity and Access Management (IAM) permission `glue:CreateBlueprint`. For the suggested permissions for an AWS Glue administrator who must register, view, and maintain blueprints, see [AWS Glue Administrator Permissions for Blueprints \(p. 425\)](#).

You can register a blueprint by using the AWS Glue console, AWS Glue API, or AWS Command Line Interface (AWS CLI).

To register a blueprint (console)

1. Ensure that you have read permissions (`s3:GetObject`) on the blueprint ZIP archive in Amazon S3.

2. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.

Sign in as a user that has permissions to register a blueprint. Switch to the same AWS Region as the Amazon S3 bucket that contains the blueprint ZIP archive.

3. In the navigation pane, choose **Blueprints**. Then on the **Blueprints** page, choose **Add blueprint**.

4. Enter a blueprint name and optional description.

5. For **ZIP archive location (S3)**, enter the Amazon S3 path of the uploaded blueprint ZIP archive. Include the archive file name in the path and begin the path with `s3://`.

6. (Optional) Add tag one or more tags.

7. Choose **Add blueprint**.

The **Blueprints** page returns and shows that the blueprint status is **CREATING**. Choose the refresh button until the status changes to **ACTIVE** or **FAILED**.

8. If the status is **FAILED**, select the blueprint, and on the **Actions** menu, choose **View**.

The detail page shows the reason for the failure. If the error message is "Unable to access object at location..." or "Access denied on object at location...", review the following requirements:

- The user that you are signed in as must have read permission on the blueprint ZIP archive in Amazon S3.

- The Amazon S3 bucket that contains the ZIP archive must have a bucket policy that grants read permission on the object to your AWS account ID. For more information, see [Developing Blueprints in AWS Glue \(p. 378\)](#).

- The Amazon S3 bucket that you're using must be in the same Region as the Region that you're signed into on the console.

9. Ensure that data analysts have permissions on the blueprint.

The suggested IAM policy for data analysts is shown in [Data Analyst Permissions for Blueprints \(p. 425\)](#). This policy grants `glue:GetBlueprint` on any resource. If your policy is more fine-grained at the resource level, then grant data analysts permissions on this newly created resource.

To register a blueprint (AWS CLI)

1. Enter the following command.

```
aws glue create-blueprint --name <blueprint-name> [--description <description>] --blueprint-location s3://<s3-path>/<archive-filename>
```

2. Enter the following command to check the blueprint status. Repeat the command until the status goes to **ACTIVE** or **FAILED**.

```
aws glue get-blueprint --name <blueprint-name>
```

If the status is **FAILED** and the error message is "Unable to access object at location..." or "Access denied on object at location...", review the following requirements:

- The user that you are signed in as must have read permission on the blueprint ZIP archive in Amazon S3.
- The Amazon S3 bucket containing the ZIP archive must have a bucket policy that grants read permission on the object to your AWS account ID. For more information, see [Publishing a Blueprint \(p. 390\)](#).
- The Amazon S3 bucket that you're using must be in the same Region as the Region that you're signed into on the console.

See Also:

- [Overview of Blueprints in AWS Glue \(p. 376\)](#)

Viewing Blueprints in AWS Glue

View a blueprint to review the blueprint description, status, and parameter specifications, and to download the blueprint ZIP archive.

You can view a blueprint by using the AWS Glue console, AWS Glue API, or AWS Command Line Interface (AWS CLI).

To view a blueprint (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Blueprints**.
3. On the **Blueprints** page, select a blueprint. Then on the **Actions** menu, choose **View**.

To view a blueprint (AWS CLI)

- Enter the following command to view just the blueprint name, description, and status. Replace `<blueprint-name>` with the name of the blueprint to view.

```
aws glue get-blueprint --name <blueprint-name>
```

The command output looks something like the following.

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
        "LastModifiedOn": 1587428838.671,
        "BlueprintLocation": "s3://awsexamplebucket1/demo/DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
}
```

Enter the following command to also view the parameter specifications.

```
aws glue get-blueprint --name <blueprint-name> --include-parameter-spec
```

The command output looks something like the following.

```
{  
    "Blueprint": {  
        "Name": "myDemoBP",  
        "CreatedOn": 1587414516.92,  
        "LastModifiedOn": 1587428838.671,  
        "ParameterSpec": "{\"WorkflowName\":{\"type\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null},\"PassRole\":{\"type\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null},\"DynamoDBTableName\":{\"type\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null},\"ScriptLocation\":{\"type\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null}}",  
        "BlueprintLocation": "s3://awsexamplebucket1/demo/DemoBlueprintProject.zip",  
        "Status": "ACTIVE"  
    }  
}
```

Add the `--include-blueprint` argument to include a URL in the output that you can paste into your browser to download the blueprint ZIP archive that AWS Glue stored.

See Also:

- [Overview of Blueprints in AWS Glue \(p. 376\)](#)

Updating a Blueprint in AWS Glue

You can update a blueprint if you have a revised layout script, a revised set of blueprint parameters, or revised supporting files. Updating a blueprint creates a new version.

Updating a blueprint doesn't affect existing workflows created from the blueprint.

You can update a blueprint by using the AWS Glue console, AWS Glue API, or AWS Command Line Interface (AWS CLI).

The following procedure assumes that the AWS Glue developer has created and uploaded an updated blueprint ZIP archive to Amazon S3.

To update a blueprint (console)

1. Ensure that you have read permissions (`s3:GetObject`) on the blueprint ZIP archive in Amazon S3.
2. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.

Sign in as a user that has permissions to update a blueprint. Switch to the same AWS Region as the Amazon S3 bucket that contains the blueprint ZIP archive.

3. In the navigation pane, choose **Blueprints**.
4. On the **Blueprints** page, select a blueprint, and on the **Actions** menu, choose **Edit**.
5. On the **Edit a blueprint** page, update the blueprint **Description** or **ZIP archive location (S3)**. Be sure to include the archive file name in the path.
6. Choose **Save**.

The **Blueprints** page returns and shows that the blueprint status is UPDATING. Choose the refresh button until the status changes to ACTIVE or FAILED.

7. If the status is FAILED, select the blueprint, and on the **Actions** menu, choose **View**.

The detail page shows the reason for the failure. If the error message is "Unable to access object at location..." or "Access denied on object at location...", review the following requirements:

- The user that you are signed in as must have read permission on the blueprint ZIP archive in Amazon S3.
- The Amazon S3 bucket that contains the ZIP archive must have a bucket policy that grants read permission on the object to your AWS account ID. For more information, see [Publishing a Blueprint \(p. 390\)](#).
- The Amazon S3 bucket that you're using must be in the same Region as the Region that you're signed into on the console.

Note

If the update fails, the next blueprint run uses the latest version of the blueprint that was successfully registered or updated.

To update a blueprint (AWS CLI)

1. Enter the following command.

```
aws glue update-blueprint --name <blueprint-name> [--description <description>] --blueprint-location s3://<s3-path>/<archive-filename>
```

2. Enter the following command to check the blueprint status. Repeat the command until the status goes to ACTIVE or FAILED.

```
aws glue get-blueprint --name <blueprint-name>
```

If the status is FAILED and the error message is "Unable to access object at location..." or "Access denied on object at location...", review the following requirements:

- The user that you are signed in as must have read permission on the blueprint ZIP archive in Amazon S3.
- The Amazon S3 bucket containing the ZIP archive must have a bucket policy that grants read permission on the object to your AWS account ID. For more information, see [Publishing a Blueprint \(p. 390\)](#).
- The Amazon S3 bucket that you're using must be in the same Region as the Region that you're signed into on the console.

See Also

- [Overview of Blueprints in AWS Glue \(p. 376\)](#)

Creating a Workflow from a Blueprint in AWS Glue

You can create an AWS Glue workflow manually, adding one component at a time, or you can create a workflow from an AWS Glue [blueprint \(p. 376\)](#). AWS Glue includes blueprints for common use cases. Your AWS Glue developers can create additional blueprints.

Important

Limit the total number of jobs, crawlers, and triggers within a workflow to 100 or less. If you include more than 100, you might get errors when trying to resume or stop workflow runs.

When you use a blueprint, you can quickly generate a workflow for a specific use case based on the generalized use case defined by the blueprint. You define the specific use case by providing values for the blueprint parameters. For example, a blueprint that partitions a dataset could have the Amazon S3 source and target paths as parameters.

AWS Glue creates a workflow from a blueprint by *running* the blueprint. The blueprint run saves the parameter values that you supplied, and is used to track the progress and outcome of the creation of the workflow and its components. When troubleshooting a workflow, you can view the blueprint run to determine the blueprint parameter values that were used to create a workflow.

To create and view workflows, you require certain IAM permissions. For a suggested IAM policy, see [Data Analyst Permissions for Blueprints \(p. 425\)](#).

You can create a workflow from a blueprint by using the AWS Glue console, AWS Glue API, or AWS Command Line Interface (AWS CLI).

To create a workflow from a blueprint (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
Sign in as a user that has permissions to create a workflow.
2. In the navigation pane, choose **Blueprints**.
3. Select a blueprint, and on the **Actions** menu, choose **Create workflow**.
4. On the **Create a workflow from <blueprint-name>** page, enter the following information:

Blueprint parameters

These vary depending on the blueprint design. For questions about the parameters, see the developer. Blueprints typically include a parameter for the workflow name.

IAM role

The role that AWS Glue assumes to create the workflow and its components. The role must have permissions to create and delete workflows, jobs, crawlers, and triggers. For a suggested policy for the role, see [Permissions for Blueprint Roles \(p. 426\)](#).

5. Choose **Submit**.
The **Blueprint Details** page appears, showing a list of blueprint runs at the bottom.
6. In the blueprint runs list, check the topmost blueprint run for workflow creation status.
The initial status is **RUNNING**. Choose the refresh button until the status goes to **SUCCEEDED** or **FAILED**.
7. Do one of the following:
 - If the completion status is **SUCCEEDED**, you can go to the **Workflows** page, select the newly created workflow, and run it. Before running the workflow, you can review the design graph.
 - If the completion status is **FAILED**, select the blueprint run, and on the **Actions** menu, choose **View** to see the error message.

For more information on workflows and blueprints, see the following topics.

- [Overview of Workflows in AWS Glue \(p. 373\)](#)
- [Updating a Blueprint in AWS Glue \(p. 397\)](#)
- [Creating and Building Out a Workflow Manually in AWS Glue \(p. 400\)](#)

Viewing Blueprint Runs in AWS Glue

View a blueprint run to see the following information:

- Name of the workflow that was created.
- Blueprint parameter values that were used to create the workflow.
- Status of the workflow creation operation.

You can view a blueprint run by using the AWS Glue console, AWS Glue API, or AWS Command Line Interface (AWS CLI).

To view a blueprint run (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Blueprints**.
3. On the **Blueprints** page, select a blueprint. Then on the **Actions** menu, choose **View**.
4. At the bottom of the **Blueprint Details** page, select a blueprint run, and on the **Actions** menu, choose **View**.

To view a blueprint run (AWS CLI)

- Enter the following command. Replace `<blueprint-name>` with the name of the blueprint. Replace `<blueprint-run-id>` with the blueprint run ID.

```
aws glue get-blueprint-run --blueprint-name <blueprint-name> --run-id <blueprint-run-id>
```

See Also:

- [Overview of Blueprints in AWS Glue \(p. 376\)](#)

Creating and Building Out a Workflow Manually in AWS Glue

You can use the AWS Glue console to manually create and build out a workflow one node at a time.

A workflow contains jobs, crawlers, and triggers. Before manually creating a workflow, create the jobs and crawlers that the workflow is to include. It's 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.

Important

Limit the total number of jobs, crawlers, and triggers within a workflow to 100 or less. If you include more than 100, you might get errors when trying to resume or stop workflow runs.

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 in AWS Glue \(p. 416\)](#).

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. Then, at the bottom of the page, ensure that the **Graph** tab is selected.
2. Choose **Add trigger**, and in the **Add trigger** dialog box, do one of the following:
 - 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**.

- Choose **Add new**, and complete the **Add trigger** form.
 1. For **Trigger type**, select **Schedule**, **On demand**, or **EventBridge event**.

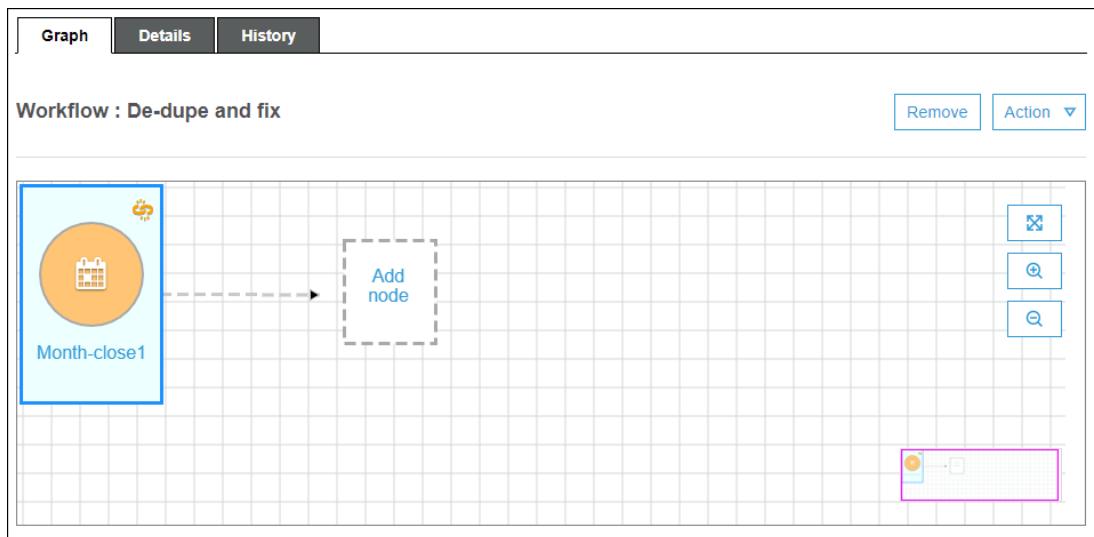
For trigger type **Schedule**, choose one of the **Frequency** options. Choose **Custom** to enter a cron expression.

For trigger type **EventBridge event**, enter **Number of events** (batch size), and optionally enter **Time delay** (batch window). If you omit **Time delay**, the batch window defaults to 15 minutes. For more information, see [Overview of Workflows in AWS Glue \(p. 373\)](#).

2. Choose **Add**.

The trigger appears on the graph, along with a placeholder node (labeled **Add node**). In the example below, the start trigger is a schedule trigger named `Month-close1`.

At this point, the trigger isn't saved yet.



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: Add More Triggers

Continue to build out your workflow by adding more triggers of type **Event**. 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:

Note

There is no action to save the workflow. After you add your last trigger and assign actions to the trigger, the workflow is complete and saved. You can always come back later and add more nodes.

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**.

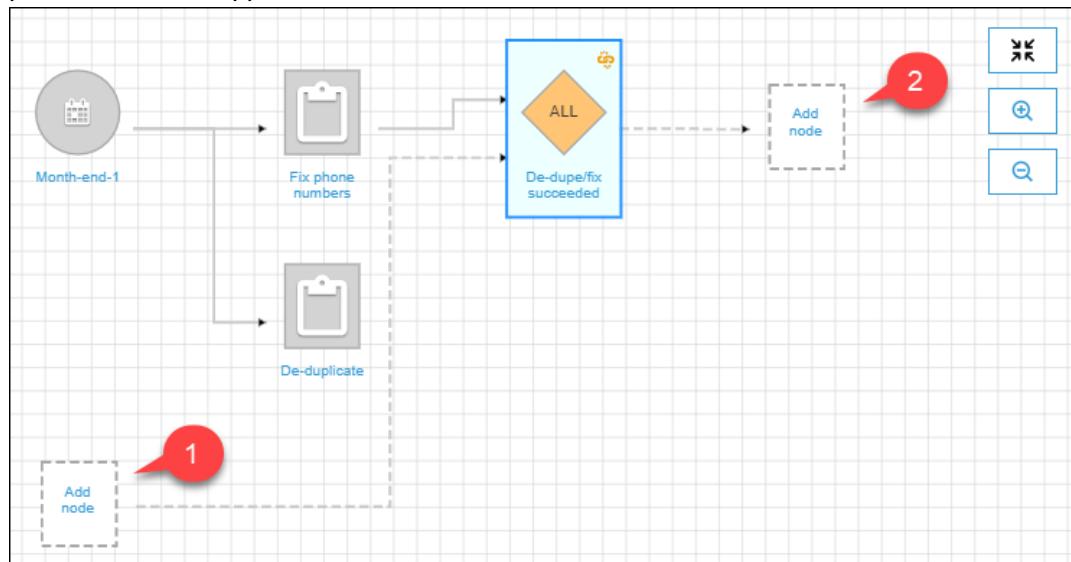
The trigger appears on the graph, along with the jobs and crawlers that it watches and the jobs and crawlers that it starts.

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:

- Select the new trigger.

As the following graph shows, the trigger `De-dupe/fix succeeded` is selected, and placeholder nodes appear for (1) events to watch and (2) actions.



- (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**.
- Ensure that the trigger is selected, and choose the actions placeholder node.
- 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.

For more information on workflows and blueprints, see the following topics.

- [Overview of Workflows in AWS Glue \(p. 373\)](#)
- [Running and Monitoring a Workflow in AWS Glue \(p. 409\)](#)
- [Creating a Workflow from a Blueprint in AWS Glue \(p. 398\)](#)

Starting an AWS Glue Workflow with an Amazon EventBridge Event

Amazon EventBridge, also known as CloudWatch Events, enables you 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 EventBridge 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.

With EventBridge support, AWS Glue can serve as an event producer and consumer in an event-driven architecture. For workflows, AWS Glue supports any type of EventBridge event as a consumer. The likely most common use case is the arrival of a new object in an Amazon S3 bucket. If you have data arriving in irregular or undefined intervals, you can process this data as close to its arrival as possible.

Note

AWS Glue does not provide guaranteed delivery of EventBridge messages. AWS Glue performs no deduplication if EventBridge delivers duplicate messages. You must manage idempotency based on your use case.

Be sure to configure EventBridge rules correctly to avoid sending unwanted events.

Before you begin

If you want to start a workflow with Amazon S3 data events, you must ensure that events for the S3 bucket of interest are logged to AWS CloudTrail and EventBridge. To do so, you must create a CloudTrail trail. For more information, see [Creating a trail for your AWS account](#).

To start a workflow with an EventBridge event

Note

In the following commands, replace:

- *<workflow-name>* with the name to assign to the workflow.
- *<trigger-name>* with the name to assign to the trigger.
- *<bucket-name>* with the name of the Amazon S3 bucket.
- *<account-id>* with a valid AWS account ID.
- *<region>* with the name of the Region (for example, `us-east-1`).
- *<rule-name>* with the name to assign to the EventBridge rule.

1. Ensure that you have AWS Identity and Access Management (IAM) permissions to create and view EventBridge rules and targets. The following is a sample policy that you can attach. You might want to scope it down to put limits on the operations and resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "events:PutRule",  
                "events:DisableRule",  
                "events>DeleteRule",  
                "events:PutTargets",  
                "events:RemoveTargets",  
                "events:EnableRule",  
                "events>List*" ,  
            ]  
        }  
    ]  
}
```

```
        "events:Describe*"
    ],
    "Resource": "*"
}
}
```

2. Create an IAM role that the EventBridge service can assume when passing an event to AWS Glue.
 - a. On the **Create role** page of the IAM console, choose **AWS Service**. Then choose the service **CloudWatch Events**.
 - b. Complete the **Create role** wizard. The wizard automatically attaches the CloudWatchEventsBuiltInTargetExecutionAccess and CloudWatchEventsInvocationAccess policies.
 - c. Attach the following inline policy to the role. This policy allows the EventBridge service to direct events to AWS Glue.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:notifyEvent"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>"
            ]
        }
    ]
}
```

3. Enter the following command to create the workflow.

See [create-workflow](#) in the *AWS CLI Command Reference* for information about additional optional command-line parameters.

```
aws glue create-workflow --name <workflow-name>
```

4. Enter the following command to create an EventBridge event trigger for the workflow. This will be the start trigger for the workflow. Replace **<actions>** with the actions to perform (the jobs and crawlers to start).

See [create-trigger](#) in the *AWS CLI Command Reference* for information about how to code the **actions** argument.

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT --name <trigger-name> --actions <actions>
```

If you want the workflow to be triggered by a batch of events instead of a single EventBridge event, enter the following command instead.

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT --name <trigger-name> --event-batching-condition BatchSize=<number-of-events>,BatchWindow=<seconds> --actions <actions>
```

For the **event-batching-condition** argument, **BatchSize** is required and **BatchWindow** is optional. If **BatchWindow** is omitted, the window defaults to 900 seconds, which is the maximum window size.

Example

The following example creates a trigger that starts the `eventtest` workflow after three EventBridge events have arrived, or five minutes after the first event arrives, whichever comes first.

```
aws glue create-trigger --workflow-name eventtest --type EVENT --name objectArrival --event-batching-condition BatchSize=3,BatchWindow=300 --actions JobName=test1
```

5. Create a rule in Amazon EventBridge.

- a. Create the JSON object for the rule details in your preferred text editor.

The following example specifies Amazon S3 as the event source, `PutObject` as the event name, and the bucket name as a request parameter. This rule starts a workflow when a new object arrives in the bucket.

```
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
    }
  }
}
```

To start the workflow when a new object arrives in a folder within the bucket, you can substitute the following code for `requestParameters`.

```
"requestParameters": {
  "bucketName": [
    "<bucket-name>"
  ]
  "key" : { "prefix" : "<folder1>/<folder2>/*"}`}
}
```

- b. Use your preferred tool to convert the rule JSON object to an escaped string.

```
{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-type\": [\n    \"AWS API\nCall via CloudTrail\"\n  ],\n  \"detail\": {\n    \"eventSource\": [\n      \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n      \"PutObject\"\n    ],\n    \"requestParameters\": {\n      \"bucketName\": [\n        \"<bucket-name>\"\n      ]\n    }\n  }\n}
```

- c. Run the following command to create a JSON parameter template that you can edit to specify input parameters to a subsequent `put-rule` command. Save the output in a file. In this example, the file is called `ruleCommand`.

```
aws events put-rule --name <rule-name> --generate-cli-skeleton >ruleCommand
```

For more information about the --generate-cli-skeleton parameter, see [Generating AWS CLI skeleton and input parameters from a JSON or YAML input file](#) in the *AWS Command Line Interface User Guide*.

The output file should look like the following.

```
{
    "Name": "",
    "ScheduleExpression": "",
    "EventPattern": "",
    "State": "ENABLED",
    "Description": "",
    "RoleArn": "",
    "Tags": [
        {
            "Key": "",
            "Value": ""
        }
    ],
    "EventBusName": ""
}
```

- d. Edit the file to optionally remove parameters and to specify at a minimum the Name, EventPattern, and State parameters. For the EventPattern parameter, provide the escaped string for the rule details that you created in a previous step.

```
{
    "Name": "<rule-name>",
    "EventPattern": "{\n        \"source\": [\n            \"aws.s3\"\n        ],\n        \"detail-type\": [\n            \"AWS API Call via CloudTrail\"\n        ],\n        \"detail\": {\n            \"eventSource\": [\n                \"s3.amazonaws.com\"\n            ],\n            \"eventName\": [\n                \"PutObject\"\n            ],\n            \"requestParameters\": {\n                \"bucketName\": [\n                    \"<bucket-name>\"\n                ]\n            }\n        }\n    }",
    "State": "DISABLED",
    "Description": "Start an AWS Glue workflow upon new file arrival in an Amazon S3 bucket"
}
```

Note

It is best to leave the rule disabled until you finish building out the workflow.

- e. Enter the following put-rule command, which reads input parameters from the file ruleCommand.

```
aws events put-rule --name <rule-name> --cli-input-json file://ruleCommand
```

The following output indicates success.

```
{
    "RuleArn": "<rule-arn>"
}
```

6. Enter the following command to attach the rule to a target. The target is the workflow in AWS Glue. Replace <role-name> with the role that you created at the beginning of this procedure.

```
aws events put-targets --rule <rule-name> --targets
  "Id"="1", "Arn"="arn:aws:glue:<region>:<account-id>:workflow/<workflow-
  name>", "RoleArn"="arn:aws:iam::<account-id>:role/<role-name>" --region <region>
```

The following output indicates success.

```
{
  "FailedEntryCount": 0,
  "FailedEntries": []
}
```

7. Confirm successful connection of the rule and target by entering the following command.

```
aws events list-rule-names-by-target --target-arn arn:aws:glue:<region>:<account-
  id>:workflow/<workflow-name>
```

The following output indicates success, where `<rule-name>` is the name of the rule that you created.

```
{
  "RuleNames": [
    "<rule-name>"
  ]
}
```

8. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
9. Select the workflow, and verify that the start trigger and its actions—the jobs or crawlers that it starts—appear on the workflow graph. Then continue with the procedure in [Step 3: Add More Triggers \(p. 402\)](#). Or add more components to the workflow by using the AWS Glue API or AWS Command Line Interface.
10. When the workflow is completely specified, enable the rule.

```
aws events enable-rule --name <rule-name>
```

The workflow is now ready to be started by an EventBridge event or event batch.

See Also

- [Amazon EventBridge User Guide](#)
- [Overview of Workflows in AWS Glue \(p. 373\)](#)
- [Creating and Building Out a Workflow Manually in AWS Glue \(p. 400\)](#)

Viewing the EventBridge Events That Started a Workflow

You can view the event ID of the Amazon EventBridge event that started your workflow. If your workflow was started by a batch of events, you can view the event IDs of all events in the batch.

For workflows with a batch size greater than one, you can also see which batch condition started the workflow: the arrival of the number of events in the batch size, or the expiration of the batch window.

To view the EventBridge events that started a workflow (console)

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, choose **Workflows**.
3. Select the workflow. Then at the bottom, choose the **History** tab.
4. Select a workflow run, and then choose **View run details**.
5. On the run details page, locate the **Run properties** field, and look for the **aws:eventIds** key.

The value for that key is a list of EventBridge event IDs.

To view the EventBridge events that started a workflow (AWS API)

- Include the following code in your Python script.

```
workflow_params =  
    glue_client.get_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id)  
batched_events = workflow_params['aws:eventIds']
```

`batched_events` will be a list of strings, where each string is an event ID.

See Also

- [Amazon EventBridge User Guide](#)
- the section called “Overview of Workflows” (p. 373)

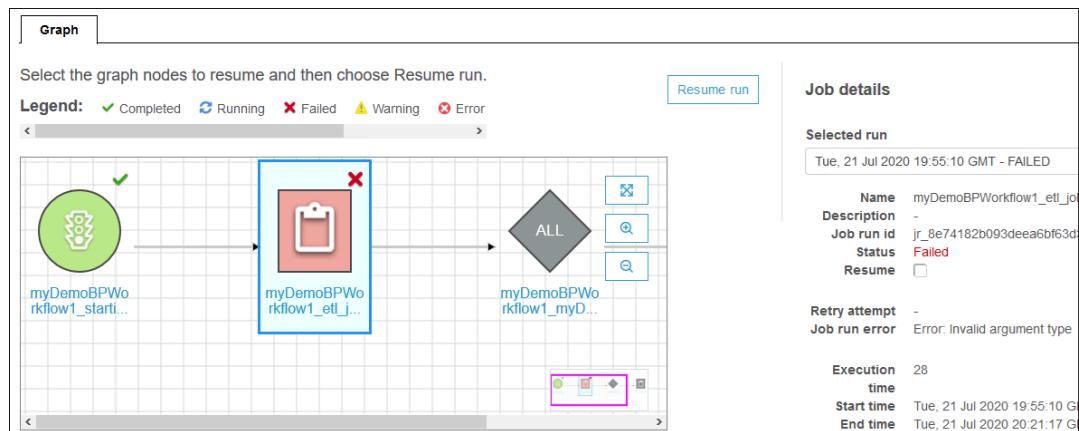
Running and Monitoring a Workflow in AWS Glue

If the start trigger for a workflow is an on-demand trigger, you can start the workflow from the AWS Glue console. Complete the following steps to run and monitor a workflow. If the workflow fails, you can view the run graph to determine the node that failed. To help troubleshoot, if the workflow was created from a blueprint, you can view the blueprint run to see the blueprint parameter values that were used to create the workflow. For more information, see [the section called “Viewing Blueprint Runs” \(p. 400\)](#).

You can run and monitor a workflow by using the AWS Glue console, API, or AWS Command Line Interface (AWS CLI).

To run and monitor a workflow (console)

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**.
 4. Check the **Last run status** column in the workflows list. Choose the refresh button to view ongoing workflow status.
 5. While the workflow is running or after it has completed (or failed), view the run details by completing the following steps.
 - a. Ensure that the workflow is selected, and choose the **History** tab.
 - b. Choose the current or most recent workflow run, and then choose **View run details**.
- The workflow runtime graph shows the current run status.
- c. Choose any node in the graph to view details and status of the node.



To run and monitor a workflow (AWS CLI)

1. Enter the following command. Replace <workflow-name> with the workflow to run.

```
aws glue start-workflow-run --name <workflow-name>
```

If the workflow is successfully started, the command returns the run ID.

2. View workflow run status by using the get-workflow-run command. Supply the workflow name and run ID.

```
aws glue get-workflow-run --name myWorkflow --run-id
wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705
```

The following is sample command output.

```
{
  "Run": {
    "Name": "myWorkflow",
    "WorkflowRunId": "wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705",
    "WorkflowRunProperties": {
      "run_state": "COMPLETED",
      "unique_id": "fee63f30-c512-4742-a9b1-7c8183bd1aae2"
    },
    "StartedOn": 1578556843.049,
    "CompletedOn": 1578558649.928,
    "Status": "COMPLETED",
    "Statistics": {
      "TotalActions": 11,
      "TimeoutActions": 0,
      "FailedActions": 0,
      "StoppedActions": 0,
      "SucceededActions": 9,
      "RunningActions": 0,
      "ErroredActions": 0
    }
  }
}
```

See Also:

- [the section called “Overview of Workflows” \(p. 373\)](#)
- [the section called “Overview of Blueprints” \(p. 376\)](#)

Stopping a Workflow Run

You can use the AWS Glue console, AWS Command Line Interface (AWS CLI) or AWS Glue API to stop a workflow run. When you stop a workflow run, all running jobs and crawlers are immediately terminated, and jobs and crawlers that are not yet started never start. It might take up to a minute for all running jobs and crawlers to stop. The workflow run status goes from **Running** to **Stopping**, and when the workflow run is completely stopped, the status goes to **Stopped**.

After the workflow run is stopped, you can view the run graph to see which jobs and crawlers completed and which never started. You can then determine if you must perform any steps to ensure data integrity. Stopping a workflow run causes no automatic rollback operations to be performed.

To stop a workflow run (console)

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
 2. In the navigation pane, under **ETL**, choose **Workflows**.
 3. Choose a running workflow, and then choose the **History** tab.
 4. Choose the workflow run, and then choose **Stop run**.
- The run status changes to **Stopping**.
5. (Optional) Choose the workflow run, choose **View run details**, and review the run graph.

To stop a workflow run (AWS CLI)

- Enter the following command. Replace `<workflow-name>` with the name of the workflow and `<run-id>` with the run ID of the workflow run to stop.

```
aws glue stop-workflow-run --name <workflow-name> --run-id <run-id>
```

The following is an example of the **stop-workflow-run** command.

```
aws glue stop-workflow-run --name my-workflow --run-id  
wr_137b88917411d128081069901e4a80595d97f719282094b7f271d09576770354
```

Repairing and Resuming a Workflow Run

If one or more nodes (jobs or crawlers) in a workflow do not successfully complete, this means that the workflow only partially ran. After you find the root causes and make corrections, you can select one or more nodes to resume the workflow run from, and then resume the workflow run. The selected nodes and all nodes that are downstream from those nodes are then run.

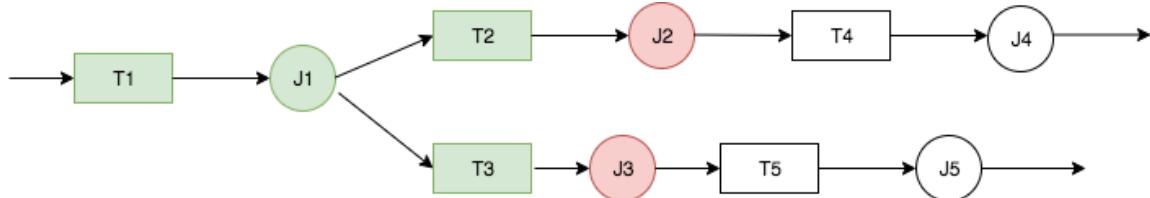
Topics

- [Resuming a Workflow Run: How It Works \(p. 412\)](#)
- [Resuming a Workflow Run \(p. 413\)](#)

- Notes and Limitations for Resuming Workflow Runs (p. 415)

Resuming a Workflow Run: How It Works

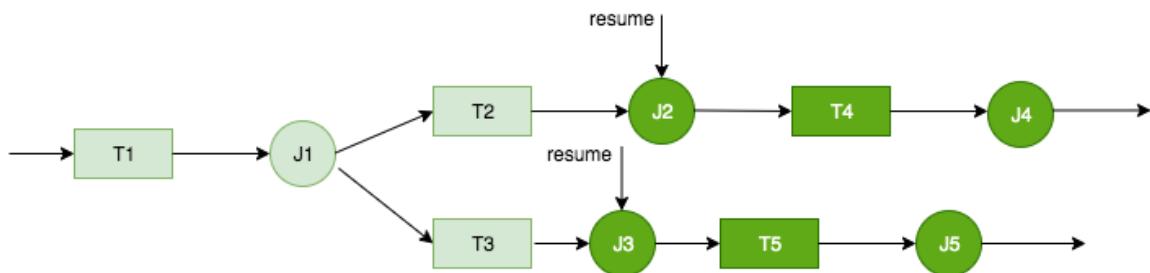
Consider the workflow W1 in the following diagram.



The workflow run proceeds as follows:

1. Trigger T1 starts job J1.
2. Successful completion of J1 fires triggers T2 and T3, which run jobs J2 and J3, respectively.
3. Jobs J2 and J3 fail.
4. Triggers T4 and T5 depend on the successful completion of J2 and J3, so they don't fire, and jobs J4 and J5 don't run. Workflow W1 is only partially run.

Now assume that the issues that caused J2 and J3 to fail are corrected. J2 and J3 are selected as the starting points to resume the workflow run from.



The workflow run resumes as follows:

1. Jobs J2 and J3 run successfully.
2. Triggers T4 and T5 fire.
3. Jobs J4 and J5 run successfully.

The resumed workflow run is tracked as a separate workflow run with a new run ID. When you view the workflow history, you can view the previous run ID for any workflow run. In the example in the following screenshot, the workflow run with run ID wr_c7a22... (the second row) had a node that did not complete. The user fixed the problem and resumed the workflow run, which resulted in run ID wr_a07e55... (the first row).

Run ID	Previous run ID	Run status	Execution time
wr_a07e55f2087afdd415a404403f644a4265278...	wr_c7a2219a8dc412f1366a5b30df3c58be30b9...	Completed	17 Minutes
wr_c7a2219a8dc412f1366a5b30df3c58be30b9...	-	Completed	8 Minutes

Note

For the rest of this discussion, the term "resumed workflow run" refers to the workflow run that was created when the previous workflow run was resumed. The "original workflow run" refers to the workflow run that only partially ran and that needed to be resumed.

Resumed Workflow Run Graph

In a resumed workflow run, although only a subset of nodes are run, the run graph is a complete graph. That is, the nodes that didn't run in the resumed workflow are copied from the run graph of the original workflow run. Copied job and crawler nodes that ran in the original workflow run include run details.

Consider again the workflow W1 in the previous diagram. When the workflow run is resumed starting with J2 and J3, the run graph for the resumed workflow run shows all jobs, J1 through J5, and all triggers, T1 through T5. The run details for J1 are copied from the original workflow run.

Workflow Run Snapshots

When a workflow run is started, AWS Glue takes a snapshot of the workflow design graph at that point in time. That snapshot is used for the duration of the workflow run. If you make changes to any triggers after the run starts, those changes don't affect the current workflow run. Snapshots ensure that workflow runs proceed in a consistent manner.

Snapshots make only triggers immutable. Changes that you make to downstream jobs and crawlers during the workflow run take effect for the current run.

Resuming a Workflow Run

Follow these steps to resume a workflow run. You can resume a workflow run by using the AWS Glue console, API, or AWS Command Line Interface (AWS CLI).

To resume a workflow run (console)

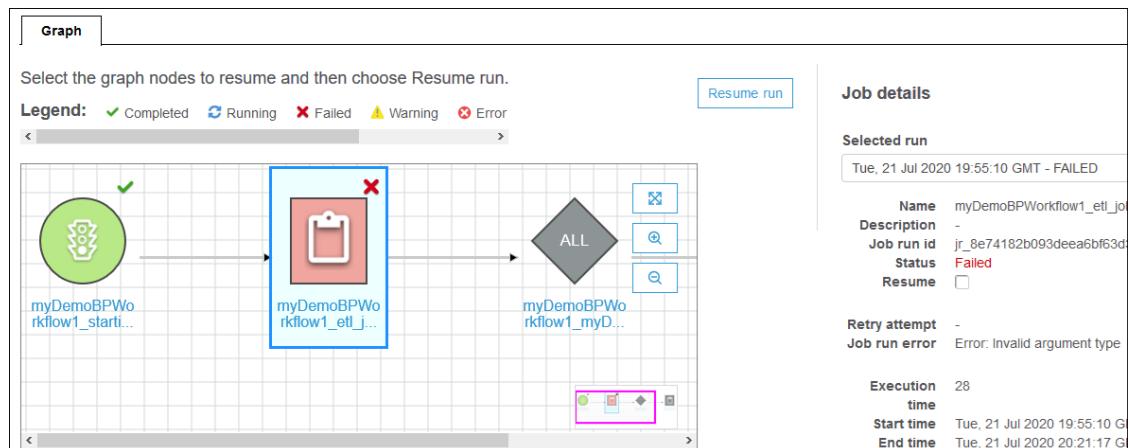
1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.

Sign in as a user who has permissions to view workflows and resume workflow runs.

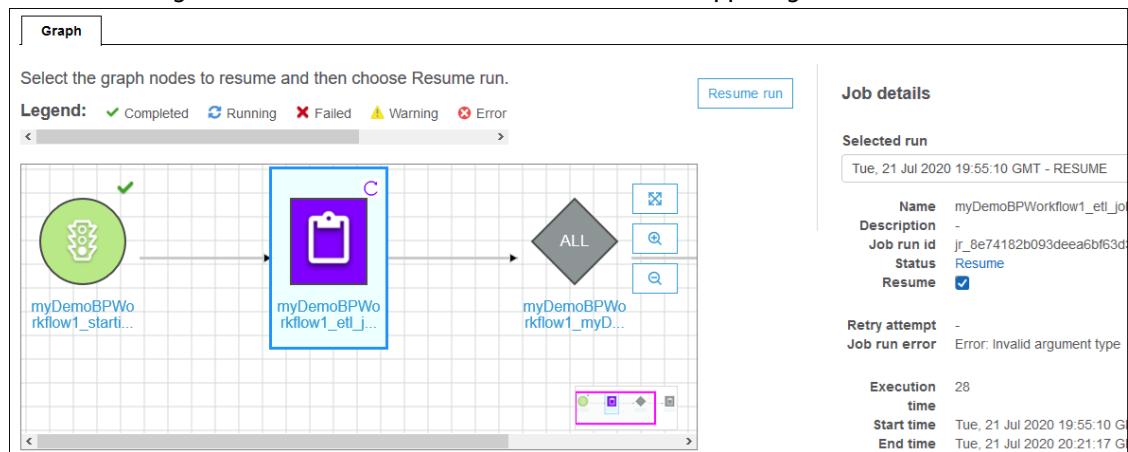
Note

To resume workflow runs, you need the `glue:ResumeWorkflowRun` AWS Identity and Access Management (IAM) permission.

2. In the navigation pane, choose **Workflows**.
3. Select a workflow, and then choose the **History** tab.
4. Select the workflow run that only partially ran, and then choose **View run details**.
5. In the run graph, select the first (or only) node that you want to restart and that you want to resume the workflow run from.
6. In the details pane to the right of the graph, select the **Resume** check box.



The node changes color and shows a small resume icon at the upper right.



7. Complete the previous two steps for any additional nodes to restart.
8. Choose **Resume run**.

To resume a workflow run (AWS CLI)

1. Ensure that you have the `glue:ResumeWorkflowRun` IAM permission.
2. Retrieve the node IDs for the nodes that you want to restart.
 - a. Run the `get-workflow-run` command for the original workflow run. Supply the workflow name and run ID, and add the `--include-graph` option, as shown in the following example. Get the run ID from the **History** tab on the console, or by running the `get-workflow` command.

```
aws glue get-workflow-run --name cloudtrailtest1 --run-id
  wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --include-
  graph
```

The command returns the nodes and edges of the graph as a large JSON object.

- b. Locate the nodes of interest by the `Type` and `Name` properties of the node objects.

The following is an example node object from the output.

```
{
    "Type": "JOB",
    "Name": "test1_post_failure_4592978",
    "UniqueId": "wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd",
    "JobDetails": {
        "JobRuns": [
            {
                "Id": "jr_690b9f7fc5cb399204bc542c6c956f39934496a5d665a42de891e5b01f59e613",
                "Attempt": 0,
                "TriggerName": "test1_aggregate_failure_649b2432",
                "JobName": "test1_post_failure_4592978",
                "StartedOn": 1595358275.375,
                "LastModifiedOn": 1595358298.785,
                "CompletedOn": 1595358298.785,
                "JobRunState": "FAILED",
                "PredecessorRuns": [],
                "AllocatedCapacity": 0,
                "ExecutionTime": 16,
                "Timeout": 2880,
                "MaxCapacity": 0.0625,
                "LogGroupName": "/aws-glue/python-jobs"
            }
        ]
    }
}
```

- c. Get the node ID from the `UniqueId` property of the node object.
3. Run the `resume-workflow-run` command. Provide the workflow name, run ID, and list of node IDs separated by spaces, as shown in the following example.

```
aws glue resume-workflow-run --name clouptrailtest1 --run-id
    wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --node-
    ids wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3
    wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd
```

The command outputs the run ID of the resumed (new) workflow run and a list of nodes that will be started.

```
{
    "RunId": "wr_2ada0d3209a262fc1156e4291134b3bd643491bcfb0ceead30bd3e4efac24de9",
    "NodeIds": [
        "wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3"
    ]
}
```

Note that although the example `resume-workflow-run` command listed two nodes to restart, the example output indicated that only one node would be restarted. This is because one node was downstream of the other node, and the downstream node would be restarted anyway by the normal flow of the workflow.

Notes and Limitations for Resuming Workflow Runs

Keep the following notes and limitations in mind when resuming workflow runs.

- You can resume a workflow run only if it's in the `COMPLETED` state.

Note

Even if one or more nodes in a workflow run don't complete, the workflow run state is shown as COMPLETED. Be sure to check the run graph to discover any nodes that didn't successfully complete.

- You can resume a workflow run from any job or crawler node that the original workflow run attempted to run. You can't resume a workflow run from a trigger node.
- Restarting a node does not reset its state. Any data that was partially processed is not rolled back.
- You can resume the same workflow run multiple times. If a resumed workflow run only partially runs, you can address the issue and resume the resumed run.
- If you select two nodes to restart and they're dependent upon each other, the upstream node is run before the downstream node. In fact, selecting the downstream node is redundant, because it will be run according to the normal flow of the workflow.

Getting and Setting Workflow Run Properties in AWS Glue

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 aren't 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 \(Python: get_workflow_run_properties\) \(p. 935\)](#)
- [PutWorkflowRunProperties Action \(Python: put_workflow_run_properties\) \(p. 935\)](#)

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 \(p. 921\)](#).

Topics

- [Querying Static Views \(p. 417\)](#)
- [Querying Dynamic Views \(p. 417\)](#)

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 `run`, 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 run, 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 based on the static view that was present when the workflow run was started.

```
Runtime 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
    }
  ]
}
```

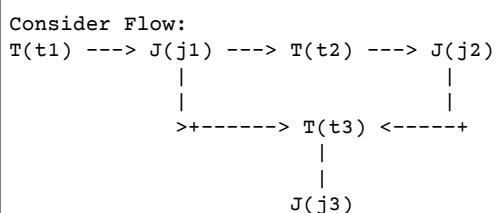
```

},
{
    "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).



Graph generated:

```
Nodes: t1, t2, t3, j1, j2, j3
Edges: (t1, j1), (j1, t2), (t2, j2), (j1, t3), (j2, t3), (t3, j3)
```

Blueprint and Workflow Restrictions in AWS Glue

The following are restrictions for blueprints and workflows.

Blueprint Restrictions

Keep the following blueprint restrictions in mind:

- The blueprint must be registered in the same AWS Region where the Amazon S3 bucket resides in.
- To share blueprints across AWS accounts you must give the read permissions on the blueprint ZIP archive in Amazon S3. Customers who have read permission on a blueprint ZIP archive can register the blueprint in their AWS account and use it.
- The set of blueprint parameters is stored as a single JSON object. The maximum length of this object is 128 KB.
- The maximum uncompressed size of the blueprint ZIP archive is 5 MB. The maximum compressed size is 1 MB.
- Limit the total number of jobs, crawlers, and triggers within a workflow to 100 or less. If you include more than 100, you might get errors when trying to resume or stop workflow runs.

Workflow Restrictions

Keep the following workflow restrictions in mind. Some of these comments are directed more at a user creating workflows manually.

- The maximum batch size for an Amazon EventBridge event trigger is 100. The maximum window size is 900 seconds (15 minutes).
- A trigger can be associated with only one workflow.
- Only one starting trigger (on-demand or schedule) is permitted.
- If a job or crawler in a workflow is started by a trigger that is outside the workflow, any triggers inside the workflow that depend on job or crawler completion (succeeded or otherwise) do not fire.
- Similarly, if a job or crawler in a workflow has triggers that depend on job or crawler completion (succeeded or otherwise) both within the workflow and outside the workflow, and if the job or crawler is started from within a workflow, only the triggers inside the workflow fire upon job or crawler completion.

Troubleshooting Blueprint errors in AWS Glue

If you encounter errors when using AWS Glue blueprints, use the following solutions to help you find the source of the problems and fix them.

Topics

- [Error: missing PySpark module \(p. 421\)](#)
- [Error: missing blueprint config file \(p. 421\)](#)
- [Error: missing imported file \(p. 422\)](#)
- [Error: not authorized to perform iamPassRole on resource \(p. 422\)](#)

- [Error: invalid cron schedule \(p. 422\)](#)
- [Error: a trigger with the same name already exists \(p. 422\)](#)
- [Error: Workflow with name: foo already exists. \(p. 423\)](#)
- [Error: module not found in specified layoutGenerator path \(p. 423\)](#)
- [Error: validation error in Connections field \(p. 423\)](#)

Error: missing PySpark module

AWS Glue returns the error "Unknown error executing layout generator function ModuleNotFoundError: No module named 'pyspark'".

When you unzip the blueprint archive it could be like either of the following:

```
$ unzip compaction.zip
Archive: compaction.zip
  creating: compaction/
  inflating: compaction/blueprint.cfg
  inflating: compaction/layout.py
  inflating: compaction/README.md
  inflating: compaction/compaction.py

$ unzip compaction.zip
Archive: compaction.zip
  inflating: blueprint.cfg
  inflating: compaction.py
  inflating: layout.py
  inflating: README.md
```

In the first case, all the files related to the blueprint were placed under a folder named `compaction` and it was then converted into a zip file named `compaction.zip`.

In the second case, all the files required for the blueprint were not included into a folder and were added as root files under the zip file `compaction.zip`.

Creating a file in either of the above formats is allowed. However make sure that `blueprint.cfg` has the correct path to the name of the function in the script that generates the layout.

Examples

In case 1: `blueprint.cfg` should have `layoutGenerator` as the following:

```
layoutGenerator": "compaction.layout.generate_layout"
```

In case 2: `blueprint.cfg` should have `layoutGenerator` as the following

```
layoutGenerator": "layout.generate_layout"
```

If this path is not included correctly, you could see an error as indicated. For example, if you have the folder structure as mentioned in case 2 and you have the `layoutGenerator` indicated as in case 1, you can see the above error.

Error: missing blueprint config file

AWS Glue returns the error "Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '/tmp/compaction/blueprint.cfg".

The blueprint.cfg should be placed at the root level of the ZIP archive or within a folder which has the same name as the ZIP archive.

When we extract the blueprint ZIP archive, blueprint.cfg is expected to be found in one of the following paths. If it is not found in one of the following paths, you can see the above error.

```
$ unzip compaction.zip
Archive: compaction.zip
  creating: compaction/
  inflating: compaction/blueprint.cfg

$ unzip compaction.zip
Archive: compaction.zip
  inflating: blueprint.cfg
```

Error: missing imported file

AWS Glue returns the error "Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '*' demo-project/foo.py'".

If your layout generation script has functionality to read other files, make sure you give a full path for the file to be imported. For example, the Conversion.py script may be referenced in Layout.py. For more information, see [Sample Blueprint Project](#).

Error: not authorized to perform iamPassRole on resource

AWS Glue returns the error "User: arn:aws:sts::123456789012:assumed-role/AWSGlueServiceRole/ GlueSession is not authorized to perform: iam:PassRole on resource: arn:aws:iam::123456789012:role/ AWSGlueServiceRole"

If the jobs and crawlers in the workflow assume the same role as the role passed to create workflow from the blueprint, then the blueprint role needs to include the `iam:PassRole` permission on itself.

If the jobs and crawlers in the workflow assume a role other than the role passed to create the entities of the workflow from the blueprint, then the blueprint role needs to include the `iam:PassRole` permission on that other role instead of on the blueprint role.

For more information, see [Permissions for Blueprint Roles](#).

Error: invalid cron schedule

AWS Glue returns the error "The schedule cron(0 0 * * *) is invalid."

Provide a valid `cron` expression. For more information, see [Time-Based Schedules for Jobs and Crawlers](#).

Error: a trigger with the same name already exists

AWS Glue returns the error "Trigger with name 'foo_starting_trigger' already submitted with different configuration".

A blueprint does not require you to define triggers in the layout script for workflow creation. Trigger creation is managed by the blueprint library based on the dependencies defined between two actions.

The naming for the triggers is as follows:

- For the starting trigger in the workflow the naming is <workflow_name>_starting_trigger.

- For a node(job/crawler) in the workflow that depends on the completion of either one or multiple upstream nodes; AWS Glue defines a trigger with the name <workflow_name>_<node_name>_trigger

This error means a trigger with same name already exists. You can delete the existing trigger and re-run the workflow creation.

Note

Deleting a workflow doesn't delete the nodes within the workflow. It is possible that though the workflow is deleted, triggers are left behind. Due to this, you may not receive a 'workflow already exists' error, but you may receive a 'trigger already exists' error in a case where you create a workflow, delete it and then try to re-create it with the same name from same blueprint.

Error: Workflow with name: foo already exists.

The workflow name should be unique. Please try with a different name.

Error: module not found in specified layoutGenerator path

AWS Glue returns the error "Unknown error executing layout generator function ModuleNotFoundError: No module named 'crawl_s3_locations'".

```
layoutGenerator": "crawl_s3_locations.layout.generate_layout"
```

For example, if you have the above layoutGenerator path, then when you unzip the blueprint archive, it needs to look like the following:

```
$ unzip crawl_s3_locations.zip
Archive: crawl_s3_locations.zip
  creating: crawl_s3_locations/
  inflating: crawl_s3_locations/blueprint.cfg
  inflating: crawl_s3_locations/layout.py
  inflating: crawl_s3_locations/README.md
```

When you unzip the archive, if the blueprint archive looks like the following, then you can get the above error.

```
$ unzip crawl_s3_locations.zip
Archive: crawl_s3_locations.zip
  inflating: blueprint.cfg
  inflating: layout.py
  inflating: README.md
```

You can see that there is no folder named crawl_s3_locations and when the layoutGenerator path refers to the layout file via the module crawl_s3_locations, you can get the above error.

Error: validation error in Connections field

AWS Glue returns the error "Unknown error executing layout generator function TypeError: Value ['foo'] for key Connections should be of type <class 'dict'>!".

This is a validation error. The Connections field in the Job class is expecting a dictionary and instead a list of values are provided causing the error.

```
User input was list of values
Connections= ['string']

Should be a dict like the following
Connections*= {'Connections': ['string']}
```

To avoid these run time errors while creating a workflow from a blueprint, you can validate the workflow, job and crawler definitions as outlined in [Testing a Blueprint](#).

Refer to the syntax in [AWS Glue Blueprint Classes Reference](#) for defining the AWS Glue job, crawler and workflow in the layout script.

Permissions for Personas and Roles for AWS Glue Blueprints

The following are the typical personas and suggested AWS Identity and Access Management (IAM) permissions policies for personas and roles for AWS Glue blueprints.

Topics

- [Blueprint Personas \(p. 424\)](#)
- [Permissions for Blueprint Personas \(p. 424\)](#)
- [Permissions for Blueprint Roles \(p. 426\)](#)

Blueprint Personas

The following are the personas typically involved in the lifecycle of AWS Glue blueprints.

Persona	Description
AWS Glue developer	Develops, tests, and publishes blueprints.
AWS Glue administrator	Registers, maintains, and grants permissions on blueprints.
Data analyst	Runs blueprints to create workflows.

For more information, see [the section called “Overview of Blueprints” \(p. 376\)](#).

Permissions for Blueprint Personas

The following are the suggested permissions for each blueprint persona.

AWS Glue Developer Permissions for Blueprints

The AWS Glue developer must have write permissions on the Amazon S3 bucket that is used to publish the blueprint. Often, the developer registers the blueprint after uploading it. In that case, the developer needs the permissions listed in [the section called “AWS Glue Administrator Permissions for Blueprints” \(p. 425\)](#). Additionally, if the developer wishes to test the blueprint after its registered, he or she also needs the permissions listed in [the section called “Data Analyst Permissions for Blueprints” \(p. 425\)](#).

AWS Glue Administrator Permissions for Blueprints

The following policy grants permissions to register, view, and maintain AWS Glue blueprints.

Important

In the following policy, replace `<s3-bucket-name>` and `<prefix>` with the Amazon S3 path to uploaded blueprint ZIP archives to register.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue>CreateBlueprint",  
                "glue:UpdateBlueprint",  
                "glue>DeleteBlueprint",  
                "glue:GetBlueprint",  
                "glue>ListBlueprints",  
                "glue:BatchGetBlueprints"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::<s3-bucket-name>/<prefix>/*"  
        }  
    ]  
}
```

Data Analyst Permissions for Blueprints

The following policy grants permissions to run blueprints and to view the resulting workflow and workflow components. It also grants `PassRole` on the role that AWS Glue assumes to create the workflow and workflow components.

The policy grants permissions on any resource. If you want to configure fine-grained access to individual blueprints, use the following format for blueprint ARNs:

`arn:aws:glue:<region>:<account-id>:blueprint/<blueprint-name>`

Important

In the following policy, replace `<account-id>` with a valid AWS account and replace `<role-name>` with the name of the role used to run a blueprint. See [the section called “Permissions for Blueprint Roles” \(p. 426\)](#) for the permissions that this role requires.

```

        "glue:GetCrawler",
        "glue>ListTriggers",
        "glue>ListJobs",
        "glue>BatchGetCrawlers",
        "glue>GetTrigger",
        "glue>BatchGetWorkflows",
        "glue>BatchGetTriggers",
        "glue>BatchGetJobs",
        "glue>BatchGetBlueprints",
        "glue>GetWorkflowRun",
        "glue>GetWorkflowRuns",
        "glue>ListCrawlers",
        "glue>ListWorkflows",
        "glue>GetJob",
        "glue>GetWorkflow",
        "glue>StartWorkflowRun"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam:<account-id>:role/<role-name>"
}
]
}

```

Permissions for Blueprint Roles

The following are the suggested permissions for the IAM role used to create a workflow from a blueprint. The role has to have a trust relationship with `glue.amazonaws.com`.

Important

In the following policy, replace `<account-id>` with a valid AWS account, and replace `<role-name>` with the name of the role.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue>CreateJob",
                "glue>GetCrawler",
                "glue>GetTrigger",
                "glue>DeleteCrawler",
                "glue>CreateTrigger",
                "glue>DeleteTrigger",
                "glue>DeleteJob",
                "glue>CreateWorkflow",
                "glue>DeleteWorkflow",
                "glue>GetJob",
                "glue>GetWorkflow",
                "glue>CreateCrawler"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam:<account-id>:role/<role-name>"
        }
    ]
}

```

}

Note

If the jobs and crawlers in the workflow assume a role other than this role, this policy must include the `iam:PassRole` permission on that other role instead of on the blueprint role.

AWS Glue Schema Registry

The AWS Glue Schema Registry is a new feature that allows you to centrally discover, control, and evolve data stream schemas. A *schema* defines the structure and format of a data record. With AWS Glue Schema Registry, you can manage and enforce schemas on your data streaming applications using convenient integrations with Apache Kafka, [Amazon Managed Streaming for Apache Kafka](#), [Amazon Kinesis Data Streams](#), [Amazon Kinesis Data Analytics for Apache Flink](#), and [AWS Lambda](#).

The AWS Glue Schema Registry supports AVRO (v1.10.2) data format, JSON Data format with [JSON Schema format](#) for the schema (specifications Draft-04, Draft-06, and Draft-07) with JSON schema validation using the [Everit library](#), Protocol Buffers (Protobuf) versions proto2 and proto3 without support for extensions or groups, and Java language support, with other data formats and languages to come. Apache Kafka Connector support is not currently available for Protobuf schemas. Supported features include compatibility, schema sourcing via metadata, auto-registration of schemas, IAM compatibility, and optional ZLIB compression to reduce storage and data transfer. AWS Glue Schema Registry is serverless and free to use.

Using a schema as a data format contract between producers and consumers leads to improved data governance, higher quality data, and enables data consumers to be resilient to compatible upstream changes.

The Schema Registry allows disparate systems to share a schema for serialization and de-serialization. For example, assume you have a producer and consumer of data. The producer knows the schema when it publishes the data. The Schema Registry supplies a serializer and deserializer for certain systems such as Amazon MSK or Apache Kafka.

For more information, see [How the Schema Registry Works \(p. 434\)](#).

Topics

- [Schemas \(p. 428\)](#)
- [Registries \(p. 430\)](#)
- [Schema Versioning and Compatibility \(p. 431\)](#)
- [Open Source Serde Libraries \(p. 434\)](#)
- [Quotas of the Schema Registry \(p. 434\)](#)
- [How the Schema Registry Works \(p. 434\)](#)
- [Getting Started with Schema Registry \(p. 436\)](#)
- [Integrating with AWS Glue Schema Registry \(p. 451\)](#)

Schemas

A *schema* defines the structure and format of a data record. A schema is a versioned specification for reliable data publication, consumption, or storage.

In this example schema for Avro, the format and structure are defined by the layout and field names, and the format of the field names is defined by the data types (e.g., `string`, `int`).

```
{  
  "type": "record",  
  "namespace": "ABC_Organization",  
  "name": "Employee",  
  "fields": [
```

```

"fields": [
  {
    "name": "Name",
    "type": "string"
  },
  {
    "name": "Age",
    "type": "int"
  },
  {
    "name": "address",
    "type": {
      "type": "record",
      "name": "addressRecord",
      "fields": [
        {
          "name": "street",
          "type": "string"
        },
        {
          "name": "zipcode",
          "type": "int"
        }
      ]
    }
  }
]
}

```

In this example JSON Schema Draft-07 for JSON, the format is defined by the [JSON Schema organization](#).

```
{
  "$id": "https://example.com/person.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    }
  }
}
```

In this example for Protobuf, the format is defined by the [version 2 of the Protocol Buffers language \(proto2\)](#).

```

syntax = "proto2";

package tutorial;

option java_multiple_files = true;
option java_package = "com.example.tutorial.protos";

```

```

option java_outer_classname = "AddressBookProtos";

message Person {
    optional string name = 1;
    optional int32 id = 2;
    optional string email = 3;

    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }

    message PhoneNumber {
        optional string number = 1;
        optional PhoneType type = 2 [default = HOME];
    }

    repeated PhoneNumber phones = 4;
}

message AddressBook {
    repeated Person people = 1;
}

```

Registries

A *registry* is a logical container of schemas. Registries allow you to organize your schemas, as well as manage access control for your applications. A registry has an Amazon Resource Name (ARN) to allow you to organize and set different access permissions to schema operations within the registry.

You may use the default registry or create as many new registries as necessary.

AWS Glue Schema Registry Hierarchy

- RegistryName: [string]
 - RegistryArn: [AWS ARN]
 - CreatedTime: [timestamp]
 - UpdatedTime: [timestamp]
- SchemaName: [string]
 - SchemaArn: [AWS ARN]
 - DataFormat: [Avro, Json, or Protobuf]
 - Compatibility: [eg. BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL, NONE, DISABLED]
 - Status: [eg. PENDING, AVAILABLE, DELETING]
 - SchemaCheckpoint: [integer]
 - CreatedTime: [timestamp]
 - UpdatedTime: [timestamp]
- SchemaVersion: [string]
 - SchemaVersionNumber: [integer]
 - Status: [eg. PENDING, AVAILABLE, DELETING, FAILURE]
 - SchemaDefinition: [string, Value: JSON]
 - CreatedTime: [timestamp]

- SchemaVersionMetadata: [list]
 - MetadataKey: [string]
 - MetadataInfo
 - MetadataValue: [string]
 - CreatedTime: [timestamp]

Schema Versioning and Compatibility

Each schema can have multiple versions. Versioning is governed by a compatibility rule that is applied on a schema. Requests to register new schema versions are checked against this rule by the Schema Registry before they can succeed.

A schema version that is marked as a checkpoint is used to determine the compatibility of registering new versions of a schema. When a schema first gets created the default checkpoint will be the first version. As the schema evolves with more versions, you can use the CLI/SDK to change the checkpoint to a version of a schema using the `UpdateSchema` API that adheres to a set of constraints. In the console, editing the schema definition or compatibility mode will change the checkpoint to the latest version by default.

Compatibility modes allow you to control how schemas can or cannot evolve over time. These modes form the contract between applications producing and consuming data. When a new version of a schema is submitted to the registry, the compatibility rule applied to the schema name is used to determine if the new version can be accepted. There are 8 compatibility modes: NONE, DISABLED, BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL.

In the Avro data format, fields may be optional or required. An optional field is one in which the `Type` includes null. Required fields do not have null as the `Type`.

In the Protobuf data format, fields can be optional (including repeated) or required in proto2 syntax, while all fields are optional (including repeated) in proto3 syntax. All compatibility rules are determined based on the understanding of the Protocol Buffers specifications as well as the guidance from the [Google Protocol Buffers documentation](#).

- **NONE:** No compatibility mode applies. You can use this choice in development scenarios or if you do not know the compatibility modes that you want to apply to schemas. Any new version added will be accepted without undergoing a compatibility check.
- **DISABLED:** This compatibility choice prevents versioning for a particular schema. No new versions can be added.
- **BACKWARD:** This compatibility choice is recommended because it allows consumers to read both the current and the previous schema version. You can use this choice to check compatibility against the previous schema version when you delete fields or add optional fields. A typical use case for BACKWARD is when your application has been created for the most recent schema.

AVRO

For example, assume you have a schema defined by first name (required), last name (required), email (required), and phone number (optional).

If your next schema version removes the required email field, this would successfully register. BACKWARD compatibility requires consumers to be able to read the current and previous schema version. Your consumers will be able to read the new schema as the extra email field from old messages is ignored.

If you have a proposed new schema version that adds a required field, for example, zip code, this would not successfully register with BACKWARD compatibility. Your consumers on the new version

would not be able to read old messages before the schema change, as they are missing the required zip code field. However, if the zip code field was set as optional in the new schema, then the proposed version would successfully register as consumers can read the old schema without the optional zip code field.

JSON

For example, assume you have a schema version defined by first name (optional), last name (optional), email (optional) and phone number (optional).

If your next schema version adds the optional phone number property, this would successfully register as long as the original schema version does not allow any additional properties by setting the `additionalProperties` field to false. BACKWARD compatibility requires consumers to be able to read the current and previous schema version. Your consumers will be able to read data produced with the original schema where phone number property does not exist.

If you have a proposed new schema version that adds the optional phone number property, this would not successfully register with BACKWARD compatibility when the original schema version sets the `additionalProperties` field to true, namely allowing any additional property. Your consumers on the new version would not be able to read old messages before the schema change, as they cannot read data with phone number property in a different type, for example string instead of number.

PROTOBUF

For example, assume you have a schema version defined by a Message Person with `first_name` (required), `last_name` (required), `email` (required), and `phone_number` (optional) fields under proto2 syntax.

Similar to AVRO scenarios, if your next schema version removes the required `email` field, this would successfully register. BACKWARD compatibility requires consumers to be able to read the current and previous schema version. Your consumers will be able to read the new schema as the extra `email` field from old messages is ignored.

If you have a proposed new schema version that adds a required field, for example, `zip_code`, this would not successfully register with BACKWARD compatibility. Your consumers on the new version would not be able to read old messages before the schema change, as they are missing the required `zip_code` field. However, if the `zip_code` field was set as optional in the new schema, then the proposed version would successfully register as consumers can read the old schema without the optional `zip_code` field.

In case of a gRPC use case, adding new RPC service or RPC method is a backward compatible change. For example, assume you have a schema version defined by an RPC service `MyService` with two RPC methods `Foo` and `Bar`.

If your next schema version adds a new RPC method called `Baz`, this would successfully register. Your consumers will be able to read data produced with the original schema according to BACKWARD compatibility since the newly added RPC method `Baz` is optional.

If you have a proposed new schema version that removes the existing RPC method `Foo`, this would not successfully register with BACKWARD compatibility. Your consumers on the new version would not be able to read old messages before the schema change, as they cannot understand and read data with the non-existent RPC method `Foo` in a gRPC application.

- **BACKWARD_ALL:** This compatibility choice allows consumers to read both the current and all previous schema versions. You can use this choice to check compatibility against all previous schema versions when you delete fields or add optional fields.
- **FORWARD:** This compatibility choice allows consumers to read both the current and the subsequent schema versions, but not necessarily later versions. You can use this choice to check compatibility against the last schema version when you add fields or delete optional fields. A typical use case for

FORWARD is when your application has been created for a previous schema and should be able to process a more recent schema.

AVRO

For example, assume you have a schema version defined by first name (required), last name (required), email (optional).

If you have a new schema version that adds a required field, e.g. phone number, this would successfully register. FORWARD compatibility requires consumers to be able to read data produced with the new schema by using the previous version.

If you have a proposed schema version that deletes the required first name field, this would not successfully register with FORWARD compatibility. Your consumers on the prior version would not be able to read the proposed schemas as they are missing the required first name field. However, if the first name field was originally optional, then the proposed new schema would successfully register as the consumers can read data based on the new schema that doesn't have the optional first name field.

JSON

For example, assume you have a schema version defined by first name (optional), last name (optional), email (optional) and phone number (optional).

If you have a new schema version that removes the optional phone number property, this would successfully register as long as the new schema version does not allow any additional properties by setting the `additionalProperties` field to false. FORWARD compatibility requires consumers to be able to read data produced with the new schema by using the previous version.

If you have a proposed schema version that deletes the optional phone number property, this would not successfully register with FORWARD compatibility when the new schema version sets the `additionalProperties` field to true, namely allowing any additional property. Your consumers on the prior version would not be able to read the proposed schemas as they could have phone number property in a different type, for example string instead of number.

PROTOBUF

For example, assume you have a schema version defined by a Message Person with `first_name` (required), `last_name` (required), `email` (optional) fields under proto2 syntax.

Similar to AVRO scenarios, if you have a new schema version that adds a required field, e.g. `phone_number`, this would successfully register. FORWARD compatibility requires consumers to be able to read data produced with the new schema by using the previous version.

If you have a proposed schema version that deletes the required `first_name` field, this would not successfully register with FORWARD compatibility. Your consumers on the prior version would not be able to read the proposed schemas as they are missing the required `first_name` field. However, if the `first_name` field was originally optional, then the proposed new schema would successfully register as the consumers can read data based on the new schema that doesn't have the optional `first_name` field.

In case of a gRPC use case, removing an RPC service or RPC method is a forward-compatible change. For example, assume you have a schema version defined by an RPC service `MyService` with two RPC methods `Foo` and `Bar`.

If your next schema version deletes the existing RPC method named `Foo`, this would successfully register according to FORWARD compatibility as the consumers can read data produced with the new schema by using the previous version. If you have a proposed new schema version that adds an RPC method `Baz`, this would not successfully register with FORWARD compatibility. Your consumers on the

prior version would not be able to read the proposed schemas as they are missing the RPC method Baz.

- **FORWARD_ALL:** This compatibility choice allows consumers to read data written by producers of any new registered schema. You can use this choice when you need to add fields or delete optional fields, and check compatibility against all previous schema versions.
- **FULL:** This compatibility choice allows consumers to read data written by producers using the previous or next version of the schema, but not earlier or later versions. You can use this choice to check compatibility against the last schema version when you add or remove optional fields.
- **FULL_ALL:** This compatibility choice allows consumers to read data written by producers using all previous schema versions. You can use this choice to check compatibility against all previous schema versions when you add or remove optional fields.

Open Source Serde Libraries

AWS provides open-source Serde libraries as a framework for serializing and deserializing data. The open source design of these libraries allows common open-source applications and frameworks to support these libraries in their projects.

For more details on how the Serde libraries work, see [How the Schema Registry Works \(p. 434\)](#).

Quotas of the Schema Registry

Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account. The following are soft limits for the Schema Registry in AWS Glue.

Registries

You can have up to 10 registries per AWS account per AWS Region.

SchemaVersion

You can have up to 1000 schema versions per AWS account per AWS Region.

Each new schema creates a new schema version, so you can theoretically have up to 1000 schemas per account per region, if each schema has only one version.

Schema version metadata key-value pairs

You can have up to 10 key-value pairs per SchemaVersion per AWS Region.

You can view or set the key-value metadata pairs using the [QuerySchemaVersionMetadata Action \(Python: query_schema_version_metadata\) \(p. 917\)](#) or [PutSchemaVersionMetadata Action \(Python: put_schema_version_metadata\) \(p. 916\)](#) APIs.

The following are hard limits for the Schema Registry in AWS Glue.

Schema payloads

There is a size limit of 170KB for schema payloads.

How the Schema Registry Works

This section describes how the serialization and deserialization processes in Schema Registry work.

1. Register a schema: If the schema doesn't already exist in the registry, the schema can be registered with a schema name equal to the name of the destination (e.g., test_topic, test_stream, prod_firehose)

or the producer can provide a custom name for the schema. Producers can also add key-value pairs to the schema as metadata, such as source: msk_kafka_topic_A, or apply AWS tags to schemas on schema creation. Once a schema is registered the Schema Registry returns the schema version ID to the serializer. If the schema exists but the serializer is using a new version that doesn't exist, the Schema Registry will check the schema reference a compatibility rule to ensure the new version is compatible before registering it as a new version.

There are two methods of registering a schema: manual registration and auto-registration. You can register a schema manually via the AWS Glue console or CLI/SDK.

When auto-registration is turned on in the serializer settings, automatic registration of the schema will be performed. If `REGISTRY_NAME` is not provided in the producer configurations, then auto-registration will register the new schema version under the default registry (default-registry). See [Installing SerDe Libraries \(p. 436\)](#) for information on specifying the auto-registration property.

2. Serializer validates data records against the schema: When the application producing data has registered its schema, the Schema Registry serializer validates the record being produced by the application is structured with the fields and data types matching a registered schema. If the schema of the record does not match a registered schema, the serializer will return an exception and the application will fail to deliver the record to the destination.

If no schema exists and if the schema name is not provided via the producer configurations, then the schema is created with the same name as the topic name (if Apache Kafka or Amazon MSK) or stream name (if Kinesis Data Streams).

Every record has a schema definition and data. The schema definition is queried against the existing schemas and versions in the Schema Registry.

By default, producers cache schema definitions and schema version IDs of registered schemas. If a record's schema version definition does not match what's available in cache, the producer will attempt to validate the schema with the Schema Registry. If the schema version is valid, then its version ID and definition will be cached locally on the producer.

You can adjust the default cache period (24 hours) within the optional producer properties in step #3 of [Installing SerDe Libraries \(p. 436\)](#).

3. Serialize and deliver records: If the record complies with the schema, the serializer decorates each record with the schema version ID, serializes the record based on the data format selected (AVRO, JSON, Protobuf, or other formats coming soon), compresses the record (optional producer configuration), and delivers it to the destination.
4. Consumers deserialize the data: Consumers reading this data use the Schema Registry deserializer library that parses the schema version ID from the record payload.
5. Deserializer may request the schema from the Schema Registry: If this is the first time the deserializer has seen records with a particular schema version ID, using the schema version ID the deserializer will request the schema from the Schema Registry and cache the schema locally on the consumer. If the Schema Registry cannot deserialize the record, the consumer can log the data from the record and move on, or halt the application.
6. The deserializer uses the schema to deserialize the record: When the deserializer retrieves the schema version ID from the Schema Registry, the deserializer decompresses the record (if record sent by producer is compressed) and uses the schema to deserialize the record. The application now processes the record.

Note

Encryption: Your clients communicate with the Schema Registry via API calls which encrypt data in-transit using TLS encryption over HTTPS. Schemas stored in the Schema Registry are always encrypted at rest using a service-managed AWS Key Management Service (AWS KMS) key.

Note

User Authorization: The Schema Registry supports identity-based IAM policies.

Getting Started with Schema Registry

The following sections provide an overview and walk you through setting up and using Schema Registry. For information about Schema Registry concepts and components, see [AWS Glue Schema Registry \(p. 428\)](#).

Topics

- [Installing SerDe Libraries \(p. 436\)](#)
- [Using AWS CLI for the AWS Glue Schema Registry APIs \(p. 437\)](#)
- [Creating a Registry \(p. 438\)](#)
- [Dealing with a Specific Record \(JAVA POJO\) for JSON \(p. 439\)](#)
- [Creating a Schema \(p. 440\)](#)
- [Updating a Schema or Registry \(p. 444\)](#)
- [Deleting a Schema or Registry \(p. 447\)](#)
- [IAM Examples for Serializers \(p. 449\)](#)
- [IAM Examples for Deserializers \(p. 450\)](#)
- [Private Connectivity using AWS PrivateLink \(p. 450\)](#)
- [Accessing Amazon CloudWatch Metrics \(p. 450\)](#)

Installing SerDe Libraries

Note

Prerequisites: Before completing the following steps, you will need to have a Amazon Managed Streaming for Apache Kafka (Amazon MSK) or Apache Kafka cluster running. Your producers and consumers need to be running on Java 8 or above.

The SerDe libraries provide a framework for serializing and deserializing data.

You will install the open source serializer for your applications producing data (collectively the "serializers"). The serializer handles serialization, compression, and the interaction with the Schema Registry. The serializer automatically extracts the schema from a record being written to a Schema Registry compatible destination, such as Amazon MSK. Likewise, you will install the open source deserializer on your applications consuming data.

To install the libraries on producers and consumers:

1. Inside both the producers' and consumers' pom.xml files, add this dependency via the code below:

```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-serde</artifactId>
    <version>1.1.5</version>
</dependency>
```

Alternatively, you can clone the [AWS Glue Schema Registry Github repository](#).

2. Setup your producers with these required properties:

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class.getName()); // Can replace StringSerializer.class.getName() with
    any other key serializer that you may use
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    GlueSchemaRegistryKafkaSerializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
```

```
properties.put(AWSSchemaRegistryConstants.DATA_FORMAT, "JSON"); // OR "AVRO"
```

If there are no existing schemas, then auto-registration needs to be turned on (next step). If you do have a schema that you would like to apply, then replace "my-schema" with your schema name. Also the "registry-name" has to be provided if schema auto-registration is off. If the schema is created under the "default-registry" then registry name can be omitted.

3. (Optional) Set any of these optional producer properties. For detailed property descriptions, see [the ReadMe file](#).

```
props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, "true"); // If not passed, uses "false"  
props.put(AWSSchemaRegistryConstants.SCHEMA_NAME, "my-schema"); // If not passed, uses transport name (topic name in case of Kafka, or stream name in case of Kinesis Data Streams)  
props.put(AWSSchemaRegistryConstants.REGISTRY_NAME, "my-registry"); // If not passed, uses "default-registry"  
props.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); // If not passed, uses 86400000 (24 Hours)  
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200  
props.put(AWSSchemaRegistryConstants.COMPATIBILITY_SETTING, Compatibility.FULL); // Pass a compatibility mode. If not passed, uses Compatibility.BACKWARD  
props.put(AWSSchemaRegistryConstants.DESCRIPTION, "This registry is used for several purposes."); // If not passed, constructs a description  
props.put(AWSSchemaRegistryConstants.COMPRESSION_TYPE,  
        AWSSchemaRegistryConstants.COMPRESSION.ZLIB); // If not passed, records are sent uncompressed
```

Auto-registration registers the schema version under the default registry ("default-registry"). If a SCHEMA_NAME is not specified in the previous step, then the topic name is inferred as SCHEMA_NAME.

See [Schema Versioning and Compatibility \(p. 431\)](#) for more information on compatibility modes.

4. Setup your consumers with these required properties:

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,  
        StringDeserializer.class.getName());  
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,  
        GlueSchemaRegistryKafkaDeserializer.class.getName());  
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2"); // Pass an AWS Region  
props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,  
        AvroRecordType.GENERIC_RECORD.getName()); // Only required for AVRO data format
```

5. (Optional) Set these optional consumer properties. For detailed property descriptions, see [the ReadMe file](#).

```
properties.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); // If not passed, uses 86400000  
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200  
props.put(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,  
        "com.amazonaws.services.schemaregistry.deserializers.external.ThirdPartyDeserializer"); // For migration fall back scenario
```

Using AWS CLI for the AWS Glue Schema Registry APIs

To use the AWS CLI for the AWS Glue Schema Registry APIs, make sure to update your AWS CLI to the latest version.

Creating a Registry

You may use the default registry or create as many new registries as necessary using the AWS Glue APIs or AWS Glue console.

AWS Glue APIs

You can use these steps to perform this task using the AWS Glue APIs.

To add a new registry, use the [CreateRegistry Action \(Python: create_registry\) \(p. 902\)](#) API. Specify `RegistryName` as the name of the registry to be created, with a max length of 255, containing only letters, numbers, hyphens, underscores, dollar signs, or hash marks.

Specify a `Description` as a string not more than 2048 bytes long, matching the [URI address multi-line string pattern](#).

Optionally, specify one or more `Tags` for your registry, as a map array of key-value pairs.

```
aws glue create-registry --registry-name registryName1 --description description
```

When your registry is created it is assigned an Amazon Resource Name (ARN), which you can view in the `RegistryArn` of the API response. Now that you've created a registry, create one or more schemas for that registry.

AWS Glue console

To add a new registry in the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Schema registries**.
3. Choose **Add registry**.
4. Enter a **Registry name** for the registry, consisting of letters, numbers, hyphens, or underscores. This name cannot be changed.
5. Enter a **Description** (optional) for the registry.
6. Optionally, apply one or more tags to your registry. Choose **Add new tag** and specify a **Tag key** and optionally a **Tag value**.
7. Choose **Add registry**.

AWS Glue

- Data catalog
- Databases
- Tables
- Connections
- Crawlers
- Classifiers
- Schema registries**
- Schemas
- Settings

ETL

- AWS Glue Studio **New**
- Workflows
- Jobs
- ML Transforms
- Triggers
- Dev endpoints
- Notebooks

Schema registries > Add registry

Add a new schema registry

Add a schema registry to store one or multiple new related schemas.

Registry name
Name can't be changed post creation.

Only letters (A-Z), numbers (0-9), hyphens (-), underscores (_), dollar signs (\$), or hash marks (#) allowed. 255 characters maximum.

Description - optional

2048 characters maximum.

Registry tags - optional
No tags defined.
[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Add registry](#)

When your registry is created it is assigned an Amazon Resource Name (ARN), which you can view by choosing the registry from the list in **Schema registries**. Now that you've created a registry, create one or more schemas for that registry.

Dealing with a Specific Record (JAVA POJO) for JSON

You can use a plain old Java object (POJO) and pass the object as a record. This is similar to the notion of a specific record in AVRO. The [mbknor-jackson-jsonschema](#) can generate a JSON schema for the POJO passed. This library can also inject additional information in the JSON schema.

The AWS Glue Schema Registry library uses the injected "className" field in schema to provide a fully classified class name. The "className" field is used by the deserializer to deserialize into an object of that class.

```

Example class :

@JsonSchemaDescription("This is a car")
@JsonSchemaTitle("Simple Car Schema")
@Builder
@AllArgsConstructor
@EqualsAndHashCode
// Fully qualified class name to be added to an additionally injected property
// called className for deserializer to determine which class to deserialize
// the bytes into
@JsonSchemaInject(
    strings = {@JsonSchemaString(path = "className",
        value =
        "com.amazonaws.services.schemaregistry.integrationtests.generators.Car")})
// List of annotations to help infer JSON Schema are defined by https://github.com/mbknor/
mbknor-jackson-jsonSchema
  
```

```
public class Car {  
    @JsonProperty(required = true)  
    private String make;  
  
    @JsonProperty(required = true)  
    private String model;  
  
    @JsonSchemaDefault("true")  
    @JsonProperty  
    public boolean used;  
  
    @JsonSchemaInject(ints = {@JsonSchemaInt(path = "multipleOf", value = 1000)})  
    @Max(200000)  
    @JsonProperty  
    private int miles;  
  
    @Min(2000)  
    @JsonProperty  
    private int year;  
  
    @JsonProperty  
    private Date purchaseDate;  
  
    @JsonProperty  
    @JsonFormat(shape = JsonFormat.Shape.NUMBER)  
    private Date listedDate;  
  
    @JsonProperty  
    private String[] owners;  
  
    @JsonProperty  
    private Collection<Float> serviceChecks;  
  
    // Empty constructor is required by Jackson to deserialize bytes  
    // into an Object of this class  
    public Car() {}  
}
```

Creating a Schema

You can create a schema using the AWS Glue APIs or the AWS Glue console.

AWS Glue APIs

You can use these steps to perform this task using the AWS Glue APIs.

To add a new schema, use the [CreateSchema Action \(Python: create_schema\) \(p. 903\)](#) API.

Specify a `RegistryId` structure to indicate a registry for the schema. Or, omit the `RegistryId` to use the default registry.

Specify a `SchemaName` consisting of letters, numbers, hyphens, or underscores, and `DataFormat` as **AVRO** or **JSON**. `DataFormat` once set on a schema is not changeable.

Specify a `Compatibility` mode:

- *Backward (recommended)* — Consumer can read both current and previous version.
- *Backward all* — Consumer can read current and all previous versions.
- *Forward* — Consumer can read both current and subsequent version.
- *Forward all* — Consumer can read both current and all subsequent versions.

- *Full* — Combination of Backward and Forward.
- *Full all* — Combination of Backward all and Forward all.
- *None* — No compatibility checks are performed.
- *Disabled* — Prevent any versioning for this schema.

Optionally, specify Tags for your schema.

Specify a SchemaDefinition to define the schema in Avro, JSON, or Protobuf data format. See the examples.

For Avro data format:

```
aws glue create-schema --registry-id RegistryName="registryName1" --schema-name testschema
--compatibility NONE --data-format AVRO --schema-definition "{\"type\": \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\": \"string\"} ]}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1" --schema-name testschema --compatibility NONE
--data-format AVRO --schema-definition "{\"type\": \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\": \"string\"} ]}"
```

For JSON data format:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name testSchemaJson
--compatibility NONE --data-format JSON --schema-definition "{$schema\": \"http://json-schema.org/draft-07/schema#\", \"type\": \"object\", \"properties\": {\"f1\": {\"type\": \"string\"}}}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName" --schema-name testSchemaJson --compatibility NONE
--data-format JSON --schema-definition "{$schema\": \"http://json-schema.org/draft-07/schema#\", \"type\": \"object\", \"properties\": {\"f1\": {\"type\": \"string\"}}}"
```

For Protobuf data format:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name testSchemaProtobuf
--compatibility NONE --data-format PROTOBUF --schema-definition "syntax = \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName" --schema-name testSchemaProtobuf --
compatibility NONE --data-format PROTOBUF --schema-definition "syntax = \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

AWS Glue console

To add a new schema using the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Schemas**.
3. Choose **Add schema**.

4. Enter a **Schema name**, consisting of letters, numbers, hyphens, underscores, dollar signs, or hashmarks. This name cannot be changed.
5. Choose the **Registry** where the schema will be stored from the drop-down menu. The parent registry cannot be changed post-creation.
6. Leave the **Data format** as *Apache Avro* or *JSON*. This format applies to all versions of this schema.
7. Choose a **Compatibility mode**.
 - *Backward (recommended)* — receiver can read both current and previous versions.
 - *Backward All* — receiver can read current and all previous versions.
 - *Forward* — sender can write both current and previous versions.
 - *Forward All* — sender can write current and all previous versions.
 - *Full* — combination of Backward and Forward.
 - *Full All* — combination of Backward All and Forward All.
 - *None* — no compatibility checks performed.
 - *Disabled* — prevent any versioning for this schema.
8. Enter an optional **Description** for the registry of up to 250 characters.

The screenshot shows the AWS Glue 'Add a new schema' interface. The left sidebar lists various AWS Glue services and features. The 'Schemas' option is selected. The main right panel has the following sections:

- Schema name:** A text input field with placeholder "Enter a schema name...". Below it is a note: "Only letters (A-Z), numbers (0-9), hyphens (-), underscores (_), dollar signs (\$), or hash marks (#) allowed. 255 characters maximum."
- Registry:** A dropdown menu labeled "Select a registry..." with a blue arrow icon, and a button "Add new registry".
- Data format:** A dropdown menu set to "Apache Avro".
- Compatibility mode:** A dropdown menu set to "Backward (default) - consumer can read both current and previous version". A callout box for "Backward compatibility" provides a detailed explanation: "This compatibility choice allows consumers to read both the current and the previous schema version. This means that for instance, a new schema version cannot drop data fields or change the type of these fields, so they can't be read by consumers using the previous version."
- Description - optional:** A text input field with placeholder "Enter a schema description...". Below it is a note: "2048 characters maximum."

9. Optionally, apply one or more tags to your schema. Choose **Add new tag** and specify a **Tag key** and optionally a **Tag value**.

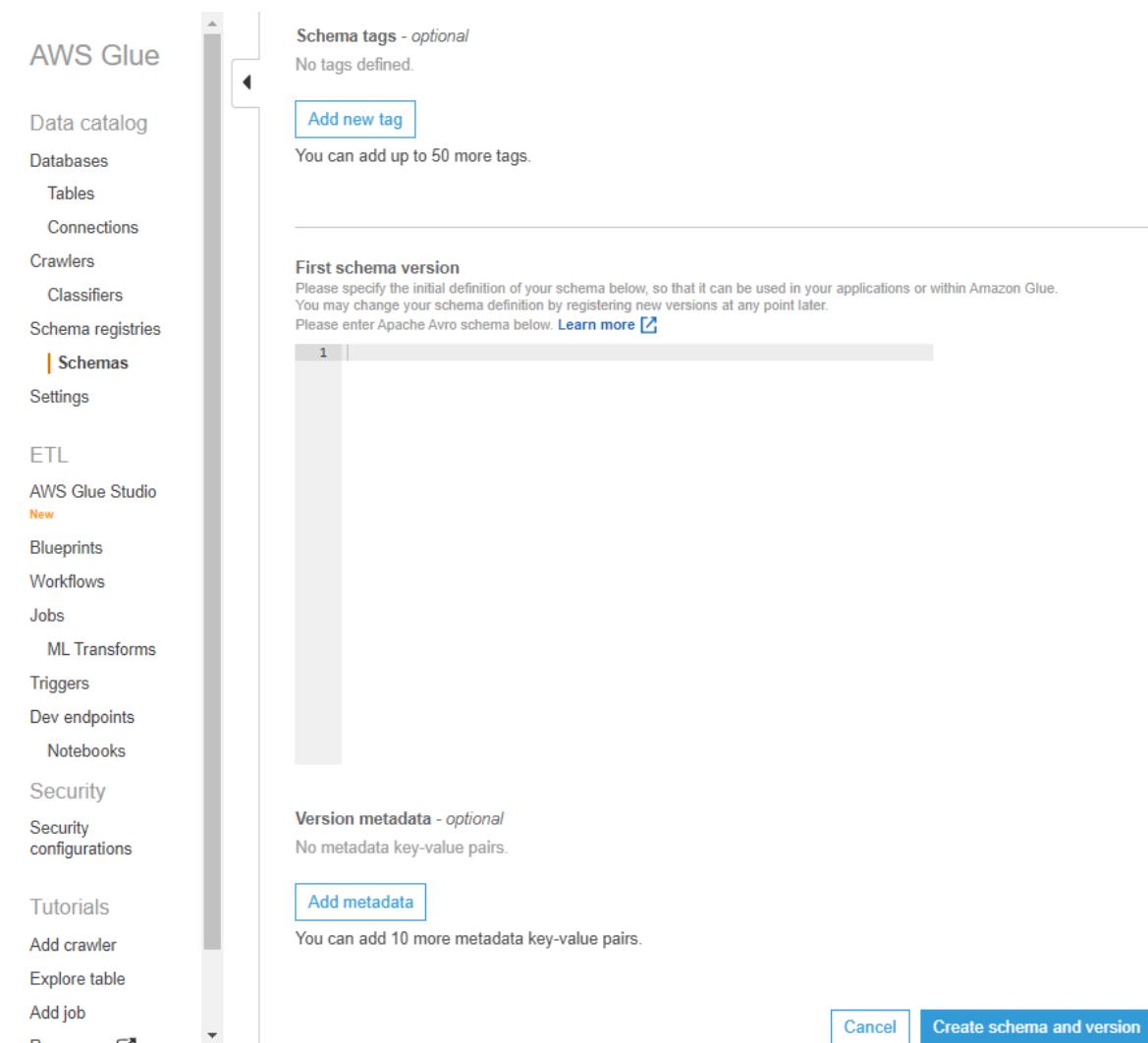
10 In the **First schema version** box, enter or paste your initial schema. . .

For Avro format, see [Working with Avro Data Format \(p. 443\)](#)

For JSON format, see [Working with JSON Data Format \(p. 444\)](#)

11 Optionally, choose **Add metadata** to add version metadata to annotate or classify your schema version.

12 Choose **Create schema and version**.



The schema is created and appears in the list under **Schemas**.

Working with Avro Data Format

Avro provides data serialization and data exchange services. Avro stores the data definition in JSON format making it easy to read and interpret. The data itself is stored in binary format.

For information on defining an Apache Avro schema, see the [Apache Avro specification](#).

Working with JSON Data Format

Data can be serialized with JSON format. [JSON Schema format](#) defines the standard for JSON Schema format.

Updating a Schema or Registry

Once created you can edit your schemas, schema versions, or registry.

Updating a Registry

You can update a registry using the AWS Glue APIs or the AWS Glue console. The name of an existing registry cannot be edited. You can edit the description for a registry.

AWS Glue APIs

To update an existing registry, use the [UpdateRegistry Action \(Python: update_registry\) \(p. 913\)](#) API.

Specify a `RegistryId` structure to indicate the registry that you want to update. Pass a `Description` to change the description for a registry.

```
aws glue update-registry --description updatedDescription --registry-id
  RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

AWS Glue console

To update a registry using the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Schema registries**.
3. Choose a registry from the the list of registries, by checking its box.
4. In the **Action** menu, choose **Edit registry**.

Updating a Schema

You can update the description or compatibility setting for a schema.

To update an existing schema, use the [UpdateSchema Action \(Python: update_schema\) \(p. 912\)](#) API.

Specify a `SchemaId` structure to indicate the schema that you want to update. One of `VersionNumber` or `Compatibility` has to be provided.

Code example 11:

```
aws glue update-schema --description testDescription --schema-id
  SchemaName="testSchema1",RegistryName="registryName1" --schema-version-number
  LatestVersion=true --compatibility NONE
```

```
aws glue update-schema --description testDescription --schema-id
  SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/testSchema1" --schema-
  version-number LatestVersion=true --compatibility NONE
```

Adding a Schema Version

When you add a schema version, you will need to compare the versions to make sure the new schema will be accepted.

To add a new version to an existing schema, use the [RegisterSchemaVersion Action \(Python: register_schema_version\) \(p. 911\)](#) API.

Specify a `SchemaId` structure to indicate the schema for which you want to add a version, and a `SchemaDefinition` to define the schema.

Code example 12:

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\": \"string\"} ]}" --schema-id SchemaArn="arn:aws:glue:us-east-1:901234567890:schema/registryName/testschema"
```

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\": \"string\"} ]}" --schema-id SchemaName="testschema",RegistryName="testregistry"
```

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 **Data catalog**, choose **Schemas**.
3. Choose the schema from the the list of schemas, by checking its box.
4. Choose one or more schemas from the list, by checking the boxes.
5. In the **Action** menu, choose **Register new version**.
6. In the **New version** box, enter or paste your new schema.
7. Choose **Compare with previous version** to see differences with the previous schema version.
8. Optionally, choose **Add metadata** to add version metadata to annotate or classify your schema version. Enter **Key** and optional **Value**.
9. Choose **Register version**.

Schemas > test-1 > Register version

Register a new schema version

Register version 4 to your schema.

Schema name	test-1
Data format	Apache Avro
Compatibility mode	Backward compatibility
Schema tags	No tags defined.

New Version 4

This is a copy of version 1's schema definition. A schema definition not associated with any existing schema versions must be defined in order to register a new schema version.

```

1  [
2   "type": "record",
3   "name": "r0",
4   "fields": [
5     {
6       "name": "f1",
7       "type": "int"
8     }
9   ]
10 ]

```

[Compare with previous version](#)

Version metadata - optional

No metadata key-value pairs.

[Add metadata](#)

You can add 10 more metadata key-value pairs.

[Cancel](#) [Register version](#)

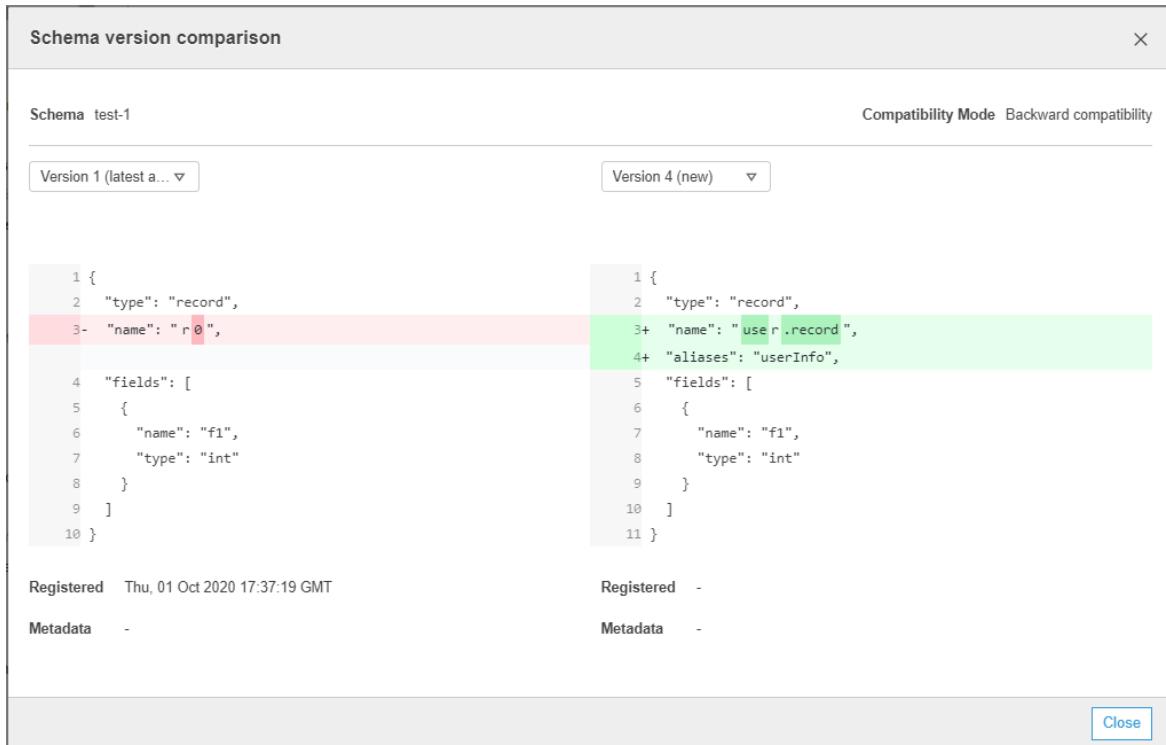
The schema(s) version appears in the list of versions. If the version changed the compatibility mode, the version will be marked as a checkpoint.

Example of a Schema Version Comparison

When you choose to **Compare with previous version**, you will see the previous and new versions displayed together. Changed information will be highlighted as follows:

- *Yellow*: indicates changed information.
- *Green*: indicates content added in the latest version.
- *Red*: indicates content removed in the latest version.

You can also compare against earlier versions.



Deleting a Schema or Registry

Deleting a schema, a schema version, or a registry are permanent actions that cannot be undone.

Deleting a Schema

You may want to delete a schema when it will no longer be used within a registry, using the AWS Management Console, or the [DeleteSchema Action \(Python: delete_schema\) \(p. 920\)](#) API.

Deleting one or more schemas is a permanent action that cannot be undone. Make sure that the schema or schemas are no longer needed.

To delete a schema from the registry, call the [DeleteSchema Action \(Python: delete_schema\) \(p. 920\)](#) API, specifying the `SchemaId` structure to identify the schema.

For example:

```
aws glue delete-schema --schema-id SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/schemaname"
```

```
aws glue delete-schema --schema-id SchemaName="TestSchema6-deleteschemabyname",RegistryName="default-registry"
```

AWS Glue console

To delete a schema from the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Schema registries**.

3. Choose the registry that contains your schema from the the list of registries.
4. Choose one or more schemas from the list, by checking the boxes.
5. In the **Action** menu, choose **Delete schema**.
6. Enter the text **Delete** in the field to confirm deletion.
7. Choose **Delete**.

The schema(s) you specified are deleted from the registry.

Deleting a Schema Version

As schemas accumulate in the registry, you may want to delete unwanted schema versions using the AWS Management Console, or the [DeleteSchemaVersions Action \(Python: delete_schema_versions\) \(p. 920\)](#) API. Deleting one or more schema versions is a permanent action that cannot be undone. Make sure that the schema versions are no longer needed.

When deleting schema versions, take note of the following constraints:

- You cannot delete a check-pointed version.
- The range of contiguous versions cannot be more than 25.
- The latest schema version must not be in a pending state.

Specify the SchemaId structure to identify the schema, and specify Versions as a range of versions to delete. For more information on specifying a version or range of versions, see [DeleteRegistry Action \(Python: delete_registry\) \(p. 919\)](#). The schema versions you specified are deleted from the registry.

Calling the [ListSchemaVersions Action \(Python: list_schema_versions\) \(p. 907\)](#) API after this call will list the status of the deleted versions.

For example:

```
aws glue delete-schema-versions --schema-id SchemaName="TestSchema6",RegistryName="default-registry" --versions "1-1"
```

```
aws glue delete-schema-versions --schema-id SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/default-registry/TestSchema6-NON-Existent" --versions "1-1"
```

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 **Data catalog**, choose **Schema registries**.
3. Choose the registry that contains your schema from the the list of registries.
4. Choose one or more schemas from the list, by checking the boxes.
5. In the **Action** menu, choose **Delete schema**.
6. Enter the text **Delete** in the field to confirm deletion.
7. Choose **Delete**.

The schema versions you specified are deleted from the registry.

Deleting a Registry

You may want to delete a registry when the schemas it contains should no longer be organized under that registry. You will need to reassign those schemas to another registry.

Deleting one or more registries is a permanent action that cannot be undone. Make sure that the registry or registries no longer needed.

The default registry can be deleted using the AWS CLI.

AWS Glue API

To delete the entire registry including the schema and all of its versions, call the [DeleteRegistry Action \(Python: delete_registry\) \(p. 919\)](#) API. Specify a `RegistryId` structure to identify the registry.

For example:

```
aws glue delete-registry --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

```
aws glue delete-registry --registry-id RegistryName="TestRegistry-deletebyname"
```

To get the status of the delete operation, you can call the `GetRegistry` API after the asynchronous call.

AWS Glue console

To delete a registry from the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Schema registries**.
3. Choose a registry from the list, by checking a box.
4. In the **Action** menu, choose **Delete registry**.
5. Enter the text **Delete** in the field to confirm deletion.
6. Choose **Delete**.

The registries you selected are deleted from AWS Glue.

IAM Examples for Serializers

Note

AWS managed policies grant necessary permissions for common use cases. For information on using managed policies to manage the schema registry, see [AWS Managed \(Predefined\) Policies for AWS Glue \(p. 66\)](#).

For serializers, you should create a minimal policy similar to that below to give you the ability to find the `schemaVersionId` for a given schema definition. Note, you should have read permissions on the registry in order to read the schemas in the registry. You can limit the registries that can be read by using the `Resource` clause.

Code example 13:

```
{  
    "Sid" : "GetSchemaByDefinition",  
    "Effect" : "Allow",  
    "Action" :  
    [  
        "glue:GetSchemaByDefinition"  
    ],  
    "Resource" : ["arn:aws:glue:us-east-2:012345678:registry/registryname-1",
```

```

    "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-1",
    "arn:aws:glue:us-east-2:012345678:schema/registryname-1/schemaname-2"
]
}

```

Further, you can also allow producers to create new schemas and versions by including the following extra methods. Note, you should be able to inspect the registry in order to add/remove/evolve the schemas inside it. You can limit the registries that can be inspected by using the Resource clause.

Code example 14:

```

{
    "Sid" : "RegisterSchemaWithMetadata",
    "Effect" : "Allow",
    "Action" :
    [
        "glue:GetSchemaByDefinition",
        "glue>CreateSchema",
        "glue:RegisterSchemaVersion",
        "glue:PutSchemaVersionMetadata",
    ],
    "Resource" : [ "arn:aws:glue:aws-region:123456789012:registry/registryname-1",
                    "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-1",
                    "arn:aws:glue:aws-region:123456789012:schema/registryname-1/schemaname-2"
                ]
}

```

IAM Examples for Deserializers

For deserializers (consumer side), you should create a policy similar to that below to allow the deserializer to fetch the schema from the Schema Registry for deserialization. Note, you should be able to inspect the registry in order to fetch the schemas inside it.

Code example 15:

```

{
    "Sid" : "GetSchemaVersion",
    "Effect" : "Allow",
    "Action" :
    [
        "glue:GetSchemaVersion"
    ],
    "Resource" : [ "*" ]
}

```

Private Connectivity using AWS PrivateLink

You can use AWS PrivateLink to connect your data producer's VPC to AWS Glue by defining an interface VPC endpoint for AWS Glue. When you use a VPC interface endpoint, communication between your VPC and AWS Glue is conducted entirely within the AWS network. For more information, see [Using AWS Glue with VPC Endpoints](#).

Accessing Amazon CloudWatch Metrics

Amazon CloudWatch metrics are available as part of CloudWatch's free tier. You can access these metrics in the CloudWatch Console. API-Level metrics include CreateSchema (Success and

Latency), GetSchemaByDefinition, (Success and Latency), GetSchemaVersion (Success and Latency), RegisterSchemaVersion (Success and Latency), PutSchemaVersionMetadata (Success and Latency). Resource-level metrics include Registry.ThrottledByLimit, SchemaVersion.ThrottledByLimit, SchemaVersion.Size.

Integrating with AWS Glue Schema Registry

These sections describe integrations with AWS Glue Schema Registry. The examples in these section show a schema with AVRO data format. For more examples, including schemas with JSON data format, see the integration tests and ReadMe information in the [AWS Glue Schema Registry open source repository](#).

Topics

- [Use Case: Connecting Schema Registry to Amazon MSK or Apache Kafka \(p. 451\)](#)
- [Use Case: Integrating Amazon Kinesis Data Streams with the AWS Glue Schema Registry \(p. 452\)](#)
- [Use Case: Amazon Kinesis Data Analytics for Apache Flink \(p. 459\)](#)
- [Use Case: Integration with AWS Lambda \(p. 463\)](#)
- [Use Case: AWS Glue Data Catalog \(p. 463\)](#)
- [Use case: AWS Glue Streaming \(p. 464\)](#)
- [Use Case: Apache Kafka Streams \(p. 465\)](#)
- [Use Case: Apache Kafka Connect \(p. 466\)](#)
- [Migration from a Third-Party Schema Registry to AWS Glue Schema Registry \(p. 469\)](#)
- [Sample AWS CloudFormation Template for Schema Registry \(p. 470\)](#)

Use Case: Connecting Schema Registry to Amazon MSK or Apache Kafka

Let's assume you are writing data to an Apache Kafka topic, and you can follow these steps to get started.

1. Create an Amazon Managed Streaming for Apache Kafka (Amazon MSK) or Apache Kafka cluster with at least one topic. If creating an Amazon MSK cluster, you can use the AWS Management Console. Follow these instructions: [Getting Started Using Amazon MSK](#) in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.
2. Follow the [Installing SerDe Libraries \(p. 436\)](#) step above.
3. To create schema registries, schemas, or schema versions, follow the instructions under the [Getting Started with Schema Registry \(p. 436\)](#) section of this document.
4. Start your producers and consumers to use the Schema Registry to write and read records to/from the Amazon MSK or Apache Kafka topic. Example producer and consumer code can be found in [the ReadMe file](#) from the Serde libraries. The Schema Registry library on the producer will automatically serialize the record and decorate the record with a schema version ID.
5. If the schema of this record has been inputted, or if auto-registration is turned on, then the schema will have been registered in the Schema Registry.
6. The consumer reading from the Amazon MSK or Apache Kafka topic, using the AWS Glue Schema Registry library, will automatically lookup the schema from the Schema Registry.

Use Case: Integrating Amazon Kinesis Data Streams with the AWS Glue Schema Registry

This integration requires that you have an existing Amazon Kinesis data stream. For more information, see [Getting Started with Amazon Kinesis Data Streams](#) in the *Amazon Kinesis Data Streams Developer Guide*.

There are two ways that you can interact with data in a Kinesis data stream.

- Through the Kinesis Producer Library (KPL) and Kinesis Client Library (KCL) libraries in Java. Multi-language support is not provided.
- Through the `PutRecords`, `PutRecord`, and `GetRecords` Kinesis Data Streams APIs available in the AWS SDK for Java.

If you currently use the KPL/KCL libraries, we recommend continuing to use that method. There are updated KCL and KPL versions with Schema Registry integrated, as shown in the examples. Otherwise, you can use the sample code to leverage the AWS Glue Schema Registry if using the KDS APIs directly.

Schema Registry integration is only available with KPL v0.14.2 or later and with KCL v2.3 or later. Schema Registry integration with JSON data format is available with KPL v0.14.8 or later and with KCL v2.3.6 or later.

Interacting with Data Using Kinesis SDK V2

This section describes interacting with Kinesis using Kinesis SDK V2

```
// Example JSON Record, you can construct a AVRO record also
private static final JsonDataWithSchema record = JsonDataWithSchema.builder(schemaString,
    payloadString);
private static final DataFormat dataFormat = DataFormat.JSON;

//Configurations for Schema Registry
GlueSchemaRegistryConfiguration gsrConfig = new GlueSchemaRegistryConfiguration("us-
east-1");

GlueSchemaRegistrySerializer glueSchemaRegistrySerializer =
    new GlueSchemaRegistrySerializerImpl(awsCredentialsProvider, gsrConfig);
GlueSchemaRegistryDataFormatsSerializer dataFormatSerializer =
    new GlueSchemaRegistrySerializerFactory().getInstance(dataFormat, gsrConfig);

Schema gsrSchema =
    new Schema(dataFormatSerializer.getSchemaDefinition(record), dataFormat.name(),
    "MySchema");

byte[] serializedBytes = dataFormatSerializer.serialize(record);

byte[] gsrEncodedBytes = glueSchemaRegistrySerializer.encode(streamName, gsrSchema,
    serializedBytes);

PutRecordRequest putRecordRequest = PutRecordRequest.builder()
    .streamName(streamName)
    .partitionKey("partitionKey")
    .data(SdkBytes.fromByteArray(gsrEncodedBytes))
    .build();
shardId = kinesisClient.putRecord(putRecordRequest)
    .get()
    .shardId();
```

```
GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer = new
    GlueSchemaRegistryDeserializerImpl(awsCredentialsProvider, gsrConfig);

SchemaRegistryDataFormatDeserializer gsrDataFormatDeserializer =
    glueSchemaRegistryDeserializerFactory.getInstance(dataFormat, gsrConfig);

GetShardIteratorRequest getShardIteratorRequest = GetShardIteratorRequest.builder()
    .streamName(streamName)
    .shardId(shardId)
    .shardIteratorType(ShardIteratorType.TRIM_HORIZON)
    .build();

String shardIterator = kinesisClient.getShardIterator(getShardIteratorRequest)
    .get()
    .shardIterator();

GetRecordsRequest getRecordRequest = GetRecordsRequest.builder()
    .shardIterator(shardIterator)
    .build();
GetRecordsResponse recordsResponse = kinesisClient.getRecords(getRecordRequest)
    .get();

List<Object> consumerRecords = new ArrayList<>();
List<Record> recordsFromKinesis = recordsResponse.records();

for (int i = 0; i < recordsFromKinesis.size(); i++) {
    byte[] consumedBytes = recordsFromKinesis.get(i)
        .data()
        .asByteArray();

    Schema gsrSchema = glueSchemaRegistryDeserializer.getSchema(consumedBytes);
    Object decodedRecord =
    gsrDataFormatDeserializer.deserialize(ByteBuffer.wrap(consumedBytes),
        gsrSchema.getSchemaDefinition());
        consumerRecords.add(decodedRecord);
}
```

Interacting with Data Using the KPL/KCL Libraries

This section describes integrating Kinesis Data Streams with Schema Registry using the KPL/KCL libraries. For more information on using KPL/KCL, see [Developing Producers Using the Amazon Kinesis Producer Library](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Setting up the Schema Registry in KPL

1. Define the schema definition for the data, data format and schema name authored in the AWS Glue Schema Registry.
2. Optionally configure the `GlueSchemaRegistryConfiguration` object.
3. Pass the schema object to the `addUserRecord` API.

```
private static final String SCHEMA_DEFINITION = "{"namespace": "example.avro",\n" + " \"type\": \"record\",\\n\" + \" \"name\": \"User\",\\n\" + \" \"fields\": [\\n\" + \" {\"name\": \"name\", \"type\": \"string\"},\\n\" + \" {\"name\": \"favorite_number\", \"type\": [\"int\", \"null\"]},\\n\" + \" {\"name\": \"favorite_color\", \"type\": [\"string\", \"null\"]}\\n\" + \" ]\\n\" + \"}";\n\nKinesisProducerConfiguration config = new KinesisProducerConfiguration();
```

```
config.setRegion("us-west-1")

// [Optional] configuration for Schema Registry.

GlueSchemaRegistryConfiguration schemaRegistryConfig =
new GlueSchemaRegistryConfiguration("us-west-1");

schemaRegistryConfig.setCompression(true);

config.setGlueSchemaRegistryConfiguration(schemaRegistryConfig);

// Optional configuration ends.

final KinesisProducer producer =
    new KinesisProducer(config);

final ByteBuffer data = getDataToSend();

com.amazonaws.services.schemaregistry.common.Schema gsrSchema =
new Schema(SCHEMA_DEFINITION, DataFormat.AVRO.toString(), "demoSchema");

ListenableFuture<UserRecordResult> f = producer.addUserRecord(
config.getStreamName(), TIMESTAMP, Utils.randomExplicitHashKey(), data, gsrSchema);

private static ByteBuffer getDataToSend() {
    org.apache.avro.Schema avroSchema =
        new org.apache.avro.Schema.Parser().parse(SCHEMA_DEFINITION);

    GenericRecord user = new GenericData.Record(avroSchema);
    user.put("name", "Emily");
    user.put("favorite_number", 32);
    user.put("favorite_color", "green");

    ByteArrayOutputStream outBytes = new ByteArrayOutputStream();
    Encoder encoder = EncoderFactory.get().directBinaryEncoder(outBytes, null);
    new GenericDatumWriter<>(avroSchema).write(user, encoder);
    encoder.flush();
    return ByteBuffer.wrap(outBytes.toByteArray());
}
```

Setting up the Kinesis Client Library

You will develop your Kinesis Client Library consumer in Java. For more information, see [Developing a Kinesis Client Library Consumer in Java](#) in the *Amazon Kinesis Data Streams Developer Guide*.

1. Create an instance of `GlueSchemaRegistryDeserializer` by passing a `GlueSchemaRegistryConfiguration` object.
2. Pass the `GlueSchemaRegistryDeserializer` to `retrievalConfig.glueSchemaRegistryDeserializer`.
3. Access the schema of incoming messages by calling `kinesisClientRecord.getSchema()`.

```
GlueSchemaRegistryConfiguration schemaRegistryConfig =
new GlueSchemaRegistryConfiguration(this.region.toString());

GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer =
new GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
schemaRegistryConfig);

RetrievalConfig retrievalConfig =
configsBuilder.retrievalConfig().retrievalSpecificConfig(new PollingConfig(streamName,
kinesisClient));
retrievalConfig.glueSchemaRegistryDeserializer(glueSchemaRegistryDeserializer);
```

```

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    retrievalConfig
);

public void processRecords(ProcessRecordsInput processRecordsInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Processing {} record(s)", processRecordsInput.records().size());
        processRecordsInput.records()
            .forEach(
                r ->
                    log.info("Processed record pk: {} -- Seq: {} : data {} with schema: {}",
                        r.partitionKey(),
                        r.sequenceNumber(), recordToAvroObj(r.toString(), r.getSchema())));
    } catch (Throwable t) {
        log.error("Caught throwable while processing records. Aborting.");
        Runtime.getRuntime().halt(1);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

private GenericRecord recordToAvroObj(KinesisClientRecord r) {
    byte[] data = new byte[r.data().remaining()];
    r.data().get(data, 0, data.length);
    org.apache.avro.Schema schema = new org.apache.avro.Schema.Parser().parse(r.schema().getSchemaDefinition());
    DatumReader datumReader = new GenericDatumReader<>(schema);

    BinaryDecoder binaryDecoder = DecoderFactory.get().binaryDecoder(data, 0,
        data.length, null);
    return (GenericRecord) datumReader.read(null, binaryDecoder);
}

```

Interacting with Data Using the Kinesis Data Streams APIs

This section describes integrating Kinesis Data Streams with Schema Registry using the Kinesis Data Streams APIs.

1. Update these Maven dependencies:

```

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-bom</artifactId>
            <version>1.11.884</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

```

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-kinesis</artifactId>
    </dependency>

    <dependency>
        <groupId>software.amazon.glue</groupId>
        <artifactId>schema-registry-serde</artifactId>
        <version>1.1.5</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-cbor</artifactId>
        <version>2.11.3</version>
    </dependency>
</dependencies>
```

2. In the producer, add schema header information using the PutRecords or PutRecord API in Kinesis Data Streams.

```
//The following lines add a Schema Header to the record
com.amazonaws.services.schemaregistry.common.Schema awsSchema =
    new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
DataFormat.AVRO.name(),
    schemaName);
GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
    new
GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(getConfigs()));
byte[] recordWithSchemaHeader =
    glueSchemaRegistrySerializer.encode(streamName, awsSchema, recordAsBytes);
```

3. In the producer, use the PutRecords or PutRecord API to put the record into the data stream.
4. In the consumer, remove the schema record from the header, and serialize an Avro schema record.

```
//The following lines remove Schema Header from record
GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
    new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
getConfigs());
byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);
com.amazonaws.services.schemaregistry.common.Schema awsSchema =
    glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);
byte[] record =
    glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

//The following lines serialize an AVRO schema record
if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
    Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
    Object genericRecord = convertBytesToRecord(avroSchema, record);
    System.out.println(genericRecord);
}
```

Interacting with Data Using the Kinesis Data Streams APIs

The following is example code for using the PutRecords and GetRecords APIs.

```
//Full sample code
import
com.amazonaws.services.schemaregistry.deserializers.GlueSchemaRegistryDeserializerImpl;
import com.amazonaws.services.schemaregistry.serializers.GlueSchemaRegistrySerializerImpl;
import com.amazonaws.services.schemaregistry.utils.AVROUtils;
import com.amazonaws.services.schemaregistry.utils.AWSGraphQLConstants;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericDatumWriter;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.Decoder;
import org.apache.avro.io.DecoderFactory;
import org.apache.avro.io.Encoder;
import org.apache.avro.io.EncoderFactory;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.glue.model.DataFormat;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class PutAndGetExampleWithEncodedData {
    static final String regionName = "us-east-2";
    static final String streamName = "testStream1";
    static final String schemaName = "User-Topic";
    static final String AVRO_USER_SCHEMA_FILE = "src/main/resources/user.avsc";
    KinesisApi kinesisApi = new KinesisApi();

    void runSampleForPutRecord() throws IOException {
        Object testRecord = getTestRecord();
        byte[] recordAsBytes = convertRecordToBytes(testRecord);
        String schemaDefinition = AVROUtils.getInstance().getSchemaDefinition(testRecord);

        //The following lines add a Schema Header to a record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
        DataFormat.AVRO.name(),
        schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
        GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
        GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema, recordAsBytes);

        //Use PutRecords api to pass a list of records
        kinesisApi.putRecords(Collections.singletonList(recordWithSchemaHeader),
        streamName, regionName);

        //OR
        //Use PutRecord api to pass single record
        //kinesisApi.putRecord(recordWithSchemaHeader, streamName, regionName);
    }
}
```

```
byte[] runSampleForGetRecord() throws IOException {
    ByteBuffer recordWithSchemaHeader = kinesisApi.getRecords(streamName, regionName);

    //The following lines remove the schema registry header
    GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
        new
    GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(), new
    GlueSchemaRegistryConfiguration(regionName));
    byte[] recordWithSchemaHeaderBytes = new byte[recordWithSchemaHeader.remaining()];
    recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
    recordWithSchemaHeaderBytes.length);

    com.amazonaws.services.schemaregistry.common.Schema awsSchema =
        glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);

    byte[] record =
    glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

    //The following lines serialize an AVRO schema record
    if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
        Schema avroSchema = new
    org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
        Object genericRecord = convertBytesToRecord(avroSchema, record);
        System.out.println(genericRecord);
    }

    return record;
}

private byte[] convertRecordToBytes(final Object record) throws IOException {
    ByteArrayOutputStream recordAsBytes = new ByteArrayOutputStream();
    Encoder encoder = EncoderFactory.get().directBinaryEncoder(recordAsBytes, null);
    GenericDatumWriter<Object> datumWriter = new
    GenericDatumWriter<>(AVROUtils.getInstance().getSchema(record));
    datumWriter.write(record, encoder);
    encoder.flush();
    return recordAsBytes.toByteArray();
}

private GenericRecord convertBytesToRecord(Schema avroSchema, byte[] record) throws
IOException {
    final GenericDatumReader<GenericRecord> datumReader = new
    GenericDatumReader<>(avroSchema);
    Decoder decoder = DecoderFactory.get().binaryDecoder(record, null);
    GenericRecord genericRecord = datumReader.read(null, decoder);
    return genericRecord;
}

private Map<String, String> getMetadata() {
    Map<String, String> metadata = new HashMap<>();
    metadata.put("event-source-1", "topic1");
    metadata.put("event-source-2", "topic2");
    metadata.put("event-source-3", "topic3");
    metadata.put("event-source-4", "topic4");
    metadata.put("event-source-5", "topic5");
    return metadata;
}

private GlueSchemaRegistryConfiguration getConfigs() {
    GlueSchemaRegistryConfiguration configs = new
    GlueSchemaRegistryConfiguration(regionName);
    configs.setSchemaName(schemaName);
    configs.setAutoRegistration(true);
    configs.setMetadata(getMetadata());
    return configs;
}
```

```
private Object getTestRecord() throws IOException {
    GenericRecord genericRecord;
    Schema.Parser parser = new Schema.Parser();
    Schema avroSchema = parser.parse(new File(AVRO_USER_SCHEMA_FILE));

    genericRecord = new GenericData.Record(avroSchema);
    genericRecord.put("name", "testName");
    genericRecord.put("favorite_number", 99);
    genericRecord.put("favorite_color", "red");

    return genericRecord;
}
```

Use Case: Amazon Kinesis Data Analytics for Apache Flink

Apache Flink is a popular open source framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Amazon Kinesis Data Analytics for Apache Flink is a fully managed AWS service that enables you to build and manage Apache Flink applications to process streaming data.

Open source Apache Flink provides a number of sources and sinks. For example, predefined data sources include reading from files, directories, and sockets, and ingesting data from collections and iterators. Apache Flink DataStream Connectors provide code for Apache Flink to interface with various third-party systems, such as Apache Kafka or Kinesis as sources and/or sinks.

For more information, see [Amazon Kinesis Data Analytics Developer Guide](#).

Note

AWS Glue Data Catalog tables can only be created from AVRO schemas registered with the Schema Registry. We plan to support creating tables from JSON schemas in a future release.

Apache Flink Kafka Connector

Apache Flink provides an Apache Kafka data stream connector for reading data from and writing data to Kafka topics with exactly-once guarantees. Flink's Kafka consumer, `FlinkKafkaConsumer`, provides access to read from one or more Kafka topics. Apache Flink's Kafka Producer, `FlinkKafkaProducer`, allows writing a stream of records to one or more Kafka topics. For more information, see [Apache Kafka Connector](#).

Apache Flink Kinesis Streams Connector

The Kinesis data stream connector provides access to Amazon Kinesis Data Streams. The `FlinkKinesisConsumer` is an exactly-once parallel streaming data source that subscribes to multiple Kinesis streams within the same AWS service region, and can transparently handle re-sharding of streams while the job is running. Each subtask of the consumer is responsible for fetching data records from multiple Kinesis shards. The number of shards fetched by each subtask will change as shards are closed and created by Kinesis. The `FlinkKinesisProducer` uses Kinesis Producer Library (KPL) to put data from an Apache Flink stream into a Kinesis stream. For more information, see [Amazon Kinesis Streams Connector](#).

For more information, see the [AWS Glue Schema Github repository](#).

Integrating with Apache Flink

The SerDes library provided with Schema Registry integrates with Apache Flink. To work with Apache Flink, you are required to implement `SerializationSchema` and

`DeserializationSchema` interfaces called `GlueSchemaRegistryAvroSerializationSchema` and `GlueSchemaRegistryAvroDeserializationSchema`, which you can plug into Apache Flink connectors.

Adding an AWS Glue Schema Registry Dependency into the Apache Flink Application

To set up the integration dependencies to AWS Glue Schema Registry in the Apache Flink application:

1. Add the dependency to the `pom.xml` file.

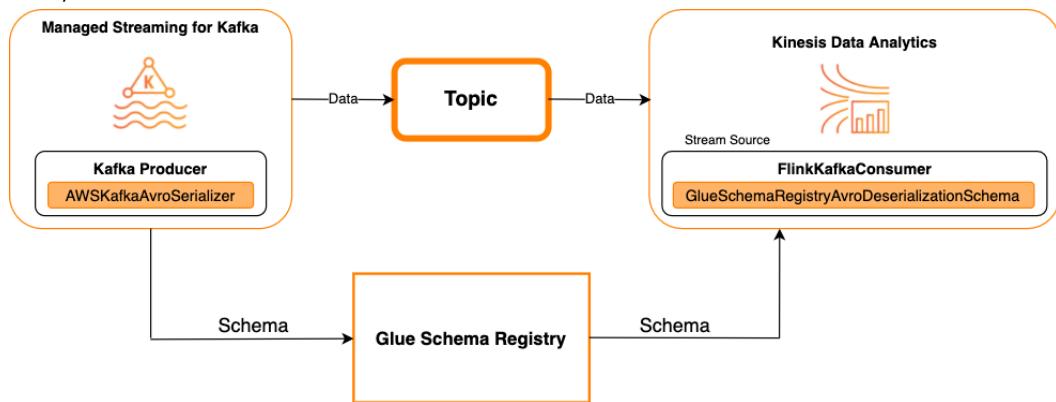
```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-flink-serde</artifactId>
    <version>1.0.0</version>
</dependency>
```

Integrating Kafka or Amazon MSK with Apache Flink

You can use Kinesis Data Analytics for Apache Flink, with Kafka as a source or Kafka as a sink.

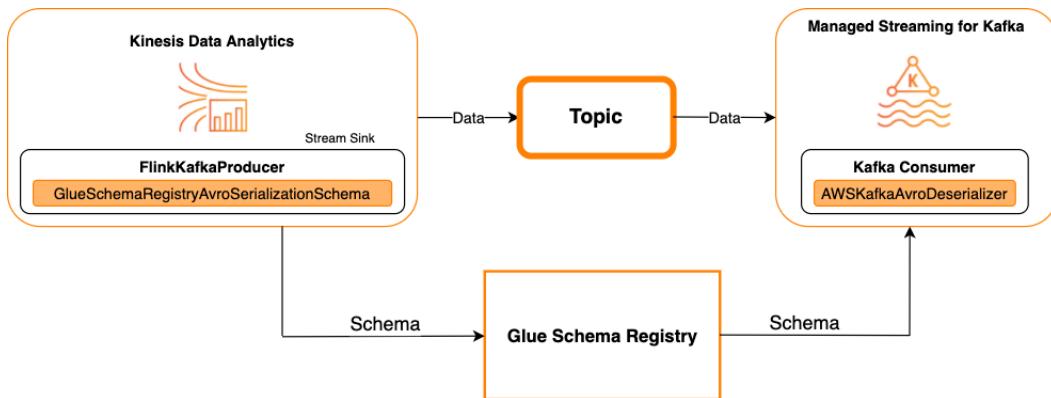
Kafka as a source

The following diagram shows integrating Kinesis Data Streams with Kinesis Data Analytics for Apache Flink, with Kafka as a source.



Kafka as a sink

The following diagram shows integrating Kinesis Data Streams with Kinesis Data Analytics for Apache Flink, with Kafka as a sink.



To integrate Kafka (or Amazon MSK) with Kinesis Data Analytics for Apache Flink, with Kafka as a source or Kafka as a sink, make the code changes below. Add the bolded code blocks to your respective code in the analogous sections.

If Kafka is the source, then use the deserializer code (block 2). If Kafka is the sink, use the serializer code (block 3).

```

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String topic = "topic";
Properties properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
properties.setProperty("group.id", "test");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
  AvroRecordType.GENERIC_RECORD.getName());

FlinkKafkaConsumer<GenericRecord> consumer = new FlinkKafkaConsumer<>(
  topic,
  // block 2
  GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
  properties);

FlinkKafkaProducer<GenericRecord> producer = new FlinkKafkaProducer<>(
  topic,
  // block 3
  GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
  properties);

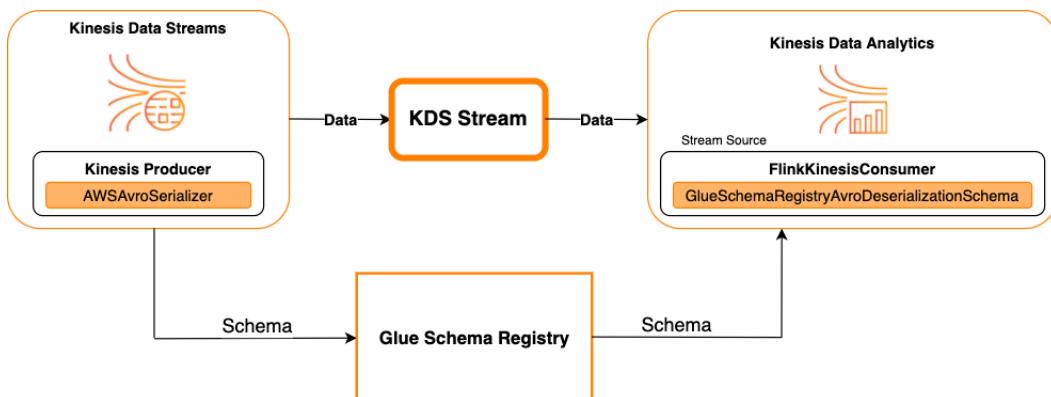
DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
  
```

Integrating Kinesis Data Streams with Apache Flink

You can use Kinesis Data Analytics for Apache Flink with Kinesis Data Streams as a source or a sink.

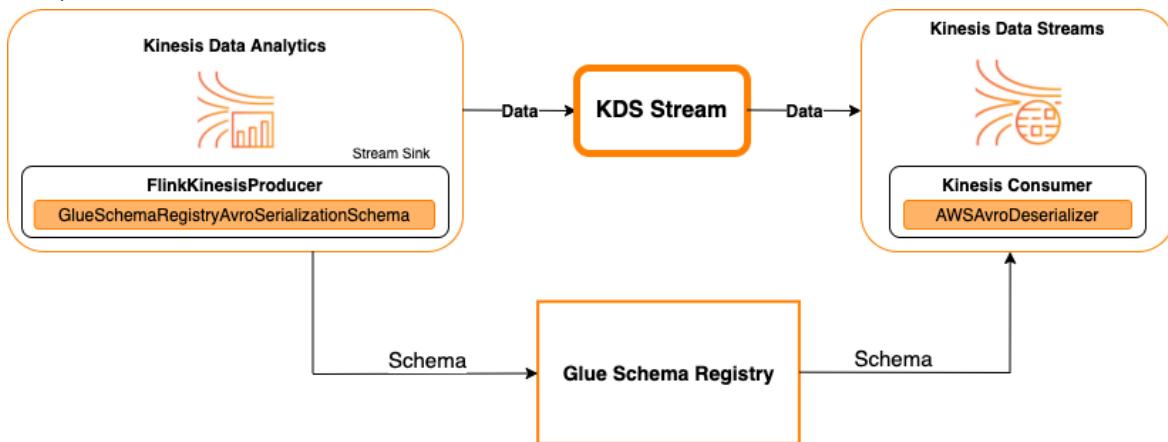
Kinesis Data Streams as a source

The following diagram shows integrating Kinesis Data Streams with Kinesis Data Analytics for Apache Flink, with Kinesis Data Streams as a source.



Kinesis Data Streams as a sink

The following diagram shows integrating Kinesis Data Streams with Kinesis Data Analytics for Apache Flink, with Kinesis Data Streams as a sink.



To integrate Kinesis Data Streams with Kinesis Data Analytics for Apache Flink, with Kinesis Data Streams as a source or Kinesis Data Streams as a sink, make the code changes below. Add the bolded code blocks to your respective code in the analogous sections.

If Kinesis Data Streams is the source, use the deserializer code (block 2). If Kinesis Data Streams is the sink, use the serializer code (block 3).

```

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String streamName = "stream";
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "aws-region");
consumerConfig.put(AWSConfigConstants.AWS_ACCESS_KEY_ID, "aws_access_key_id");
consumerConfig.put(AWSConfigConstants.AWS_SECRET_ACCESS_KEY, "aws_secret_access_key");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
AvroRecordType.GENERIC_RECORD.getName());

FlinkKinesisConsumer<GenericRecord> consumer = new FlinkKinesisConsumer<>(
    streamName,
    
```

```
// block 2
GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
properties);

FlinkKinesisProducer<GenericRecord> producer = new FlinkKinesisProducer<>(
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);
producer.setDefaultStream(streamName);
producer.setDefaultPartition("0");

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
```

Use Case: Integration with AWS Lambda

To use an AWS Lambda function as an Apache Kafka/Amazon MSK consumer and deserialize Avro-encoded messages using AWS Glue Schema Registry, visit the [MSK Labs page](#).

Use Case: AWS Glue Data Catalog

AWS Glue tables support schemas that you can specify manually or by reference to the AWS Glue Schema Registry. The Schema Registry integrates with the Data Catalog to allow you to optionally use schemas stored in the Schema Registry when creating or updating AWS Glue tables or partitions in the Data Catalog. To identify a schema definition in the Schema Registry, at a minimum, you need to know the ARN of the schema it is part of. A schema version of a schema, which contains a schema definition, can be referenced by its UUID or version number. There is always one schema version, the "latest" version, that can be looked up without knowing its version number or UUID.

Note

For schemas stored in AWS Glue Schema Registry, currently AWS Glue tables can only be created from AVRO schemas. However, support for the creation of AWS Glue tables from JSON schemas may be included in a future release.

When calling the `CreateTable` or `UpdateTable` operations, you will pass a `TableInput` structure that contains a `StorageDescriptor`, which may have a `SchemaReference` to an existing schema in the Schema Registry. Similarly, when you call the `GetTable` or `GetPartition` APIs, the response may contain the schema and the `SchemaReference`. When a table or partition was created using a schema references, the Data Catalog will try to fetch the schema for this schema reference. In case it is unable to find the schema in the Schema Registry, it returns an empty schema in the `GetTable` response; otherwise the response will have both the schema and schema reference.

You can also perform the actions from the AWS Glue console.

To perform these operations and create, update, or view the schema information, you must give the caller IAM user permissions for the `GetSchemaVersion` API.

Adding a Table or Updating the Schema for a Table

Adding a new table from an existing schema binds the table to a specific schema version. Once new schema versions get registered, you can update this table definition from the View table page in the AWS Glue console or using the [UpdateTable Action \(Python: update_table\) \(p. 739\)](#) API.

Adding a Table from an Existing Schema

You can create an AWS Glue table from a schema version in the registry using the AWS Glue console or `CreateTable` API.

AWS Glue API

When calling the `CreateTable` API, you will pass a `TableInput` that contains a `StorageDescriptor` which has a `SchemaReference` to an existing schema in the Schema Registry.

AWS Glue console

To create a table from the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Tables**.
3. In the **Add Tables** menu, choose **Add table from existing schema**.
4. Configure the table properties and data store per the AWS Glue Developer Guide.
5. In the **Choose a Glue schema** page, select the **Registry** where the schema resides.
6. Choose the **Schema name** and select the **Version** of the schema to apply.
7. Review the schema preview, and choose **Next**.
8. Review and create the table.

The schema and version applied to the table appears in the **Glue schema** column in the list of tables. You can view the table to see more details.

Updating the Schema for a Table

When a new schema version becomes available, you may want to update a table's schema using the [UpdateTable Action \(Python: update_table\) \(p. 739\)](#) API or the AWS Glue console.

Important

When updating the schema for an existing table that has an AWS Glue schema specified manually, the new schema referenced in the Schema Registry may be incompatible. This can result in your jobs failing.

AWS Glue API

When calling the `UpdateTable` API, you will pass a `TableInput` that contains a `StorageDescriptor` which has a `SchemaReference` to an existing schema in the Schema Registry.

AWS Glue console

To update the schema for a table from the AWS Glue console:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. In the navigation pane, under **Data catalog**, choose **Tables**.
3. View the table from the list of tables.
4. Click **Update schema** in the box that informs you about a new version.
5. Review the differences between the current and new schema.
6. Choose **Show all schema differences** to see more details.
7. Choose **Save table** to accept the new version.

Use case: AWS Glue Streaming

AWS Glue streaming consumes data from streaming sources and perform ETL operations before writing to an output sink. Input streaming source can be specified using a Data Table or directly by specifying the source configuration.

AWS Glue streaming supports a Data Catalog table for the streaming source created with the schema present in the AWS Glue Schema Registry. You can create a schema in the AWS Glue Schema Registry and create an AWS Glue table with a streaming source using this schema. This AWS Glue table can be used as an input to an AWS Glue streaming job for deserializing data in the input stream.

One point to note here is when the schema in the AWS Glue Schema Registry changes, you need to restart the AWS Glue streaming job needs to reflect the changes in the schema.

Use Case: Apache Kafka Streams

The Apache Kafka Streams API is a client library for processing and analyzing data stored in Apache Kafka. This section describes the integration of Apache Kafka Streams with AWS Glue Schema Registry, which allows you to manage and enforce schemas on your data streaming applications. For more information on Apache Kafka Streams, see [Apache Kafka Streams](#).

Integrating with the SerDes Libraries

There is a `GlueSchemaRegistryKafkaStreamsSerde` class that you can configure a Streams application with.

Kafka Streams Application Example Code

To use the AWS Glue Schema Registry within an Apache Kafka Streams application:

1. Configure the Kafka Streams application.

```
final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "avro-streams");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
    Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
    AWSKafkaAvroSerDe.class.getName());
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    props.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
    props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
    props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
    AvroRecordType.GENERIC_RECORD.getName());
    props.put(AWSSchemaRegistryConstants.DATA_FORMAT, DataFormat.AVRO.name());
```

2. Create a stream from the topic avro-input.

```
StreamsBuilder builder = new StreamsBuilder();
final KStream<String, GenericRecord> source = builder.stream("avro-input");
```

3. Process the data records (the example filters out those records whose value of favorite_color is pink or where the value of amount is 15).

```
final KStream<String, GenericRecord> result = source
    .filter((key, value) -> !"pink".equals(String.valueOf(value.get("favorite_color"))));
    .filter((key, value) -> !"15.0".equals(String.valueOf(value.get("amount"))));
```

4. Write the results back to the topic avro-output.

```
result.to("avro-output");
```

5. Start the Apache Kafka Streams application.

```
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

Implementation Results

These results show the filtering process of records that were filtered out in step 3 as a favorite_color of "pink" or value of "15.0".

Records before filtering:

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
 {"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
 {"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}
 {"name": "Jay", "favorite_number": 0, "favorite_color": "pink"}

 {"id": "commute_1", "amount": 3.5}
 {"id": "grocery_1", "amount": 25.5}
 {"id": "entertainment_1", "amount": 19.2}
 {"id": "entertainment_2", "amount": 105}
 {"id": "commute_1", "amount": 15}
```

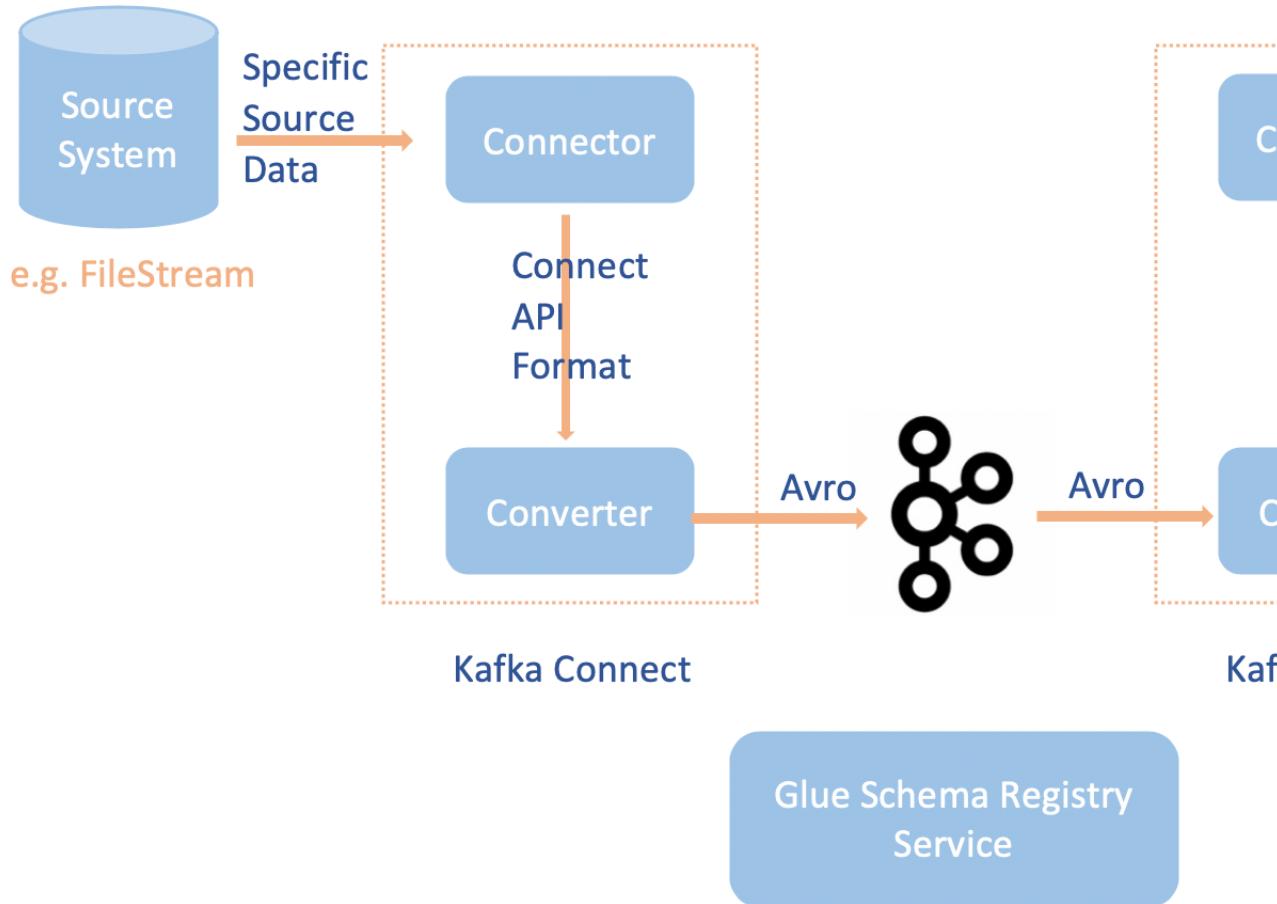
Records after filtering:

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
 {"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
 {"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
 {"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}

 {"id": "commute_1", "amount": 3.5}
 {"id": "grocery_1", "amount": 25.5}
 {"id": "entertainment_1", "amount": 19.2}
 {"id": "entertainment_2", "amount": 105}
```

Use Case: Apache Kafka Connect

The integration of Apache Kafka Connect with the AWS Glue Schema Registry enables you to get schema information from connectors. The Apache Kafka converters specify the format of data within Apache Kafka and how to translate it into Apache Kafka Connect data. Every Apache Kafka Connect user will need to configure these converters based on the format they want their data in when loaded from or stored into Apache Kafka. In this way, you can define your own converters to translate Apache Kafka Connect data into the type used in the AWS Glue Schema Registry (for example: Avro) and utilize our serializer to register its schema and do serialization. Then converters are also able to use our deserializer to deserialize data received from Apache Kafka and convert it back into Apache Kafka Connect data. An example workflow diagram is given below.



1. Install the `aws-glue-schema-registry` project by cloning the [Github repository for the AWS Glue Schema Registry](#).

```
git clone git@github.com:awslabs/aws-glue-schema-registry.git
cd aws-glue-schema-registry
mvn clean install
mvn dependency:copy-dependencies
```

2. If you plan on using Apache Kafka Connect in *Standalone* mode, update `connect-standalone.properties` using the instructions below for this step. If you plan on using Apache Kafka Connect in *Distributed* mode, update `connect-avro-distributed.properties` using the same instructions.

- a. Add these properties also to the Apache Kafka connect properties file:

```
key.converter.region=aws-region
value.converter.region=aws-region
key.converter.schemaAutoRegistrationEnabled=true
value.converter.schemaAutoRegistrationEnabled=true
key.converter.avroRecordType=GENERIC_RECORD
value.converter.avroRecordType=GENERIC_RECORD
```

- b. Add the command below to the **Launch mode** section under **kafka-run-class.sh**:

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

3. Add the command below to the **Launch mode** section under **kafka-run-class.sh**

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

It should look like this:

```
# Launch mode
if [ "x$DAEMON_MODE" = "xtrue" ]; then
    nohup "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
    $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@" > "$CONSOLE_OUTPUT_FILE" 2>&1 < /dev/null
    &
else
    exec "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
    $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@"
fi
```

4. If using bash, run the below commands to set-up your CLASSPATH in your bash_profile. For any other shell, update the environment accordingly.

```
echo 'export GSR_LIB_BASE_DIR=<>' >>~/.bash_profile
echo 'export GSR_LIB_VERSION=1.0.0' >>~/.bash_profile
echo 'export KAFKA_HOME=<your Apache Kafka installation directory>' >>~/.bash_profile
echo 'export CLASSPATH=$CLASSPATH:$GSR_LIB_BASE_DIR/avro-kafkaconnect-converter/target/
schema-registry-kafkaconnect-converter-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/common/
target/schema-registry-common-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/avro-serializer-
deserializer/target/schema-registry-serde-$GSR_LIB_VERSION.jar' >>~/.bash_profile
source ~/.bash_profile
```

5. (Optional) If you want to test with a simple file source, then clone the file source connector.

```
git clone https://github.com/mmolimar/kafka-connect-fs.git
cd kafka-connect-fs/
```

- a. Under the source connector configuration, edit the data format to Avro, file reader to `AvroFileReader` and update an example Avro object from the file path you are reading from. For example:

```
vim config/kafka-connect-fs.properties
```

```
fs.uris=<path to a sample avro object>
policy.regexp=^.*\.avro$
file_reader.class=com.github.mmolimar.kafka.connect.fs.file.reader.AvroFileReader
```

- b. Install the source connector.

```
mvn clean package
echo "export CLASSPATH=\$CLASSPATH:\$(find target/ -type f -name '*.jar'| grep '\-
package' | tr '\n' ':')"" >>~/.bash_profile
source ~/.bash_profile
```

- c. Update the sink properties under <your Apache Kafka installation directory>/config/connect-file-sink.properties update the topic name and out file name.

```
file=<output file full path>
topics=<my topic>
```

6. Start the Source Connector (in this example it is a file source connector).

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-standalone.properties
config/kafka-connect-fs.properties
```

7. Run the Sink Connector (in this example it is a file sink connector).

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-standalone.properties
$KAFKA_HOME/config/connect-file-sink.properties
```

For an example Kafka Connect usage, look at the run-local-tests.sh script under integration-tests folder in the [Github repository for the AWS Glue Schema Registry](#).

Migration from a Third-Party Schema Registry to AWS Glue Schema Registry

The migration from a third-party schema registry to the AWS Glue Schema Registry has a dependency on the existing, current third-party schema registry. If there are records in an Apache Kafka topic which were sent using a third-party schema registry, consumers need the third-party schema registry to deserialize those records. The `AWSKafkaAvroDeserializer` provides the ability to specify a secondary deserializer class which points to the third-party deserializer and is used to deserialize those records.

There are two criteria for retirement of a third-party schema. First, retirement can occur only after records in Apache Kafka topics using the 3rd party schema registry are either no longer required by and for any consumers. Second, retirement can occur by aging out of the Apache Kafka topics, depending on the retention period specified for those topics. Note that if you have topics which have infinite retention, you can still migrate to the AWS Glue Schema Registry but you will not be able to retire the third-party schema registry. As a workaround, you can use an application or Mirror Maker 2 to read from the current topic and produce to a new topic with the AWS Glue Schema Registry.

To migrate from a third-party schema registry to the AWS Glue Schema Registry:

1. Create a registry in the AWS Glue Schema Registry, or use the default registry.
2. Stop the consumer. Modify it to include AWS Glue Schema Registry as the primary deserializer, and the third-party schema registry as the secondary.
 - Set the consumer properties. In this example, the `secondary_deserializer` is set to a different deserializer. The behavior is as follows: the consumer retrieves records from Amazon MSK and first tries to use the `AWSKafkaAvroDeserializer`. If it is unable to read the magic byte that contains the Avro Schema ID for the AWS Glue Schema Registry schema, the `AWSKafkaAvroDeserializer` then tries to use the deserializer class provided in the `secondary_deserializer`. The properties specific to the secondary deserializer also need to be provided in the consumer properties, such as the `schema_registry_url_config` and `specific_avro_reader_config`, as shown below.

```
consumerProps.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
consumerProps.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
AWSKafkaAvroDeserializer.class.getName());
consumerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION,
KafkaClickstreamConsumer.gsrRegion);
```

```
consumerProps.setProperty(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,  
    KafkaAvroDeserializer.class.getName());  
consumerProps.setProperty(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "URL  
for third-party schema registry");  
consumerProps.setProperty(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG,  
    "true");
```

3. Restart the consumer.
4. Stop the producer and point the producer to the AWS Glue Schema Registry.
 - a. Set the producer properties. In this example, the producer will use the default-registry and auto register schema versions.

```
producerProps.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,  
    StringSerializer.class.getName());  
producerProps.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,  
    AWSKafkaAvroSerializer.class.getName());  
producerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");  
producerProps.setProperty(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,  
    AvroRecordType.SPECIFIC_RECORD.getName());  
producerProps.setProperty(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING,  
    "true");
```

5. (Optional) Manually move existing schemas and schema versions from the current third-party schema registry to the AWS Glue Schema Registry, either to the default-registry in AWS Glue Schema Registry or to a specific non-default registry in AWS Glue Schema Registry. This can be done by exporting schemas from the third-party schema registries in JSON format and creating new schemas in AWS Glue Schema Registry using the AWS Management Console or the AWS CLI.

This step may be important if you need to enable compatibility checks with previous schema versions for newly created schema versions using the AWS CLI and the AWS Management Console, or when producers send messages with a new schema with auto-registration of schema versions turned on.

6. Start the producer.

Sample AWS CloudFormation Template for Schema Registry

The following is a sample template for creating Schema Registry resources in AWS CloudFormation. To create this stack in your account, copy the above template into a file `SampleTemplate.yaml`, and run the following command:

```
aws cloudformation create-stack --stack-name ABCSchemaRegistryStack --template-body "'cat  
SampleTemplate.yaml'"
```

This example uses `AWS::Glue::Registry` to create a registry, `AWS::Glue::Schema` to create a schema, `AWS::Glue::SchemaVersion` to create a schema version, and `AWS::Glue::SchemaVersionMetadata` to populate schema version metadata.

```
Description: "A sample CloudFormation template for creating Schema Registry resources."  
Resources:  
  ABCRegistry:  
    Type: "AWS::Glue::Registry"  
    Properties:  
      Name: "ABCSchemaRegistry"  
      Description: "ABC Corp. Schema Registry"  
      Tags:  
        - Key: "Project"
```

```
        Value: "Foo"
ABCSchema:
  Type: "AWS::Glue::Schema"
  Properties:
    Registry:
      Arn: !Ref ABCRegistry
    Name: "TestSchema"
    Compatibility: "NONE"
    DataFormat: "AVRO"
    SchemaDefinition: >
      {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"name","type":"string"}, {"name":"favorite_number","type":"int"}]}
    Tags:
      - Key: "Project"
        Value: "Foo"
SecondSchemaVersion:
  Type: "AWS::Glue::SchemaVersion"
  Properties:
    Schema:
      SchemaArn: !Ref ABCSchema
    SchemaDefinition: >
      {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"status","type":"string", "default":"ON"}, {"name":"name","type":"string"}, {"name":"favorite_number","type":"int"}]}
FirstSchemaVersionMetadata:
  Type: "AWS::Glue::SchemaVersionMetadata"
  Properties:
    SchemaVersionId: !GetAtt ABCSchema.InitialSchemaVersionId
    Key: "Application"
    Value: "Kinesis"
SecondSchemaVersionMetadata:
  Type: "AWS::Glue::SchemaVersionMetadata"
  Properties:
    SchemaVersionId: !Ref SecondSchemaVersion
    Key: "Application"
    Value: "Kinesis"
```

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 \(p. 472\)](#)
- [Program AWS Glue ETL Scripts in Python \(p. 533\)](#)
- [Programming AWS Glue ETL Scripts in Scala \(p. 625\)](#)

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 \(p. 472\)](#)
- [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#)
- [Examples: Setting Connection Types and Options \(p. 501\)](#)
- [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 504\)](#)
- [Managing Partitions for ETL Output in AWS Glue \(p. 509\)](#)
- [Reading Input Files in Larger Groups \(p. 511\)](#)
- [Reading from JDBC Tables in Parallel \(p. 512\)](#)
- [Moving Data to and from Amazon Redshift \(p. 513\)](#)
- [AWS Glue Data Catalog Support for Spark SQL Jobs \(p. 514\)](#)
- [Excluding Amazon S3 Storage Classes \(p. 517\)](#)
- [Developing and testing AWS Glue job scripts \(p. 518\)](#)
- [Cross-Account Cross-Region Access to DynamoDB Tables \(p. 531\)](#)

Special Parameters Used by AWS Glue

AWS Glue recognizes 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 value 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 applies only if your `--job-language` is set to `scala`.
- `--scriptLocation` — The Amazon Simple Storage Service (Amazon S3) location where your ETL script is located (in the form `s3://path/to/my/script.py`). This parameter 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.
- **--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 (,).
- **--user-jars-first** — When setting this value to `true`, it prioritizes the customer's extra JAR files in the classpath. This option is only available in AWS Glue version 2.0 or later.
- **--use-postgres-driver** — When setting this value to `true`, it prioritizes the Postgres JDBC driver in the class path to avoid a conflict with the Amazon Redshift JDBC driver. This option is only available in AWS Glue version 2.0.
- **--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.
- **--disable-proxy** — Disable the service proxy to allow AWS calls to Amazon S3 for downloading scripts and libraries originating from your script through your VPC. For more information, see [Configuring AWS calls to go through your VPC](#).
- **--job-bookmark-option** — Controls the behavior of a job bookmark. The following option values can be set.

--job-bookmark-option Value	Description
<code>job-bookmark-enable</code>	Keep track of previously processed data. When a job runs, process new data since the last checkpoint.
<code>job-bookmark-disable</code>	Always process the entire dataset. You are responsible for managing the output from previous job runs.
<code>job-bookmark-pause</code>	<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 the last bookmark. You are responsible for managing the output from previous job runs. The two suboptions are as follows:</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'
```

Note

AWS Glue creates a temporary bucket for jobs if a bucket doesn't already exist in a region. This bucket might permit public access. You can either modify the bucket in Amazon S3 to set the public access block, or delete the bucket later after all jobs in that region have completed.

- `--enable-auto-scaling` — When set this value to `true`, turns on the feature to use auto scaling and per worker billing. This feature is only available for preview in the us-east-2 Region.
- `--enable-s3-parquet-optimized-committer` — Enables the EMRFS S3-optimized committer for writing Parquet data into Amazon S3. You can supply the parameter/value pair via the AWS Glue console when creating or updating an AWS Glue job. Setting the value to `true` enables the committer. By default the flag is turned off.

For more information, see [Using the EMRFS S3-optimized Committer](#).

- `--enable-rename-algorithm-v2` — Sets the EMRFS rename algorithm version to version 2. When a Spark job uses dynamic partition overwrite mode, there is a possibility that a duplicate partition is created. For instance, you can end up with a duplicate partition such as `s3://bucket/table/location/p1=1/p1=1`. Here, P1 is the partition that is being overwritten. Rename algorithm version 2 fixes this issue.

This option is only available on AWS Glue version 1.0.

- `--enable-glue-datacatalog` — Enables you to use the AWS Glue Data Catalog as an Apache Spark Hive metastore. To enable this feature, only specify the key; no value is needed.
- `--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-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 Amazon 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 applies only to driver logs and executor logs. It does not affect the AWS Glue progress bar.
- `--enable-spark-ui` — When set to `true`, turns on the feature to use the Spark UI to monitor and debug AWS Glue ETL jobs.
- `--spark-event-logs-path` — Specifies an Amazon S3 path. When using the Spark UI monitoring feature, AWS Glue flushes the Spark event logs to this Amazon S3 path every 30 seconds to a bucket that can be used as a temporary directory for storing Spark UI events.

For example, the following is the syntax for running a job using `--arguments` to set a special parameter.

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://my_glue/libraries/test_lib.py"'
```

AWS Glue uses the following arguments internally and you should never use them:

- `--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

In AWS Glue, various PySpark and Scala methods and transforms specify the connection type using a `connectionType` parameter. They specify connection options using a `connectionOptions` or `options` parameter.

The `connectionType` parameter can take the values shown in the following table. The associated `connectionOptions` (or `options`) parameter values for each type are documented in the following sections. Except where otherwise noted, the parameters apply when the connection is used as a source or sink.

For sample code that demonstrates setting and using connection options, see [the section called "Examples: Setting Connection Types and Options" \(p. 501\)](#).

connectionType	Connects To
custom.* (p. 496)	Spark, Athena, or JDBC data stores (see Custom and AWS Marketplace connectionType values (p. 496))
documentdb (p. 476)	Amazon DocumentDB (with MongoDB compatibility) database
dynamodb (p. 477)	Amazon DynamoDB database
kafka (p. 486)	Kafka or Amazon Managed Streaming for Apache Kafka
kinesis (p. 488)	Amazon Kinesis Data Streams
marketplace.* (p. 496)	Spark, Athena, or JDBC data stores (see Custom and AWS Marketplace connectionType values (p. 496))
mongodb (p. 490)	MongoDB database
mysql (p. 492)	MySQL database (see JDBC connectionType Values (p. 492))
oracle (p. 492)	Oracle database (see JDBC connectionType Values (p. 492))
orc (p. 491)	Files stored in Amazon Simple Storage Service (Amazon S3) in the Apache Hive Optimized Row Columnar (ORC) file format
parquet (p. 491)	Files stored in Amazon S3 in the Apache Parquet file format
postgresql (p. 492)	PostgreSQL database (see JDBC connectionType Values (p. 492))
redshift (p. 492)	Amazon Redshift database (see JDBC connectionType Values (p. 492))
s3 (p. 491)	Amazon S3

connectionType	Connects To
sqlserver (p. 492)	Microsoft SQL Server database (see JDBC connectionType Values (p. 492))

"connectionType": "documentdb"

Designates a connection to Amazon DocumentDB (with MongoDB compatibility).

Connection options differ for a source connection and a sink connection.

"connectionType": "documentdb" as Source

Use the following connection options with "connectionType": "documentdb" as a source:

- "uri": (Required) The Amazon DocumentDB host to read from, formatted as `mongodb://<host>:<port>`.
- "database": (Required) The Amazon DocumentDB database to read from.
- "collection": (Required) The Amazon DocumentDB collection to read from.
- "username": (Required) The Amazon DocumentDB user name.
- "password": (Required) The Amazon DocumentDB password.
- "ssl": (Required if using SSL) If your connection uses SSL, then you must include this option with the value "true".
- "ssl.domain_match": (Required if using SSL) If your connection uses SSL, then you must include this option with the value "false".
- "batchSize": (Optional): The number of documents to return per batch, used within the cursor of internal batches.
- "partitioner": (Optional): The class name of the partitioner for reading input data from Amazon DocumentDB. The connector provides the following partitioners:
 - `MongoDefaultPartitioner` (default)
 - `MongoSamplePartitioner`
 - `MongoShardedPartitioner`
 - `MongoSplitVectorPartitioner`
 - `MongoPaginateByCountPartitioner`
 - `MongoPaginateBySizePartitioner`
- "partitionerOptions" (Optional): Options for the designated partitioner. The following options are supported for each partitioner:
 - `MongoSamplePartitioner:partitionKey, partitionSizeMB, samplesPerPartition`
 - `MongoShardedPartitioner:shardkey`
 - `MongoSplitVectorPartitioner:partitionKey, partitionSizeMB`
 - `MongoPaginateByCountPartitioner:partitionKey, numberOfPartitions`
 - `MongoPaginateBySizePartitioner:partitionKey, partitionSizeMB`

For more information about these options, see [Partitioner Configuration](#) in the MongoDB documentation. For sample code, see [the section called "Examples: Setting Connection Types and Options" \(p. 501\)](#).

"connectionType": "documentdb" as Sink

Use the following connection options with "connectionType": "documentdb" as a sink:

- "uri": (Required) The Amazon DocumentDB host to write to, formatted as `mongodb://<host>:<port>`.
- "database": (Required) The Amazon DocumentDB database to write to.
- "collection": (Required) The Amazon DocumentDB collection to write to.
- "username": (Required) The Amazon DocumentDB user name.
- "password": (Required) The Amazon DocumentDB password.
- "extendedBsonTypes": (Optional) If `true`, allows extended BSON types when writing data to Amazon DocumentDB. The default is `true`.
- "replaceDocument": (Optional) If `true`, replaces the whole document when saving datasets that contain an `_id` field. If `false`, only fields in the document that match the fields in the dataset are updated. The default is `true`.
- "maxBatchSize": (Optional): The maximum batch size for bulk operations when saving data. The default is 512.

For sample code, see [the section called “Examples: Setting Connection Types and Options” \(p. 501\)](#).

"connectionType": "dynamodb"

Designates a connection to Amazon DynamoDB.

Connection options differ for a source connection and a sink connection.

"connectionType": "dynamodb" with the ETL connector as Source

Use the following connection options with "connectionType": "dynamodb" as a source, when using the AWS Glue DynamoDB ETL connector:

- "dynamodb.input.tableName": (Required) The DynamoDB table to read from.
- "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.
 - 0.5 represents the default read rate, meaning that AWS Glue will attempt to consume half of the read capacity of the table. If you increase the value above 0.5, AWS Glue increases the request rate; decreasing the value below 0.5 decreases the read request rate. (The actual read rate will vary, depending on factors such as whether there is a uniform key distribution in the DynamoDB table.)
 - When the DynamoDB table is in on-demand mode, AWS Glue handles the read capacity of the table as 40000. For exporting a large table, we recommend switching your DynamoDB table to on-demand mode.
- "dynamodb.splits": (Optional) 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.
 - 1 represents there is no parallelism. We highly recommend that you specify a larger value for better performance by using the below formula.
 - We recommend you to calculate `numSlots` using the following formula, and use it as `dynamodb.splits`. If you need more performance, we recommend you to scale out your job by increasing the number of DPUs.
 - `numExecutors` =
 - `(DPU - 1) * 2 - 1` if `WorkerType` is Standard
 - `(NumberOfWorkers - 1)` if `WorkerType` is G.1X or G.2X
 - `numSlotsPerExecutor` =
 - 4 if `WorkerType` is Standard
 - 8 if `WorkerType` is G.1X
 - 16 if `WorkerType` is G.2X

- numSlots = numSlotsPerExecutor * numExecutors
- "dynamodb.sts.roleArn": (Optional) The IAM role ARN to be assumed for cross-account access. This parameter is available in AWS Glue 1.0 or later.
- "dynamodb.sts.roleSessionName": (Optional) STS session name. The default is set to "glue-dynamodb-read-sts-session". This parameter is available in AWS Glue 1.0 or later.

The following code examples show how to read from (via the ETL connector) and write to DynamoDB tables. They demonstrate reading from one table and writing to another table.

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={"dynamodb.input.tableName": "test_source",
        "dynamodb.throughput.read.percent": "1.0",
        "dynamodb.splits": "100"
    }
)
print(dyf.getNumPartitions())

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={"dynamodb.output.tableName": "test_sink",
        "dynamodb.throughput.write.percent": "1.0"
    }
)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
```

```
options = JsonOptions(Map(
    "dynamodb.input.tableName" -> "test_source",
    "dynamodb.throughput.read.percent" -> "1.0",
    "dynamodb.splits" -> "100"
))
).getDynamicFrame()

print(dynamicFrame.getNumPartitions())

val dynamoDbSink: DynamoDbDataSink = glueContext.getSinkWithFormat(
    connectionType = "dynamodb",
    options = JsonOptions(Map(
        "dynamodb.output.tableName" -> "test_sink",
        "dynamodb.throughput.write.percent" -> "1.0"
    ))
).asInstanceOf[DynamoDbDataSink]

dynamoDbSink.writeDynamicFrame(dynamicFrame)

Job.commit()
}

}
```

Note

AWS Glue supports reading data from another AWS account's DynamoDB table. For more information, see [the section called "Cross-Account Cross-Region Access to DynamoDB Tables" \(p. 531\)](#).

Note

The DynamoDB ETL reader does not support filters or pushdown predicates.

"connectionType": "dynamodb" with the AWS Glue DynamoDB export connector as Source

This feature is in preview release for AWS Glue and is subject to change.

In addition to the AWS Glue DynamoDB ETL connector, AWS Glue offers a DynamoDB export connector, that invokes a `DynamoDB ExportTableToPointInTime` request and stores it in an Amazon S3 location you supply, in the format of [DynamoDB JSON](#). AWS Glue then creates a `DynamicFrame` object by reading the data from the Amazon S3 export location.

The export connector performs better than the ETL connector when the DynamoDB table size is larger than 80 GB. In addition, given that the export request is conducted outside from the Spark processes in an AWS Glue job, you can enable [auto scaling of AWS Glue jobs](#) to save DPU usage during the export request. With the export connector, you also do not need to configure the number of splits for Spark executor parallelism or DynamoDB throughput read percentage.

Use the following connection options with "connectionType": "dynamodb" as a source, when using the AWS Glue DynamoDB export connector, which is available only for AWS Glue version 2.0 onwards:

- "dynamodb.export": (Required) A string value:
 - If set to `ddb` enables the AWS Glue DynamoDB export connector where a new `ExportTableToPointInTimeRequest` will be invoked during the AWS Glue job. A new export will be generated with the location passed from `dynamodb.s3.bucket` and `dynamodb.s3.prefix`.
 - If set to `s3` enables the AWS Glue DynamoDB export connector but skips the creation of a new DynamoDB export and instead uses the `dynamodb.s3.bucket` and `dynamodb.s3.prefix` as the Amazon S3 location of a past export of that table.

- "dynamodb.tableArn": (Required) The DynamoDB table to read from.
- "dynamodb.unnestDDBJson": (Optional) Takes a boolean value. If set to true, performs an unnest transformation of the DynamoDB JSON structure that is present in exports. The default value is set to false.
- "dynamodb.s3.bucket": (Optional) Indicates the Amazon S3 bucket location in which the DynamoDB ExportTableToPointInTime process is to be conducted. The file format for the export is DynamoDB JSON.
- "dynamodb.s3.prefix": (Optional) Indicates the Amazon S3 prefix location inside the Amazon S3 bucket in which the DynamoDB ExportTableToPointInTime loads are to be stored. If neither dynamodb.s3.prefix nor dynamodb.s3.bucket are specified, these values will default to the Temporary Directory location specified in the AWS Glue job configuration. For more information, see [Special Parameters Used by AWS Glue](#).
- "dynamodb.s3.bucketOwner": Indicates the bucket owner needed for cross-account Amazon S3 access.
- "dynamodb.sts.roleArn": (Optional) The IAM role ARN to be assumed for cross-account access and/or cross-region access for the DynamoDB table. Note: The same IAM role ARN will be used to access the Amazon S3 location specified for the ExportTableToPointInTime request.
- "dynamodb.sts.roleSessionName": (Optional) STS session name. The default is set to "glue-dynamodb-read-sts-session".

Note

DynamoDB has specific requirements to invoke the `ExportTableToPointInTime` requests. For more information, see [Requesting a table export in DynamoDB](#). For example, Point-in-Time-Restore (PITR) needs to be enabled on the table to use this connector. The DynamoDB connector also supports KMS encryption for DynamoDB exports to Amazon S3. Supplying your security configuration in the AWS Glue job configuration enables KMS encryption for a DynamoDB export. The KMS key must be in the same Region as the Amazon S3 bucket.

Note that additional charges for DynamoDB export and Amazon S3 storage costs apply. Exported data in Amazon S3 persists after a job run finishes so you can reuse it without additional DynamoDB exports. A requirement for using this connector is that Point-In-time Recovery (PITR) is enabled for the table.

The DynamoDB ETL connector or export connector do not support filters or pushdown predicates to be applied at the DynamoDB source.

The following code examples show how to read from (via the export connector) and print the number of partitions.

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
    }
)
```

```

        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
print(dyf.getNumPartitions())

job.commit()

```

Scala

```

import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getSourceWithFormat(
            connectionType = "dynamodb",
            options = JsonOptions(Map(
                "dynamodb.export" -> "ddb",
                "dynamodb.tableArn" -> "<test_source>",
                "dynamodb.s3.bucket" -> "<bucket name>",
                "dynamodb.s3.prefix" -> "<bucket prefix>",
                "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
            )))
        .getDynamicFrame()

        print(dynamicFrame.getNumPartitions())

        Job.commit()
    }
}

```

These examples show how to do the read from (via the export connector) and print the number of partitions from an AWS Glue Data Catalog table that has a dynamodb classification:

Python

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_catalog(
    database="<catalog_database>",
    table_name="<catalog_table_name>",

```

```

        additional_options={
            "dynamodb.export": "ddb",
            "dynamodb.s3.bucket": "<s3_bucket>",
            "dynamodb.s3.prefix": "<s3_bucket_prefix>"
        }
    )
print(dynamicFrame.getNumPartitions())

job.commit()

```

Scala

```

import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getCatalogSource(
            database = "<catalog_database>",
            tableName = "<catalog_table_name>",
            additionalOptions = JsonOptions(Map(
                "dynamodb.export" -> "ddb",
                "dynamodb.s3.bucket" -> "<s3_bucket>",
                "dynamodb.s3.prefix" -> "<s3_bucket_prefix>"
            )))
        .getDynamicFrame()
        print(dynamicFrame.getNumPartitions())
    }
}

```

Traversing the DynamoDB JSON structure

The DynamoDB exports with the AWS Glue DynamoDB export connector can result in JSON files of specific nested structures. For more information, see [Data objects](#). AWS Glue supplies a DynamicFrame transformation, which can unnest such structures into an easier-to-use form for downstream applications.

The transform can be invoked in one of two ways, the first being a boolean flag which is passed with the AWS Glue DynamoDB export connector. The second, by calling the transform function itself.

The following code examples show how to use the AWS Glue DynamoDB export connector, invoke an unnest, and print the number of partitions:

Python

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])

```

```

glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.unnestDDBJson": True,
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
print(dyf.getNumPartitions())

job.commit()

```

Scala

```

import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getSourceWithFormat(
            connectionType = "dynamodb",
            options = JsonOptions(Map(
                "dynamodb.export" -> "ddb",
                "dynamodb.tableArn" -> "<test_source>",
                "dynamodb.unnestDDBJson" -> true
                "dynamodb.s3.bucket" -> "<bucket name>",
                "dynamodb.s3.prefix" -> "<bucket prefix>",
                "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
            )))
        .getDynamicFrame()

        print(dynamicFrame.getNumPartitions())

        Job.commit()
    }
}

```

The other invocation of the transform is through a separate `DynamicFrame` function call. For more information, see [DynamicFrame Class](#) for Python and [AWS Glue Scala DynamicFrame Class](#) for Scala.

"connectionType": "dynamodb" with the ETL connector as Sink

Use the following connection options with "connectionType": "dynamodb" as a sink:

- "dynamodb.output.tableName": (Required) The DynamoDB table to write to.
- "dynamodb.throughput.write.percent": (Optional) The percentage of write capacity units (WCUs) to use. The default is set to "0.5". Acceptable values are from "0.1" to "1.5", inclusive.
 - 0.5 represents the default write rate, meaning that AWS Glue will attempt to consume half of the write capacity of the table. If you increase the value above 0.5, AWS Glue increases the request rate; decreasing the value below 0.5 decreases the write request rate. (The actual write rate will vary, depending on factors such as whether there is a uniform key distribution in the DynamoDB table).
- When the DynamoDB table is in on-demand mode, AWS Glue handles the write capacity of the table as 40000. For importing a large table, we recommend switching your DynamoDB table to on-demand mode.
- "dynamodb.output.numParallelTasks": (Optional) Defines how many parallel tasks write into DynamoDB at the same time. Used to calculate permissive WCU per Spark task. If you do not want to control these details, you do not need to specify this parameter.
 - `permissiveWcuPerTask = TableWCUs * dynamodb.throughput.write.percent / dynamodb.output.numParallelTasks`
 - If you do not specify this parameter, the permissive WCU per Spark task will be automatically calculated by the following formula:
 - `numPartitions = dynamicframe.getNumPartitions()`
 - `numExecutors =

 - $(DPU - 1) * 2 - 1$ if WorkerType is Standard
 - $(\text{NumberOfWorkers} - 1)$ if WorkerType is G.1X or G.2X`
 - `numSlotsPerExecutor =

 - 4 if WorkerType is Standard
 - 8 if WorkerType is G.1X
 - 16 if WorkerType is G.2X`
 - `numSlots = numSlotsPerExecutor * numExecutors`
 - `numParallelTasks = min(numPartitions, numSlots)`
 - Example 1. DPU=10, WorkerType=Standard. Input DynamicFrame has 100 RDD partitions.
 - `numPartitions = 100`
 - `numExecutors = $(10 - 1) * 2 - 1 = 17$`
 - `numSlots = $4 * 17 = 68$`
 - `numParallelTasks = min(100, 68) = 68`
 - Example 2. DPU=10, WorkerType=Standard. Input DynamicFrame has 20 RDD partitions.
 - `numPartitions = 20`
 - `numExecutors = $(10 - 1) * 2 - 1 = 17$`
 - `numSlots = $4 * 17 = 68$`
 - `numParallelTasks = min(20, 68) = 20`
- "dynamodb.output.retry": (Optional) Defines how many retries we perform when there is a ProvisionedThroughputExceededException from DynamoDB. The default is set to "10".
- "dynamodb.sts.roleArn": (Optional) The IAM role ARN to be assumed for cross-account access.
- "dynamodb.sts.roleSessionName": (Optional) STS session name. The default is set to "glue-dynamodb-write-sts-session".

Note

The DynamoDB writer is supported in AWS Glue version 1.0 or later.

Note

AWS Glue supports writing data into another AWS account's DynamoDB table. For more information, see [the section called "Cross-Account Cross-Region Access to DynamoDB Tables" \(p. 531\)](#).

The following code examples show how to read from and write to DynamoDB tables. They demonstrate reading from one table and writing to another table.

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.input.tableName": "test_source",
        "dynamodb.throughput.read.percent": "1.0",
        "dynamodb.splits": "100"
    }
)

print(dyf.getNumPartitions())

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={
        "dynamodb.output.tableName": "test_sink",
        "dynamodb.throughput.write.percent": "1.0"
    }
)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getSourceWithFormat(
            connectionType = "dynamodb",
            options = JsonOptions(Map(
                "dynamodb.input.tableName" -> "test_source",
                "dynamodb.throughput.read.percent" -> "1.0",
                "dynamodb.splits" -> "100"
            )))
        .getDynamicFrame()
```

```

        print(dynamicFrame.getNumPartitions())

        val dynamoDbSink: DynamoDbDataSink = glueContext.getSinkWithFormat(
            connectionType = "dynamodb",
            options = JsonOptions(Map(
                "dynamodb.output.tableName" -> "test_sink",
                "dynamodb.throughput.write.percent" -> "1.0"
            )))
        .asInstanceOf[DynamoDbDataSink]

        dynamoDbSink.writeDynamicFrame(dynamicFrame)

        Job.commit()
    }
}

```

"connectionType": "kafka"

Designates a connection to a Kafka cluster or an Amazon Managed Streaming for Apache Kafka cluster.

You can use the following methods under the `GlueContext` object to consume records from a Kafka streaming source:

- `getCatalogSource`
- `getSource`
- `getSourceWithFormat`
- `createDataFrameFromOptions`

If you use `getCatalogSource`, then the job has the Data Catalog database and table name information, and can use that to obtain some basic parameters for reading from the Apache Kafka stream. If you use `getSource`, `getSourceWithFormat`, or `createDataFrameFromOptions` you must explicitly specify these parameters:

You can specify these options using `connectionOptions` with `getSource` or `createDataFrameFromOptions`, `options` with `getSourceWithFormat`, or `additionalOptions` with `getCatalogSource`.

Use the following connection options with "connectionType": "kafka":

- `bootstrap.servers` (Required) A list of bootstrap server URLs, for example, as `b-1.vpc-test-2.04q880.c6.kafka.us-east-1.amazonaws.com:9094`. This option must be specified in the API call or defined in the table metadata in the Data Catalog.
- `security.protocol` (Required) The protocol used to communicate with brokers. The possible values are "SSL" or "PLAINTEXT".
- `topicName` (Required) A comma-separated list of topics to subscribe to. You must specify one and only one of "topicName", "assign" or "subscribePattern".
- "assign": (Required) A JSON string specifying the specific `TopicPartitions` to consume. You must specify one and only one of "topicName", "assign" or "subscribePattern".

Example: `'{"topicA":[0,1],"topicB":[2,4]}'`

- "subscribePattern": (Required) A Java regex string that identifies the topic list to subscribe to. You must specify one and only one of "topicName", "assign" or "subscribePattern".

Example: `'topic.*'`

- `classification` (Optional)

- **delimiter** (Optional)
- "startingOffsets": (Optional) The starting position in the Kafka topic to read data from. The possible values are "earliest" or "latest". The default value is "latest".
- "endingOffsets": (Optional) The end point when a batch query is ended. Possible values are either "latest" or a JSON string that specifies an ending offset for each TopicPartition.

For the JSON string, the format is {"topicA":{"0":23,"1":-1}, "topicB":{"0":-1}}. The value -1 as an offset represents "latest".

- "pollTimeoutMs": (Optional) The timeout in milliseconds to poll data from Kafka in Spark job executors. The default value is 512.
- "numRetries": (Optional) The number of times to retry before failing to fetch Kafka offsets. The default value is 3.
- "retryIntervalMs": (Optional) The time in milliseconds to wait before retrying to fetch Kafka offsets. The default value is 10.
- "maxOffsetsPerTrigger": (Optional) The rate limit on the maximum number of offsets that are processed per trigger interval. The specified total number of offsets is proportionally split across topicPartitions of different volumes. The default value is null, which means that the consumer reads all offsets until the known latest offset.
- "minPartitions": (Optional) The desired minimum number of partitions to read from Kafka. The default value is null, which means that the number of spark partitions is equal to the number of Kafka partitions.
- "includeHeaders": (Optional) Whether to include the Kafka headers. When the option is set to "true", the data output will contain an additional column named "glue_streaming_kafka_headers" with type Array[Struct(key: String, value: String)]. The default value is "false". This option is available in AWS Glue version 3.0 only.
- "schema": (Required when inferSchema set to false) The schema to use to process the payload. If classification is avro the provided schema must be in the Avro schema format. If the classification is not avro the provided schema must be in the DDL schema format.

The following are schema examples.

Example in DDL schema format

```
'column1' INT, 'column2' STRING , 'column3' FLOAT
```

Example in Avro schema format

```
{
  "type": "array",
  "items": [
    {
      "type": "record",
      "name": "test",
      "fields": [
        {
          "name": "_id",
          "type": "string"
        },
        {
          "name": "index",
          "type": [
            "int",
            "string",
            "float"
          ]
        }
      ]
    }
  ]
}
```

```
        ]  
    }  
}
```

- "inferSchema": (Optional) The default value is 'false'. If set to 'true', the schema will be detected at runtime from the payload within `foreachbatch`.
- "avroSchema": (Deprecated) Parameter used to specify a schema of Avro data when Avro format is used. This parameter is now deprecated. Use the `schema` parameter.

"connectionType": "kinesis"

Designates connection options for Amazon Kinesis Data Streams.

You can read from an Amazon Kinesis data stream using information stored in a Data Catalog table, or by providing information to directly access the data stream. If you directly access the data stream, use these options to provide the information about how to access the data stream.

If you use `getCatalogSource` or `create_data_frame_from_catalog` to consume records from a Kinesis streaming source, the job has the Data Catalog database and table name information, and can use that to obtain some basic parameters for reading from the Kinesis streaming source. If you use `getSource`, `getSourceWithFormat`, `createDataFrameFromOptions` or `create_data_frame_from_options`, you must specify these basic parameters using the connection options described here.

You can specify the connection options for Kinesis using the following arguments for the specified methods in the `GlueContext` class.

- Scala
 - `connectionOptions`: Use with `getSource`, `createDataFrameFromOptions`
 - `additionalOptions`: Use with `getCatalogSource`
 - `options`: Use with `getSourceWithFormat`
- Python
 - `connection_options`: Use with `create_data_frame_from_options`
 - `additional_options`: Use with `create_data_frame_from_catalog`
 - `options`: Use with `getSource`

Use the following connection options for Kinesis streaming data sources:

- `streamARN` (Required) The ARN of the Kinesis data stream.
- `classification` (Optional)
- `delimiter` (Optional)
- "startingPosition": (Optional) The starting position in the Kinesis data stream to read data from. The possible values are "latest", "trim_horizon", or "earliest". The default value is "latest".
- "awsSTSSRoleARN": (Optional) The Amazon Resource Name (ARN) of the role to assume using AWS Security Token Service (AWS STS). This role must have permissions for describe or read record operations for the Kinesis data stream. You must use this parameter when accessing a data stream in a different account. Used in conjunction with "awsSTSSessionName".
- "awsSTSSessionName": (Optional) An identifier for the session assuming the role using AWS STS. You must use this parameter when accessing a data stream in a different account. Used in conjunction with "awsSTSSRoleARN".
- "maxFetchTimeInMs": (Optional) The maximum time spent in the job executor to fetch a record from the Kinesis data stream per shard, specified in milliseconds (ms). The default value is 1000.

- "maxFetchRecordsPerShard": (Optional) The maximum number of records to fetch per shard in the Kinesis data stream. The default value is 100000.
- "maxRecordPerRead": (Optional) The maximum number of records to fetch from the Kinesis data stream in each getRecords operation. The default value is 10000.
- "addIdleTimeBetweenReads": (Optional) Adds a time delay between two consecutive getRecords operations. The default value is "False". This option is only configurable for Glue version 2.0 and above.
- "idleTimeBetweenReadsInMs": (Optional) The minimum time delay between two consecutive getRecords operations, specified in ms. The default value is 1000. This option is only configurable for Glue version 2.0 and above.
- "describeShardInterval": (Optional) The minimum time interval between two ListShards API calls for your script to consider resharding. For more information, see [Strategies for Resharding in Amazon Kinesis Data Streams Developer Guide](#). The default value is 1s.
- "numRetries": (Optional) The maximum number of retries for Kinesis Data Streams API requests. The default value is 3.
- "retryIntervalMs": (Optional) The cool-off time period (specified in ms) before retrying the Kinesis Data Streams API call. The default value is 1000.
- "maxRetryIntervalMs": (Optional) The maximum cool-off time period (specified in ms) between two retries of a Kinesis Data Streams API call. The default value is 10000.
- "avoidEmptyBatches": (Optional) Avoids creating an empty microbatch job by checking for unread data in the Kinesis data stream before the batch is started. The default value is "False".
- "schema": (Required when inferSchema set to false) The schema to use to process the payload. If classification is avro the provided schema must be in the Avro schema format. If the classification is not avro the provided schema must be in the DDL schema format.

The following are schema examples.

Example in DDL schema format

```
'column1' INT, 'column2' STRING , 'column3' FLOAT
```

Example in Avro schema format

```
{
  "type": "array",
  "items": [
    {
      "type": "record",
      "name": "test",
      "fields": [
        {
          "name": "_id",
          "type": "string"
        },
        {
          "name": "index",
          "type": [
            "int",
            "string",
            "float"
          ]
        }
      ]
    }
  ]
}
```

- "inferSchema": (Optional) The default value is 'false'. If set to 'true', the schema will be detected at runtime from the payload within `foreachbatch`.
- "avroSchema": (Deprecated) Parameter used to specify a schema of Avro data when Avro format is used. This parameter is now deprecated. Use the `schema` parameter.

"connectionType": "mongodb"

Designates a connection to MongoDB. Connection options differ for a source connection and a sink connection.

"connectionType": "mongodb" as Source

Use the following connection options with "connectionType": "mongodb" as a source:

- "uri": (Required) The MongoDB host to read from, formatted as `mongodb://<host>:<port>`.
- "database": (Required) The MongoDB database to read from. This option can also be passed in `additional_options` when calling `glue_context.create_dynamic_frame_from_catalog` in your job script.
- "collection": (Required) The MongoDB collection to read from. This option can also be passed in `additional_options` when calling `glue_context.create_dynamic_frame_from_catalog` in your job script.
- "username": (Required) The MongoDB user name.
- "password": (Required) The MongoDB password.
- "ssl": (Optional) If `true`, initiates an SSL connection. The default is `false`.
- "ssl.domain_match": (Optional) If `true` and `ssl` is `true`, domain match check is performed. The default is `true`.
- "batchSize": (Optional): The number of documents to return per batch, used within the cursor of internal batches.
- "partitioner": (Optional): The class name of the partitioner for reading input data from MongoDB. The connector provides the following partitioners:
 - `MongoDefaultPartitioner` (default)
 - `MongoSamplePartitioner` (Requires MongoDB 3.2 or later)
 - `MongoShardedPartitioner`
 - `MongoSplitVectorPartitioner`
 - `MongoPaginateByCountPartitioner`
 - `MongoPaginateBySizePartitioner`
- "partitionerOptions" (Optional): Options for the designated partitioner. The following options are supported for each partitioner:
 - `MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition`
 - `MongoShardedPartitioner: shardkey`
 - `MongoSplitVectorPartitioner: partitionKey, partitionSizeMB`
 - `MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions`
 - `MongoPaginateBySizePartitioner: partitionKey, partitionSizeMB`

For more information about these options, see [Partitioner Configuration](#) in the MongoDB documentation. For sample code, see [the section called "Examples: Setting Connection Types and Options" \(p. 501\)](#).

"connectionType": "mongodb" as Sink

Use the following connection options with "connectionType": "mongodb" as a sink:

- "uri": (Required) The MongoDB host to write to, formatted as `mongodb://<host>:<port>`.
- "database": (Required) The MongoDB database to write to.
- "collection": (Required) The MongoDB collection to write to.
- "username": (Required) The MongoDB user name.
- "password": (Required) The MongoDB password.
- "ssl": (Optional) If `true`, initiates an SSL connection. The default is `false`.
- "ssl.domain_match": (Optional) If `true` and `ssl` is `true`, domain match check is performed. The default is `true`.
- "extendedBsonTypes": (Optional) If `true`, allows extended BSON types when writing data to MongoDB. The default is `true`.
- "replaceDocument": (Optional) If `true`, replaces the whole document when saving datasets that contain an `_id` field. If `false`, only fields in the document that match the fields in the dataset are updated. The default is `true`.
- "maxBatchSize": (Optional): The maximum batch size for bulk operations when saving data. The default is 512.

For sample code, see [the section called “Examples: Setting Connection Types and Options” \(p. 501\)](#).

"connectionType": "orc"

Designates a connection to files stored in Amazon S3 in the [Apache Hive Optimized Row Columnar \(ORC\)](#) file format.

Use the following connection options with "connectionType": "orc":

- `paths`: (Required) A list of the Amazon S3 paths to read from.
- (*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": "parquet"

Designates a connection to files stored in Amazon S3 in the [Apache Parquet](#) file format.

Use the following connection options with "connectionType": "parquet":

- `paths`: (Required) A list of the Amazon S3 paths to read from.
- (*Other option name/value pairs*): Any additional options, including formatting options, are passed directly to the SparkSQL DataSource. For more information, see [Amazon Redshift data source for Spark](#) on the GitHub website.

"connectionType": "s3"

Designates a connection to Amazon S3.

Use the following connection options with "connectionType": "s3":

- "paths": (Required) A list of the Amazon S3 paths to read from.
- "exclusions": (Optional) A string containing a JSON list of Unix-style glob patterns to exclude. For example, "[\\"**.pdf\\"]" excludes all PDF files. For more information about the glob syntax that AWS Glue supports, see [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 turned on by default when the input contains more than 50,000 files. To turn on 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 milliseconds after which the s3 listing is likely to be consistent. Files with modification timestamps falling within the last maxBand milliseconds are tracked specially when using JobBookmarks to account for Amazon S3 eventual consistency. Most users don't need to set this option. The default is 900000 milliseconds, or 15 minutes.
- "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 don't need to set this option.
- "isFailFast": (Optional) This option determines if an AWS Glue ETL job throws reader parsing exceptions. If set to true, jobs fail fast if four retries of the Spark task fail to parse the data correctly.

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 versions for Glue 0.9, 1.0, 2.0	JDBC driver versions for Glue 3.0
Microsoft SQL Server	6.x	7.x
MySQL	5.1	8.0.23
Oracle Database	11.2	21.1
PostgreSQL	42.1.x	42.2.18
MongoDB	2.0.0	4.0.0
Amazon Redshift	4.1	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 user name to use when connecting.
- "password": (Required) The password to use when connecting.
- (Optional) The following options allow 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 allows 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.
- "bulksize": (Optional) Used to configure parallel inserts for speeding up bulk loads into JDBC targets. Specify an integer value for the degree of parallelism to use when writing or inserting data. This option is helpful for improving the performance of writes into databases such as Arch User Repository (AUR).
- "sampleQuery": (Optional) The custom SQL query statement for sampling. By default the sample query is executed by single executor. If you're reading a large dataset, you may need to enable JDBC partitioning to query a table in parallel. For more information, see [Reading from JDBC Tables in Parallel](#). To use sampleQuery with JDBC partitioning, also set enablePartitioningForSampleQuery to true.
- "enablePartitioningForSampleQuery": (Optional) By default this option is false. Required if you want to use sampleQuery with a partitioned JDBC table. If set to true, sampleQuery must end with "where" or "and" for AWS Glue to append partitioning conditions. See the example below.
- "sampleSize": (Optional) Limit the number of rows returned by the sample query. Works only when enablePartitioningForSampleQuery is true. If partitioning is not enabled, you should instead directly add "limit x" in the sampleQuery to limit the size.

Example Use sampleQuery without partitioning

The following code example shows how to use sampleQuery without partitioning.

```
//A full sql query statement.
val query = "select name from $tableName where age > 0 limit 1"
val connectionOptions = JsonOptions(Map(
    "url" # url,
    "dbtable" # table,
    "user" # user,
    "password" # password,
    "basePath" # basePath,
    "sampleQuery" # query ))
val dyf = glueContext.getSource("mysql", connectionOptions)
    .getDynamicFrame()
```

Example Use sampleQuery with JDBC partitioning

The following code example shows how to use sampleQuery with JDBC partitioning.

```
//note that the query should end with "where" or "and" if use with JDBC partitioning.
val query = "select name from $tableName where age > 0 and"

//Enable JDBC partitioning by setting hashfield.
//to use sampleQuery with partitioning, set enablePartitioningForSampleQuery.
//use sampleSize to limit the size of returned data.
```

```

val connectionOptions = JsonOptions(Map(
    "url" # url,
    "dbtable" # table,
    "user" # user,
    "password" # password,
    "basePath" # basePath,
    "hashfield" -> primaryKey,
    "sampleQuery" # query,
    "enablePartitioningForSampleQuery" -> true,
    "sampleSize" -> "1" ))
val dyf = glueContext.getSource("mysql", connectionOptions)
    .getDynamicFrame()

```

For the Redshift connection type, 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.getSource("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)
}
}

```

Custom and AWS Marketplace connectionType values

These include the following:

- "connectionType": "marketplace.athena": Designates a connection to an Amazon Athena data store. The connection uses a connector from AWS Marketplace.
- "connectionType": "marketplace.spark": Designates a connection to an Apache Spark data store. The connection uses a connector from AWS Marketplace.
- "connectionType": "marketplace.jdbc": Designates a connection to a JDBC data store. The connection uses a connector from AWS Marketplace.
- "connectionType": "custom.athena": Designates a connection to an Amazon Athena data store. The connection uses a custom connector that you upload to AWS Glue Studio.
- "connectionType": "custom.spark": Designates a connection to an Apache Spark data store. The connection uses a custom connector that you upload to AWS Glue Studio.
- "connectionType": "custom.jdbc": Designates a connection to a JDBC data store. The connection uses a custom connector that you upload to AWS Glue Studio.

Connection options for type `custom.jdbc` or `marketplace.jdbc`

- `className` – String, required, driver class name.
- `connectionName` – String, required, name of the connection that is associated with the connector.
- `url` – String, required, JDBC URL with placeholders `$()` which are used to build the connection to the data source. The placeholder `$(secretKey)` is replaced with the secret of the same name in AWS Secrets Manager. Refer to the data store documentation for more information about constructing the URL.
- `secretId` or `user/password` – String, required, used to retrieve credentials for the URL.
- `dbTable` or `query` – String, required, the table or SQL query to get the data from. You can specify either `dbTable` or `query`, but not both.

- **partitionColumn** – String, optional, the name of an integer column that is used for partitioning. This option works only when it's included with `lowerBound`, `upperBound`, and `numPartitions`. This option works the same way as in the Spark SQL JDBC reader. For more information, see [JDBC To Other Databases](#) in the *Apache Spark SQL, DataFrames and Datasets Guide*.

The `lowerBound` and `upperBound` values are used to decide the partition stride, not for filtering the rows in table. All rows in the table are partitioned and returned.

Note

When using a query instead of a table name, you should validate that the query works with the specified partitioning condition. For example:

- If your query format is "SELECT col1 FROM table1", then test the query by appending a WHERE clause at the end of the query that uses the partition column.
- If your query format is "SELECT col1 FROM table1 WHERE col2=val", then test the query by extending the WHERE clause with AND and an expression that uses the partition column.
- `lowerBound` – Integer, optional, the minimum value of `partitionColumn` that is used to decide partition stride.
- `upperBound` – Integer, optional, the maximum value of `partitionColumn` that is used to decide partition stride.
- `numPartitions` – Integer, optional, the number of partitions. This value, along with `lowerBound` (inclusive) and `upperBound` (exclusive), form partition strides for generated WHERE clause expressions that are used to split the `partitionColumn`.

Important

Be careful with the number of partitions because too many partitions might cause problems on your external database systems.

- `filterPredicate` – String, optional, extra condition clause to filter data from source. For example:

```
BillingCity='Mountain View'
```

When using a *query* instead of a *table* name, you should validate that the query works with the specified `filterPredicate`. For example:

- If your query format is "SELECT col1 FROM table1", then test the query by appending a WHERE clause at the end of the query that uses the filter predicate.
- If your query format is "SELECT col1 FROM table1 WHERE col2=val", then test the query by extending the WHERE clause with AND and an expression that uses the filter predicate.
- `dataTypeMapping` – Dictionary, optional, custom data type mapping that builds a mapping from a **JDBC** data type to a **Glue** data type. For example, the option "dataTypeMapping":
`{"FLOAT": "STRING"}` maps data fields of JDBC type FLOAT into the Java String type by calling the `ResultSet.getString()` method of the driver, and uses it to build Glue Record. The `ResultSet` object is implemented by each driver, so the behavior is specific to the driver you use. Refer to the documentation for your JDBC driver to understand how the driver performs the conversions.
- The AWS Glue data type supported currently are:
 - DATE
 - STRING
 - TIMESTAMP
 - INT
 - FLOAT
 - LONG
 - BIGDECIMAL
 - BYTE
 - SHORT

- DOUBLE

The JDBC data types supported are [Java8 java.sql.types](#).

The default data type mappings (from JDBC to AWS Glue) are:

- DATE -> DATE
- VARCHAR -> STRING
- CHAR -> STRING
- LONGNVARCHAR -> STRING
- TIMESTAMP -> TIMESTAMP
- INTEGER -> INT
- FLOAT -> FLOAT
- REAL -> FLOAT
- BIT -> BOOLEAN
- BOOLEAN -> BOOLEAN
- BIGINT -> LONG
- DECIMAL -> BIGDECIMAL
- NUMERIC -> BIGDECIMAL
- TINYINT -> SHORT
- SMALLINT -> SHORT
- DOUBLE -> DOUBLE

If you use a custom data type mapping with the option `dataTypeMapping`, then you can override a default data type mapping. Only the JDBC data types listed in the `dataTypeMapping` option are affected; the default mapping is used for all other JDBC data types. You can add mappings for additional JDBC data types if needed. If a JDBC data type is not included in either the default mapping or a custom mapping, then the data type converts to the AWS Glue `STRING` data type by default.

The following Python code example shows how to read from JDBC databases with AWS Marketplace JDBC drivers. It demonstrates reading from a database and writing to an S3 location.

```

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: [connection_type = "marketplace.jdbc", connection_options =
{"dataTypeMapping":{"INTEGER":"STRING"}, "upperBound": "200", "query": "select id,
    name, department from department where id < 200", "numPartitions": "4",
    "partitionColumn": "id", "lowerBound": "0", "connectionName": "test-connection-jdbc"}, transformation_ctx = "DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
    "marketplace.jdbc", connection_options = {"dataTypeMapping": {"INTEGER": "STRING"},
```

```

"upperBound":"200","query":"select id, name, department from department where
id < 200","numPartitions":"4","partitionColumn":"id","lowerBound":"0",
"connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
## @type: ApplyMapping
## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
"name", "string"), ("id", "int", "id", "int")], transformation_ctx = "Transform0"]
## @return: Transform0
## @inputs: [frame = DataSource0]
Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
"department", "string"), ("name", "string", "name", "string"), ("id", "int", "id",
"int")],
transformation_ctx = "Transform0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path":
"s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = Transform0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
connection_type = "s3", format = "json", connection_options = {"path":
"s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()

```

Connection options for type custom.athena or marketplace.athena

- **className** – String, required, driver class name. When you're using the Athena-CloudWatch connector, this parameter value is the prefix of the class Name (for example, "com.amazonaws.athena.connectors"). The Athena-CloudWatch connector is composed of two classes: a metadata handler and a record handler. If you supply the common prefix here, then the API loads the correct classes based on that prefix.
- **tableName** – String, required, the name of the CloudWatch log stream to read. This code snippet uses the special view name all_log_streams, which means that the dynamic data frame returned will contain data from all log streams in the log group.
- **schemaName** – String, required, the name of the CloudWatch log group to read from. For example, /aws-glue/jobs/output.
- **connectionName** – String, required, name of the connection that is associated with the connector.

For additional options for this connector, see the [Amazon Athena CloudWatch Connector README](#) file on GitHub.

The following Python code example shows how to read from an Athena data store using an AWS Marketplace connector. It demonstrates reading from Athena and writing to an S3 location.

```

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: [connection_type = "marketplace.athena", connection_options =
  {"tableName":"all_log_streams","schemaName":"/aws-glue/jobs/output",
   "connectionName":"test-connection-athena"}, transformation_ctx = "DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
    "marketplace.athena", connection_options = {"tableName":"all_log_streams",
    "schemaName":"/aws-glue/jobs/output","connectionName":
    "test-connection-athena"}, transformation_ctx = "DataSource0")
## @type: ApplyMapping
## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
    "name", "string"), ("id", "int", "id", "int")], transformation_ctx = "Transform0"]
## @return: Transform0
## @inputs: [frame = DataSource0]
Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
    "department", "string"), ("name", "string", "name", "string"), ("id", "int", "id",
"int")],
    transformation_ctx = "Transform0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path":
"s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = Transform0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
    connection_type = "s3", format = "json", connection_options = {"path":
"s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()

```

Connection options for type custom.spark or marketplace.spark

- **className** – String, required, connector class name.
- **secretId** – String, optional, used to retrieve credentials for the connector connection.
- **connectionName** – String, required, name of the connection that is associated with the connector.
- Other options depend on the data store. For example, OpenSearch configuration options start with the prefix `es`, as described in the [Elasticsearch for Apache Hadoop](#) documentation. Spark connections to Snowflake use options such as `sfUser` and `sfPassword`, as described in [Using the Spark Connector](#) in the [Connecting to Snowflake](#) guide.

The following Python code example shows how to read from an OpenSearch data store using a `marketplace.spark` connection.

```

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: [connection_type = "marketplace.spark", connection_options = {"path":"test",
"es.nodes.wan.only":"true","es.nodes":"https://<AWS endpoint>"}

```

```

    "connectionName": "test-spark-es", "es.port": "443"}, transformation_ctx =
"DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
"marketplace.spark", connection_options = {"path": "test", "es.nodes.wan.only": "true", "es.nodes": "https://<AWS endpoint>", "connectionName": "test-spark-es", "es.port": "443"}, transformation_ctx = "DataSource0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path": "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = DataSource0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = DataSource0,
connection_type = "s3", format = "json", connection_options = {"path": "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()

```

Examples: Setting Connection Types and Options

The sample code in this section demonstrates how to set connection types and connection options when connecting to extract, transform, and load (ETL) sources and sinks. The code shows how to specify connection types and connection options in both Python and Scala for connections to MongoDB and Amazon DocumentDB (with MongoDB compatibility). The code is similar for connecting to other data stores that AWS Glue supports.

Note

When you create an ETL job that connects to Amazon DocumentDB, for the `Connections` job property, you must designate a connection object that specifies the virtual private cloud (VPC) in which Amazon DocumentDB is running. For the connection object, the connection type must be `JDBC`, and the `JDBC URL` must be `mongo://<DocumentDB_host>:27017`.

Topics

- [Connecting with Python \(p. 501\)](#)
- [Connecting with Scala \(p. 503\)](#)

Connecting with Python

The following Python script demonstrates using connection types and connection options for reading and writing to MongoDB and Amazon DocumentDB.

```

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

## @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)

```

```

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
mongo_uri = "mongodb://<mongo-instanced-ip-address>:27017"
mongo_ssl_uri = "mongodb://<mongo-instanced-ip-address>:27017"
documentdb_uri = "mongodb://<mongo-instanced-ip-address>:27017"
write_uri = "mongodb://<mongo-instanced-ip-address>:27017"
documentdb_write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_docdb_options = {
    "uri": documentdb_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "1234567890",
    "ssl": "true",
    "ssl.domain_match": "false",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"
}

read_mongo_options = {
    "uri": mongo_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "pwd",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"}

ssl_mongo_options = {
    "uri": mongo_ssl_uri,
    "database": "test",
    "collection": "coll",
    "ssl": "true",
    "ssl.domain_match": "false"
}

write_mongo_options = {
    "uri": write_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "pwd"
}

write_documentdb_options = {
    "uri": documentdb_write_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "pwd"
}

# Get DynamicFrame from MongoDB and DocumentDB
dynamic_frame = glueContext.create_dynamic_frame.from_options(connection_type="mongodb",
    connection_options=read_mongo_options)
dynamic_frame2 =
    glueContext.create_dynamic_frame.from_options(connection_type="documentdb",
        connection_options=read_docdb_options)

# Write DynamicFrame to MongoDB and DocumentDB
glueContext.write_dynamic_frame.from_options(dynamic_frame, connection_type="mongodb",
    connection_options=write_mongo_options)
glueContext.write_dynamic_frame.from_options(dynamic_frame2, connection_type="documentdb",
    connection_options=write_documentdb_options)

```

```

connection_options=write_documentdb_options)

job.commit()

```

Connecting with Scala

The following Scala script demonstrates using connection types and connection options for reading and writing to MongoDB and Amazon DocumentDB.

```

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 DEFAULT_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
    val WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
    val DOC_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
    val DOC_WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
    lazy val defaultJsonOption = jsonOptions(DEFAULT_URI)
    lazy val documentDBJsonOption = jsonOptions(DOC_URI)
    lazy val writeJsonOption = jsonOptions(WRITE_URI)
    lazy val writeDocumentDBJsonOption = jsonOptions(DOC_WRITE_URI)
    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)

        // Get DynamicFrame from MongoDB and DocumentDB
        val resultFrame: DynamicFrame = glueContext.getSource("mongodb",
        defaultJsonOption).getDynamicFrame()
        val resultFrame2: DynamicFrame = glueContext.getSource("documentdb",
        documentDBJsonOption).getDynamicFrame()

        // Write DynamicFrame to MongoDB and DocumentDB
        glueContext.getSink("mongodb", writeJsonOption).writeDynamicFrame(resultFrame)
        glueContext.getSink("documentdb", writeJsonOption).writeDynamicFrame(resultFrame2)

        Job.commit()
    }

    private def jsonOptions(uri: String): JsonOptions = {
        new JsonOptions(
            s"""{"uri": "${uri}",
               |"database": "test",
               |"collection": "coll",
               |"username": "username",
               |"password": "pwd",
               |"ssl": "true",
               |"ssl.domain_match": "false",
               |"partitioner": "MongoSamplePartitioner",
               |"partitionerOptions.partitionSizeMB": "10",
               |"partitionerOptions.partitionKey": "_id"}""".stripMargin)
    }
}

```

For more information, see the following:

- the section called “Connection Parameters” (p. 475)
- Amazon DocumentDB Developer Guide

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.

Note

Currently, the only formats that streaming ETL jobs support are JSON, CSV, Parquet, ORC, Avro, and Grok.

For jobs that access AWS Lake Formation governed tables, AWS Glue supports reading and writing all formats supported by Lake Formation governed tables. For the current list of supported formats for AWS Lake Formation governed tables, see [Notes and Restrictions for Governed Tables](#) in the *AWS Lake Formation Developer Guide*.

Note

For writing Apache Parquet, AWS Glue ETL only supports writing to a governed table by specifying an option for a custom Parquet writer type optimized for Dynamic Frames.

When writing to a governed table with the `parquet` format, you should add the key `useGlueParquetWriter` with a value of `true` in the table parameters.

format="avro"

This value designates the [Apache Avro](#) data format.

You can use the following `format_options` values with `format="avro"`:

- `version` — Specifies the version of Apache Avro reader/writer format to support. The default is “1.7”. You can specify `format_options={"version": "1.8"}` to enable Avro logical type reading and writing. For more information, see the [Apache Avro 1.7.7 Specification](#) and [Apache Avro 1.8.2 Specification](#).

The Apache Avro 1.8 connector supports the following logical type conversions:

For the reader: this table shows the conversion between Avro data type (logical type and Avro primitive type) and AWS Glue `DynamicFrame` data type for Avro reader 1.7 and 1.8.

Avro Data Type: Logical Type	Avro Data Type: Avro Primitive Type	GlueDynamicFrame Data Type: Avro Reader 1.7	GlueDynamicFrame Data Type: Avro Reader 1.8
Decimal	bytes	BINARY	Decimal
Decimal	fixed	BINARY	Decimal
Date	int	INT	Date
Time (millisecond)	int	INT	INT
Time (microsecond)	long	LONG	LONG

Avro Data Type: Logical Type	Avro Data Type: Avro Primitive Type	GlueDynamicFrame Data Type: Avro Reader 1.7	GlueDynamicFrame Data Type: Avro Reader 1.8
Timestamp (millisecond)	long	LONG	Timestamp
Timestamp (microsecond)	long	LONG	LONG
Duration (not a logical type)	fixed of 12	BINARY	BINARY

For the writer: this table shows the conversion between AWS Glue DynamicFrame data type and Avro data type for Avro writer 1.7 and 1.8.

AWS Glue DynamicFrame Data Type	Avro Data Type: Avro Writer 1.7	Avro Data Type: Avro Writer 1.8
Decimal	String	decimal
Date	String	date
Timestamp	String	timestamp-micros

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. This option is used only when reading CSV files. 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 turn off 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`.
- `optimizePerformance` — A Boolean value that specifies whether to use the advanced SIMD CSV reader along with Apache Arrow based columnar memory formats. Only available in AWS Glue 3.0.

The following example shows how to specify the format options within an AWS Glue ETL job script.

```
glueContext.write_dynamic_frame.from_options(
    frame = datasource1,
    connection_type = "s3",
    connection_options = {
        "path": "s3://s3path"
    },
    format = "csv",
    format_options={
        "quoteChar": -1,
        "separator": "|"
    },
    transformation_ctx = "datasink2")
```

Using vectorized SIMD CSV reader with Apache Arrow columnar format

AWS Glue version 3.0 adds the support for using Apache Arrow as the in-memory columnar format, which allows for batching records together in columnar fashion. Batching reduces CPU bottlenecks related to handling changing schema across record batches and increases retrieval efficiency of data from column buffers. AWS Glue vectorizes the CSV readers with the use of CPU SIMD instruction sets and microparallel algorithms. With these optimizations, AWS Glue version 3.0 achieves a significant performance speedup compared to using row-based AWS Glue DynamicFrame or Spark DataFrames.

To use the optimized reader with Arrow format, set "optimizePerformance" to true in the `format_options` or `table` property.

```
glueContext.create_dynamic_frame.from_options(
    frame = datasource1,
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "csv",
    format_options={
        "optimizePerformance": True,
        "separator": ","
    },
    transformation_ctx = "datasink2")
```

Limitations for the vectorized CSV reader

Note the following limitations:

- It doesn't support the `multiLine` and `escaper` format options. It uses the default `escaper` of double quote char `"`. When these options are set, AWS Glue automatically falls back to using the row-based CSV reader.
- It doesn't support creating a DynamicFrame with [ChoiceType](#).
- It doesn't support creating a DynamicFrame with [error records](#).
- It doesn't support reading CSV files with multiByte characters such as Japanese or Chinese characters.

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 turned on. 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.

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, but also provides an option to use a custom Parquet writer type that is optimized for Dynamic Frames. A pre-computed schema is not required before writing. When specified with `useGlueParquetWriter` option the writer computes and modifies the schema dynamically, as data comes in.

You can use the following `format_options` values:

- `useGlueParquetWriter` — Specifies the use of a custom Parquet writer type that is optimized for Dynamic Frames. The output format is Parquet.

See the following example, where `sink` is the object returned by `glue_context.getSink()`.

```
sink.setFormat("parquet", useGlueParquetWriter=True)
```

Limitations when specifying `useGlueParquetWriter`:

- The writer supports only schema evolution, such as adding or removing columns, but not changing column types, such as with `ResolveChoice`.
- The writer is not able to store a schema-only file.
- The option can only be passed as a format for data targets.

Additionally, any options that are accepted by the underlying SparkSQL code can be passed to this format by way of the `connection_options` map parameter. For example, a Spark configuration such as `mergeSchema` can be set for the AWS Glue Spark reader to merge the schema for all files.

- `compression` — Specifies the compression codec used when writing Parquet files. The compression codec used with the `glueparquet` format is fully compatible with `org.apache.parquet.hadoop.metadata.CompressionCodecName` *, which includes support for "uncompressed", "snappy", "gzip" and "lzo". The default value is "snappy".
- `blockSize` — Specifies the size in bytes of a row group being buffered in memory in megabytes. The default value is 134217728, or 128 MB.
- `pageSize` — Specifies the size in bytes of the smallest unit that must be read fully to access a single record. The default value is 1048576, or 1 MB.

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".
- `withSchema` — A string value that contains the expected schema. If you do not use this option, AWS Glue infers the schema from the XML data.

Example

This is an example of using the `withSchema` format option to specify the schema for XML data.

```
schema = StructType([
    Field("id", IntegerType()),
```

```

        Field("name", StringType()),
        Field("nested", StructType([
            Field("x", IntegerType()),
            Field("y", StringType()),
            Field("z", ChoiceType([IntegerType(), StringType()]))
        ]))
    ])

datasource0 = create_dynamic_frame_from_options(
    connection_type,
    connection_options={"paths": ["s3://xml_bucket/someprefix"]},
    format="xml",
    format_options={"withSchema": json.dumps(schema.jsonValue())},
    transformation_ctx = ""
)

```

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. In the AWS Glue console, choose **Tables** in the left navigation pane. Choose the table created by the crawler, and then choose **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. Because the partition information is stored in the Data Catalog, use the `from_catalog` API calls to include the partition columns in the `DynamicFrame`. For example, use `create_dynamic_frame.from_catalog` instead of `create_dynamic_frame.from_options`.

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.

Server-side filtering using catalog partition predicates

The `push_down_predicate` option is applied after listing all the partitions from the catalog and before listing files from Amazon S3 for those partitions. If you have a lot of partitions for a table, catalog partition listing can still incur additional time overhead. To address this overhead, you can use server-side partition pruning with the `catalogPartitionPredicate` option that uses [partition indexes](#) in the AWS Glue Data Catalog. This makes partition filtering much faster when you have millions of partitions in one table. You can use both `push_down_predicate` and `catalogPartitionPredicate` in `additional_options` together if your `catalogPartitionPredicate` requires predicate syntax that is not yet supported with the catalog partition indexes.

Python:

```
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=dbname,
    table_name=tablename,
    transformation_ctx="datasource0",
    push_down_predicate="day>=10 and customer_id like '10%'",
    additional_options={"catalogPartitionPredicate": "year='2021' and month='06'"}
)
```

Scala:

```
val dynamicFrame = glueContext.getCatalogSource(
    database = dbname,
    tableName = tablename,
    transformationContext = "datasource0",
    pushDownPredicate="day>=10 and customer_id like '10%'",
    additionalOptions = JsonOptions("""{
        "catalogPartitionPredicate": "year='2021' and month='06'"}
    """).getDynamicFrame()
```

Note

The `push_down_predicate` and `catalogPartitionPredicate` use different syntaxes. The former one uses Spark SQL standard syntax and the later one uses JSON parser.

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 \(p. 106\)](#).

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")
```

Note

`groupFiles` is supported for DynamicFrames created from the following data formats: csv, ion, grokLog, json, and xml. This option is not supported for avro, parquet, and orc.

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 \(p. 106\)](#). 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 \(p. 574\)](#) and [from_catalog \(p. 574\)](#).

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 existing Amazon Redshift table defined 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.

```
my_conn_options = {
    "url": "jdbc:redshift://host:port/redshift database name",
    "dbtable": "redshift table name",
    "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", my_conn_options)
```

The options are similar when writing to Amazon Redshift.

```
my_conn_options = {
    "dbtable": "redshift table name",
    "database": "redshift database name",
    "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 = my_conn_options,
    redshift_tmp_dir = args["TempDir"])
```

By default, the data in the temporary folder used by AWS Glue when reading data from the Amazon Redshift table is encrypted using SSE-S3 encryption. To use customer managed keys from AWS Key Management Service (AWS KMS) to encrypt your data, you must specify `extraunloadoptions` in `additional_options`, and provide the key ID from AWS KMS, as shown in the following example:

```
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "database-name",
    table_name = "table-name",
    redshift_tmp_dir = args["TempDir"],
    additional_options = {"extraunloadoptions": "ENCRYPTED_KMS_KEY_ID 'CMK key ID'" },
    transformation_ctx = "datasource0"
)
```

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.

This feature requires network access to the AWS Glue API endpoint. For AWS Glue jobs with connections located in private subnets, you must configure either a VPC endpoint or NAT gateway to provide the network access. For information about configuring a VPC endpoint, see [Setting Up Your Environment to Access Data Stores \(p. 32\)](#). To create a NAT gateway, see [NAT Gateways](#) in the *Amazon VPC User Guide*.

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 AWS 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 AWS Glue Data Catalog, Spark SQL needs the [Hive SerDe](#) class for the format defined in the AWS 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 \(p. 472\)](#).

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, run 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, run 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 run 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 \(p. 517\)](#)
- [Excluding Amazon S3 Storage Classes on a Data Catalog Table \(p. 518\)](#)

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 AWS Glue job scripts

When you develop and test your AWS Glue job scripts, there are multiple available options:

- AWS Glue Studio console
 - Visual editor
 - Script editor
 - AWS Glue Studio notebook
- Interactive sessions
 - Jupyter notebook
- Docker image
 - Local development
 - Remote development
- AWS Glue Studio ETL library
 - Local development

You can choose any of the above options based on your requirements.

If you prefer no code or less code experience, the AWS Glue Studio visual editor is a good choice.

If you prefer an interactive notebook experience, AWS Glue Studio notebook is a good choice. For more information, see [Using Notebooks with AWS Glue Studio and AWS Glue](#). If you want to use your own local environment, interactive sessions is a good choice. For more information, see [Using Interactive Sessions with AWS Glue](#).

If you prefer local/remote development experience, the Docker image is a good choice. This helps you to develop and test Glue job script anywhere you prefer without incurring AWS Glue cost.

If you prefer local development without Docker, installing the AWS Glue ETL library directory locally is a good choice.

Developing using AWS Glue Studio console

The AWS Glue Studio visual editor is a graphical interface that makes it easy to create, run, and monitor extract, transform, and load (ETL) jobs in AWS Glue. You can visually compose data transformation workflows and seamlessly run them on AWS Glue's Apache Spark-based serverless ETL engine. You can inspect the schema and data results in each step of the job. For more information, see the [AWS Glue Studio User Guide](#).

Developing using interactive sessions

Interactive sessions allow you to build and test applications from the environment of your choice. For more information, see [Using Interactive Sessions with AWS Glue](#).

Developing using a Docker image

For a production-ready data platform, the development process and CI/CD pipeline for AWS Glue jobs is a key topic. You can flexibly develop and test AWS Glue jobs such as by using a local laptop, and a Docker container on Amazon EC2. You can achieve that by utilizing AWS Glue Docker images hosted on DockerHub. The Docker images help you set up your development environment with additional utilities. You can use your preferred IDE, notebook, or REPL using AWS Glue ETL library. This topic describes how to develop and test AWS Glue version 3.0 jobs on a Docker container using the Docker image.

There are the following Docker images available for AWS Glue on Docker Hub.

- For AWS Glue version 3.0: `amazon/aws-glue-libs:glue_libs_3.0.0_image_01`
- For AWS Glue version 2.0: `amazon/aws-glue-libs:glue_libs_2.0.0_image_01`

These images are for x86_64.

This example describes using `amazon/aws-glue-libs:glue_libs_3.0.0_image_01` and running the container on a local machine (Mac/Windows/Linux). This container image has been tested for an AWS Glue version 3.0 Spark jobs. This image contains the following:

- Amazon Linux
- AWS Glue ETL library ([aws-glue-libs](#))
- Apache Spark 3.1.1
- Spark history server
- Jupyter Lab
- Livy
- Other library dependencies (the same set as the ones of AWS Glue job system)

Complete one of the following sections according to your requirements.

- Set up the container to use spark-submit
- Set up the container to use REPL shell (PySpark)
- Set up the container to use Pytest
- Set up the container to use Jupyter Lab
- Set up the container to use Visual Studio Code

Prerequisites

Before you start, make sure that Docker is installed and the Docker daemon is running. For installation instructions, see the Docker documentation for [Mac](#), [Windows](#), or [Linux](#). The machine running the Docker hosts the AWS Glue container. Also make sure that you have at least 7 GB of disk space for the image on the host running the Docker.

For more information about restrictions when developing AWS Glue code locally, see [Local Development Restrictions](#).

Configuring AWS credentials

To enable AWS API calls from the container, set up AWS credentials by following steps:

1. Create an AWS named profile.
2. Open cmd on Windows or terminal on Mac/Linux, and run the following command:

```
PROFILE_NAME="profile_name"
```

In the following sections, we will use this AWS named profile.

Setting up and running the container

Setting up the container to run PySpark code through the spark-submit command includes the following high-level steps:

1. Pull the image from Docker Hub.
2. Run the container.

Pulling the image from Docker Hub

If you're running Docker on Windows, choose the Docker icon (right-click) and choose **Switch to Linux containers...** before pulling the image.

Run the following command to pull the image from Docker Hub:

```
docker pull amazon/aws-glue-libs:glue_libs_3.0.0_image_01
```

Running the container

You can now run a container using this image. You can choose any of following based on your requirements.

spark-submit

You can run an AWS Glue job script by running the spark-submit command on the container.

1. Write the script and save it as `sample1.py` under the `/local_path_to_workspace` directory. Sample code is included as the appendix in this topic.

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/src
```

```
$ vim ${WORKSPACE_LOCATION}/src/${SCRIPT_FILE_NAME}
```

- Run the following command to execute the `spark-submit` command on the container to submit a new Spark application:

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_spark_submit amazon/aws-glue-libs:glue_libraries_3.0.0_image_01 spark-submit /home/glue_user/workspace/src/$SCRIPT_FILE_NAME
...22/01/26 09:08:55 INFO DAGScheduler: Job 0 finished: fromRDD at
DynamicFrame.scala:305, took 3.639886 s
root
|-- family_name: string
|-- name: string
|-- links: array
| |-- element: struct
| | |-- note: string
| | |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
| |-- element: struct
| | |-- schema: 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
```

REPL shell (PySpark)

You can run REPL (read-eval-print loops) shell for interactive development.

Run the following command to execute the PySpark command on the container to start the REPL shell:

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -e AWS_PROFILE=$PROFILE_NAME -e  
DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-  
libs:glue_libs_3.0.0_image_01 pyspark  
...  
/ __--  
 \ \/\ - \V - ^ / / / '  
 /_ / .__\_,/_/_ /_\ \  version 3.1.1-amzn-0  
/_/  
  
Using Python version 3.7.10 (default, Jun 3 2021 00:02:01)  
Spark context Web UI available at http://56e99d000c99:4040
```

```
| Spark context available as 'sc' (master = local[*], app id = local-1643011860812).  
| SparkSession available as 'spark'.  
>>>
```

Pytest

For unit testing, you can use pytest for AWS Glue Spark job scripts.

Run the following commands for preparation.

```
$ WORKSPACE_LOCATION=/local_path_to_workspace  
$ SCRIPT_FILE_NAME=sample.py  
$ UNIT_TEST_FILE_NAME=test_sample.py  
$ mkdir -p ${WORKSPACE_LOCATION}/tests  
$ vim ${WORKSPACE_LOCATION}/tests/${UNIT_TEST_FILE_NAME}
```

Run the following command to execute pytest on the test suite:

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pytest amazon/aws-glue-libs:glue_libs_3.0.0_image_01 -c "python3 -m pytest"  
starting org.apache.spark.deploy.history.HistoryServer, logging to /home/glue_user/spark/logs/spark-glue_user-org.apache.spark.deploy.history.HistoryServer-1-5168f209bd78.out  
*===== test session starts =====*  
platform linux -- Python 3.7.10, pytest-6.2.3, py-1.11.0, pluggy-0.13.1  
rootdir: /home/glue_user/workspace  
plugins: anyio-3.4.0  
*collected 1 item *  
  
tests/test_sample.py . [100%]  
  
===== warnings summary =====  
tests/test_sample.py::test_counts  
/home/glue_user/spark/python/pyspark/sql/context.py:79: DeprecationWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.  
DeprecationWarning)  
  
-- Docs: https://docs.pytest.org/en/stable/warnings.html  
===== 1 passed, *1 warning* in 21.07s =====
```

Jupyter Lab

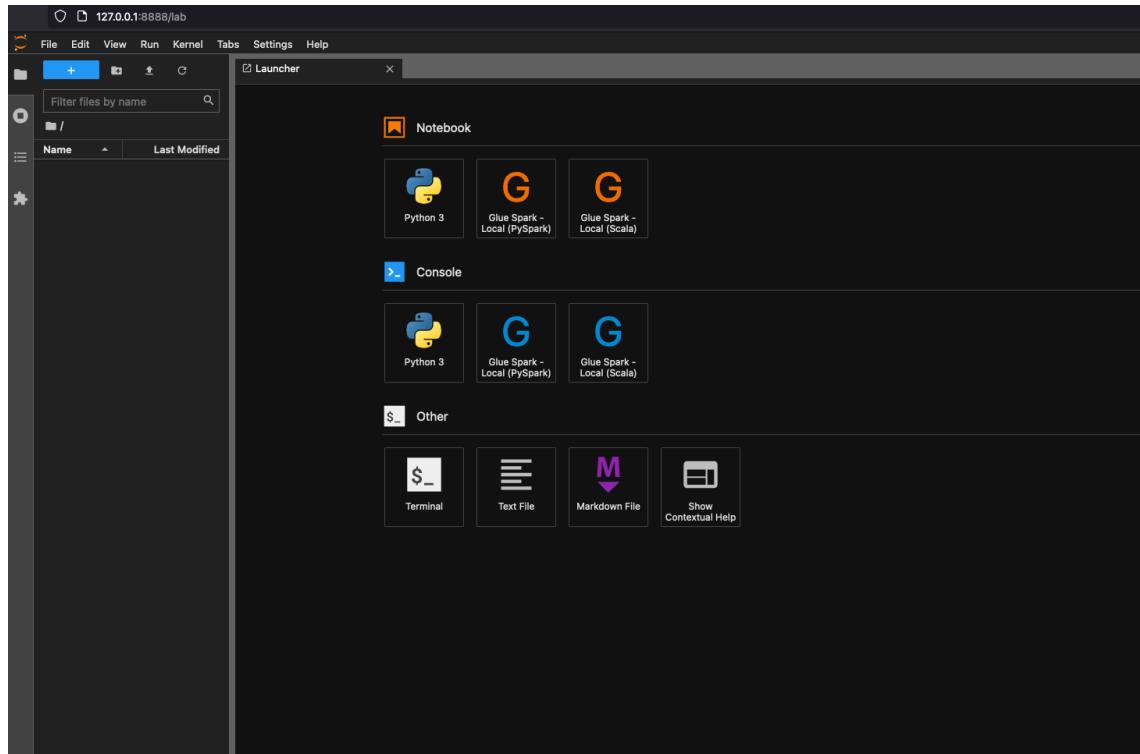
You can start Jupyter for interactive development and ad-hoc queries on notebooks.

1. Run the following command to start Jupyter Lab:

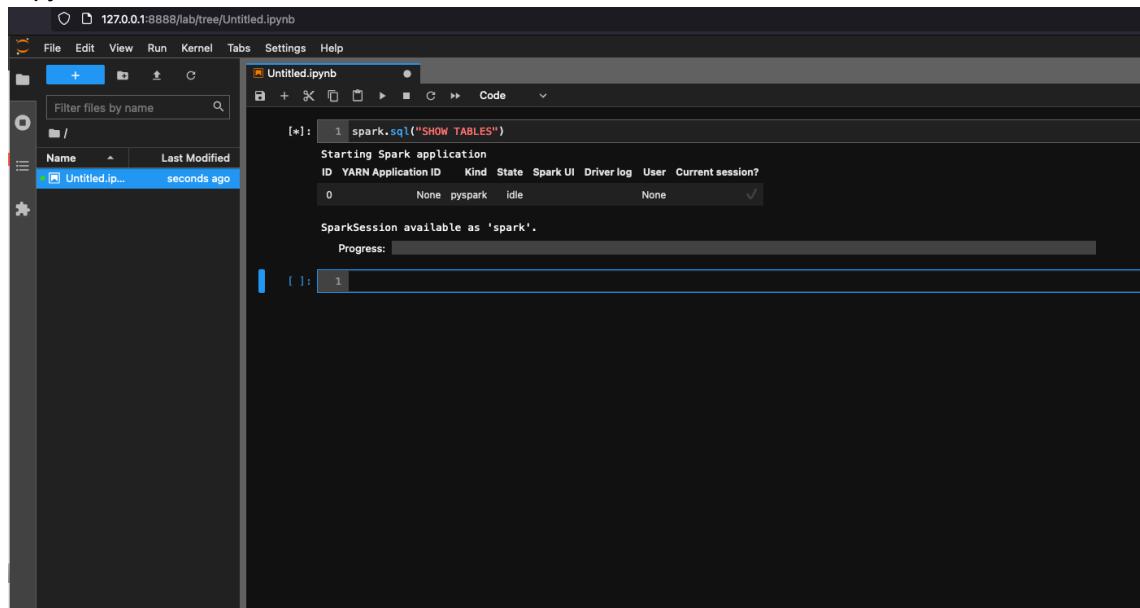
```
$ JUPYTER_WORKSPACE_LOCATION=/local_path_to_workspace/jupyter_workspace/  
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $JUPYTER_WORKSPACE_LOCATION:/home/glue_user/workspace/jupyter_workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 -p 8998:8998 -p 8888:8888 --name glue_jupyter_lab amazon/aws-glue-libs:glue_libs_3.0.0_image_01 /home/glue_user/jupyter/jupyter_start.sh  
...  
[I 2022-01-24 08:19:21.368 ServerApp] Serving notebooks from local directory: /home/glue_user/workspace/jupyter_workspace  
[I 2022-01-24 08:19:21.368 ServerApp] Jupyter Server 1.13.1 is running at:  
[I 2022-01-24 08:19:21.368 ServerApp] http://faa541f8f99f:8888/lab  
[I 2022-01-24 08:19:21.368 ServerApp] or http://127.0.0.1:8888/lab
```

```
[I 2022-01-24 08:19:21.368 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

2. Open <http://127.0.0.1:8888/lab> in your web browser in your local machine, to see the Jupyter lab UI.



3. Choose **Glue Spark Local (PySpark)** under **Notebook**. You can start developing code in the interactive Jupyter notebook UI.



Setting up the container to use Visual Studio Code

Prerequisites:

1. Install Visual Studio Code.
2. Install [Python](#).
3. Install [Visual Studio Code Remote - Containers](#)
4. Open the workspace folder in Visual Studio Code.
5. Choose **Settings**.
6. Choose **Workspace**.
7. Choose **Open Settings (JSON)**.
8. Paste the following JSON and save it.

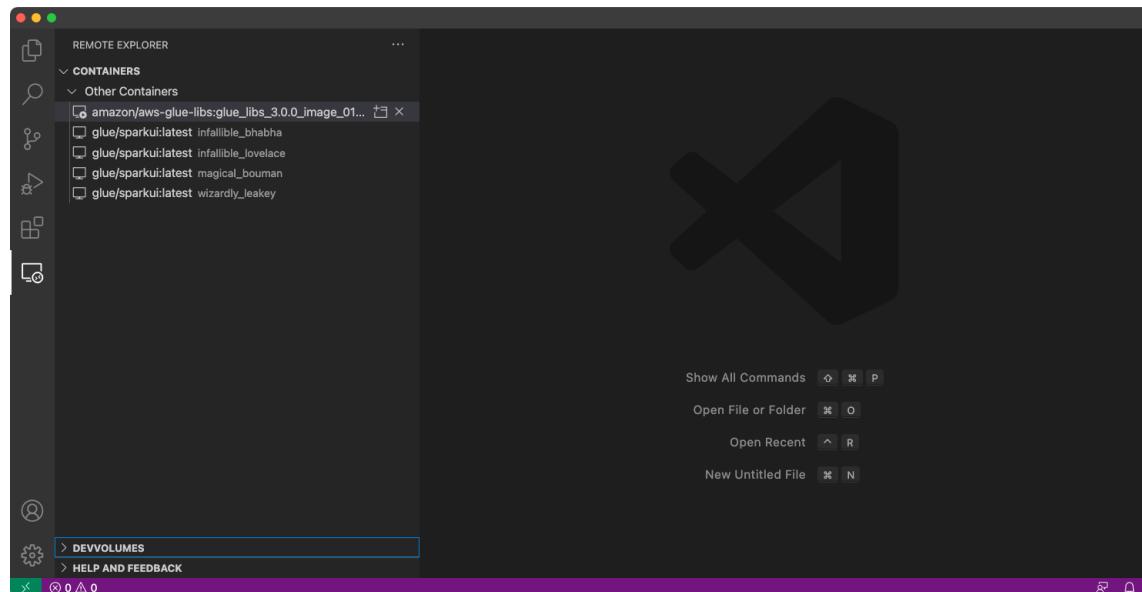
```
{  
    "python.defaultInterpreterPath": "/usr/bin/python3",  
    "python.analysis.extraPaths": [  
        "/home/glue_user/aws-glue-libs/PyGlue.zip:/home/glue_user/spark/python/lib/",  
        "/home/glue_user/spark/python/",  
    ]  
}
```

Steps:

1. Run the Docker container.

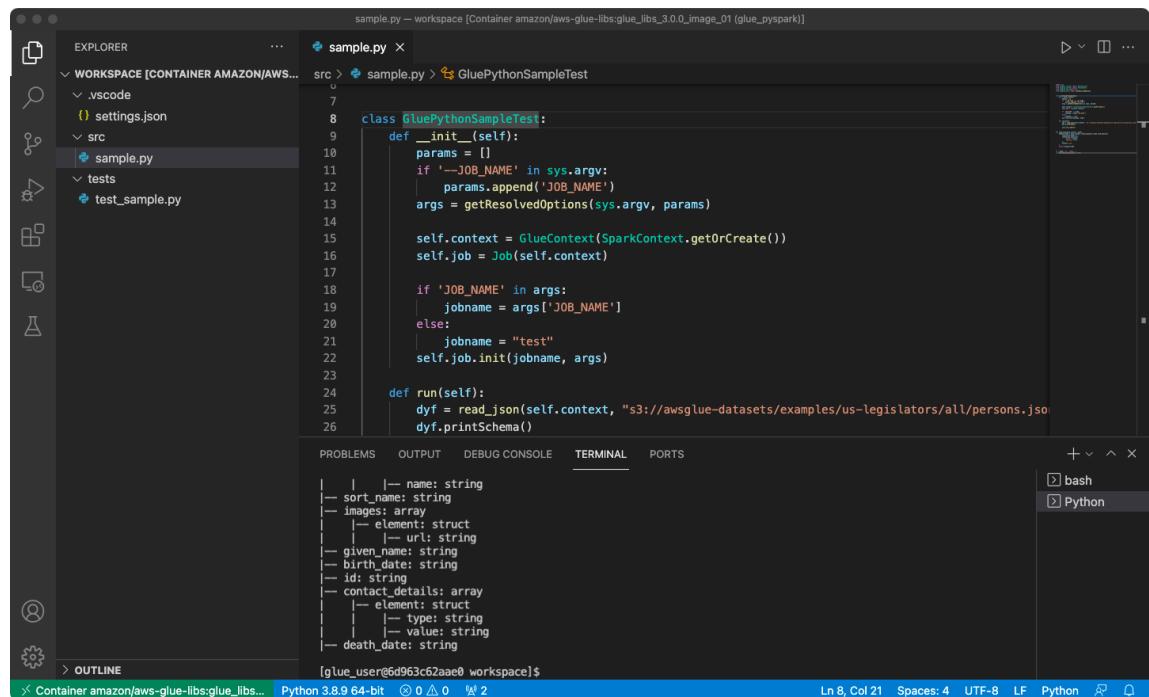
```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-libs:glue_libs_3.0.0_image_01 pyspark
```

2. Start Visual Studio Code.
3. Choose **Remote Explorer** on the left menu, and choose `amazon/aws-glue-libs:glue_libs_3.0.0_image_01`.



4. Right click and choose **Attach to Container**. If a dialog is shown, choose **Got it**.
5. Open `/home/glue_user/workspace/`.
6. Create a Glue PySpark script and choose **Run**.

You will see the successful run of the script.



Appendix: AWS Glue job sample code for testing

This appendix provides scripts as AWS Glue job sample code for testing purposes.

sample.py: Sample code to utilize the AWS Glue ETL library with an Amazon S3 API call

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

class GluePythonSampleTest:
    def __init__(self):
        params = []
        if '--JOB_NAME' in sys.argv:
            params.append('JOB_NAME')
        args = getResolvedOptions(sys.argv, params)

        self.context = GlueContext(SparkContext.getOrCreate())
        self.job = Job(self.context)

        if 'JOB_NAME' in args:
            jobname = args['JOB_NAME']
        else:
            jobname = "test"
        self.job.init(jobname, args)

    def run(self):
        df = read_json(self.context, "s3://awsglue-datasets/examples/us-legislators/all/persons.json")
        df.printSchema()

        self.job.commit()

```

```

def read_json(glue_context, path):
    dynamicframe = glue_context.create_dynamic_frame.from_options(
        connection_type='s3',
        connection_options={
            'paths': [path],
            'recurse': True
        },
        format='json'
    )
    return dynamicframe

if __name__ == '__main__':
    GluePythonSampleTest().run()

```

The above code requires Amazon S3 permissions in AWS IAM. You need to grant the IAM managed policy `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess` or an IAM custom policy which allows you to call `ListBucket` and `GetObject` for the Amazon S3 path.

`test_sample.py`: Sample code for unit test of `sample.py`.

```

import pytest
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
import sys
from src import sample

@pytest.fixture(scope="module", autouse=True)
def glue_context():
    sys.argv.append('--JOB_NAME')
    sys.argv.append('test_count')

    args = getResolvedOptions(sys.argv, ['JOB_NAME'])
    context = GlueContext(SparkContext.getOrCreate())
    job = Job(context)
    job.init(args['JOB_NAME'], args)

    yield(context)

    job.commit()

def test_counts(glue_context):
    dyf = sample.read_json(glue_context, "s3://awsglue-datasets/examples/us-legislators/all/persons.json")
    assert dyf.toDF().count() == 1961

```

Developing using the AWS Glue ETL library

The AWS Glue ETL 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 all AWS Glue versions, including AWS Glue version 0.9 and AWS Glue version 1.0 and later. For information about the versions of Python and Apache Spark that are available with AWS Glue, see the [Glue version job property \(p. 198\)](#).

The library is released with the Amazon Software license (<https://aws.amazon.com/asl>).

Note

The instructions in this section have not been tested on Microsoft Windows operating systems. For local development and testing on Windows platforms, see the blog [Building an AWS Glue ETL pipeline locally without an AWS account](#)

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="parquet"](#) (p. 507))
 - [FillMissingValues transform](#) ([Scala](#) or [Python](#))

These feature are available only within the AWS Glue job system.

- The [FindMatches transform](#) is not supported with local development.

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. Clone the AWS Glue Python repository from GitHub (<https://github.com/awslabs/aws-glue-libs>).
2. Do one of the following:
 - For AWS Glue version 0.9, check out branch `glue-0.9`.
 - For AWS Glue versions 1.0, check out branch `glue-1.0`. All versions above AWS Glue 0.9 support Python 3.
 - For AWS Glue versions 2.0, check out branch `glue-2.0`.
 - For AWS Glue version 3.0, check out the `master` branch.
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 AWS Glue version 0.9: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>
 - For AWS Glue version 1.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - For AWS Glue version 2.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - For AWS Glue version 3.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz>
5. Export the `SPARK_HOME` environment variable, setting it to the root location extracted from the Spark archive. For example:
 - For AWS Glue version 0.9: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
 - For AWS Glue version 1.0 and 2.0: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`

- For AWS Glue version 3.0: `export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`

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
AWS Glue Shell	<code>./bin/gluepyspark</code>	Enter and run Python scripts in a shell that integrates with AWS Glue ETL libraries.
AWS 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 AWS Glue version 0.9: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>
 - For AWS Glue version 1.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - For AWS Glue version 2.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - For AWS Glue version 3.0: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz>
3. Export the `SPARK_HOME` environment variable, setting it to the root location extracted from the Spark archive. For example:
 - For AWS Glue version 0.9: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
 - For AWS Glue version 1.0 and 2.0: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
 - For AWS Glue version 3.0: `export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`

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 one of the following:

- 3.0.0 for AWS Glue version 3.0
- 1.0.0 for AWS Glue version 1.0 or 2.0
- 0.9.0 for AWS 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 ETL application</description>

    <properties>
        <scala.version>*<2.11.1> for AWS Glue 2.0 or below, or <2.12.7> for AWS Glue 3.0*</scala.version>
        <glue.version>Glue version (as mentioned above)</glue.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-library</artifactId>
            <version>${scala.version}</version>
        <!-- A "provided" dependency, this will be ignored when you package your application -->
        <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>AWSGlueETL</artifactId>
            <version>${glue.version}</version>
            <!-- A "provided" dependency, this will be ignored when you package your application -->
            <scope>provided</scope>
        </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>
<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>
<!-- The shade plugin will be helpful in building a uberjar or fatjar.
You can use this jar in the AWS Glue runtime environment. For more information, see
https://maven.apache.org/plugins/maven-shade-plugin/ -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>3.2.4</version>
    <configuration>
        <!-- any other shade configurations -->
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>

```

```
<goals>
    <goal>shade</goal>
  </goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

Running Your Scala ETL Script

Run the following command from the Maven project root directory to run 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.

Configuring a Test Environment

For examples of configuring a local test environment, see the following blog articles:

- [Building an AWS Glue ETL pipeline locally without an AWS account](#)
- [Developing AWS Glue ETL jobs locally using a container](#)

If you want to use development endpoints or notebooks for testing your ETL scripts, see [Developing Scripts Using Development Endpoints \(p. 252\)](#).

Note

Development endpoints are not supported for use with AWS Glue version 2.0 jobs. For more information, see [Running Spark ETL Jobs with Reduced Startup Times](#).

Cross-Account Cross-Region Access to DynamoDB Tables

AWS Glue ETL jobs support both cross-region and cross-account access to DynamoDB tables. AWS Glue ETL jobs support both reading data from another AWS Account's DynamoDB table, and writing data into another AWS Account's DynamoDB table. AWS Glue also supports both reading from a DynamoDB table in another region, and writing into a DynamoDB table in another region. This section gives instructions on setting up the access, and provides an example script.

The procedures in this section reference an IAM tutorial for creating an IAM role and granting access to the role. The tutorial also discusses assuming a role, but here you will instead use a job script to assume the role in AWS Glue. This tutorial also contains information about general cross-account practices. For more information, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the *IAM User Guide*.

Create a Role

Follow [step 1 in the tutorial](#) to create an IAM role in account A. When defining the permissions of the role, you can choose to attach existing policies such as `AmazonDynamoDBReadOnlyAccess`, or `AmazonDynamoDBFullAccess` to allow the role to read/write DynamoDB. The following example shows creating a role named `DynamoDBCrossAccessRole`, with the permission policy `AmazonDynamoDBFullAccess`.

Grant Access to the Role

Follow [step 2 in the tutorial](#) in the *IAM User Guide* to allow account B to switch to the newly-created role. The following example creates a new policy with the following statement:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "<DynamoDBCrossAccessRole's ARN>"
        }
    ]
}
```

Then, you can attach this policy to the group/role/user you would like to use to access DynamoDB.

Assume the Role in the AWS Glue Job Script

Now, you can log in to account B and create an AWS Glue job. To create a job, refer to the instructions at [Adding Jobs in AWS Glue \(p. 197\)](#).

In the job script you need to use the `dynamodb sts.roleArn` parameter to assume the `DynamoDBCrossAccessRole` role. Assuming this role allows you to get the temporary credentials, which need to be used to access DynamoDB in account B. Review these example scripts.

For a cross-account read across regions (ETL connector):

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
        "dynamodb.input.tableName": "test_source",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

For a cross-account read across regions (ELT connector):

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
```

```
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.useExportConnector": true,
        "dynamodb.tableArn": "<test_source ARN>",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

For a read and cross-account write across regions:

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
        "dynamodb.input.tableName": "test_source"
    }
)
dyf.show()

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-west-2",
        "dynamodb.output.tableName": "test_sink",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)

job.commit()
```

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 \(p. 535\)](#)

- [Calling AWS Glue APIs in Python \(p. 535\)](#)
- [Using Python Libraries with AWS Glue \(p. 537\)](#)
- [AWS Glue Python Code Samples \(p. 540\)](#)

AWS Glue PySpark Extensions

AWS Glue has created the following extensions to the PySpark Python dialect.

- [Accessing Parameters Using `getResolvedOptions` \(p. 553\)](#)
- [PySpark Extension Types \(p. 554\)](#)
- [DynamicFrame Class \(p. 558\)](#)
- [DynamicFrameCollection Class \(p. 570\)](#)
- [DynamicFrameWriter Class \(p. 571\)](#)
- [DynamicFrameReader Class \(p. 573\)](#)
- [GlueContext Class \(p. 575\)](#)

AWS Glue PySpark Transforms

AWS Glue has created the following transform Classes to use in PySpark ETL operations.

- [GlueTransform Base Class \(p. 589\)](#)
- [ApplyMapping Class \(p. 591\)](#)
- [DropFields Class \(p. 592\)](#)
- [DropNullFields Class \(p. 596\)](#)
- [ErrorsAsDynamicFrame Class \(p. 597\)](#)
- [FillMissingValues Class \(p. 598\)](#)
- [Filter Class \(p. 599\)](#)
- [FindIncrementalMatches Class \(p. 601\)](#)
- [FindMatches Class \(p. 602\)](#)
- [FlatMap Class \(p. 603\)](#)
- [Join Class \(p. 604\)](#)
- [Map Class \(p. 605\)](#)
- [MapToCollection Class \(p. 608\)](#)
- [mergeDynamicFrame \(p. 563\)](#)
- [Relationalize Class \(p. 609\)](#)
- [RenameField Class \(p. 611\)](#)
- [ResolveChoice Class \(p. 612\)](#)
- [SelectFields Class \(p. 614\)](#)
- [SelectFromCollection Class \(p. 615\)](#)
- [Spigot Class \(p. 617\)](#)
- [SplitFields Class \(p. 618\)](#)
- [SplitRows Class \(p. 620\)](#)
- [Unbox Class \(p. 622\)](#)
- [UnnestFrame Class \(p. 624\)](#)

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 AWS Glue version of the job. For more information on AWS Glue versions, see the [Glue version job property \(p. 198\)](#).

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 \(Boto3\) 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 \(p. 701\)](#) 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 \(p. 845\)](#) or [JobRun Structure \(p. 856\)](#). 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']
```

If you want to pass an argument that is a nested JSON string, to preserve the parameter value as it gets passed to your AWS Glue ETL job, you must encode the parameter string before starting the job run, and then decode the parameter string before referencing it in your job script. For example, consider the following argument string:

```
glue_client.start_job_run(JobName = "gluejobname", Arguments={
    "--my_curly_braces_string": '{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
})
```

To pass this parameter correctly, you should encode the argument as a Base64 encoded string.

```
import base64
...
sample_string='{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
sample_string_bytes = sample_string.encode("ascii")

base64_bytes = base64.b64encode(sample_string_bytes)
base64_string = base64_bytes.decode("ascii")
...
glue_client.start_job_run(JobName = "gluejobname", Arguments={
    "--my_curly_braces_string": base64_bytes})
...
sample_string_bytes = base64.b64decode(base64_bytes)
sample_string = sample_string_bytes.decode("ascii")
print(f"Decoded string: {sample_string}")
```

...

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=myNewJobRun['JobRunId'])
```

5. Print the current state of the job run:

```
print status['JobRun']['JobRunState']
```

Using Python Libraries with AWS Glue

AWS Glue lets you install additional Python modules and libraries for use with AWS Glue ETL.

Topics

- [Installing Additional Python Modules in AWS Glue 2.0 with pip \(p. 537\)](#)
- [Python Modules Already Provided in AWS Glue Version 2.0 \(p. 538\)](#)
- [Zipping Libraries for Inclusion \(p. 539\)](#)
- [Loading Python Libraries in a Development Endpoint \(p. 539\)](#)
- [Using Python Libraries in a Job or JobRun \(p. 540\)](#)

Installing Additional Python Modules in AWS Glue 2.0 with pip

AWS Glue uses the Python Package Installer (`pip3`) to install additional modules to be used by AWS Glue ETL. You can use the `--additional-python-modules` option with a list of comma-separated Python modules to add a new module or change the version of an existing module. You can pass additional options specified by the `python-modules-installer-option` to `pip3` for installing the modules. Any incompatibility or limitations from `pip3` will apply.

For example to update or to add a new scikit-learn module use the following key/value: "--additional-python-modules", "scikit-learn==0.21.3".

Also, within the --additional-python-modules option you can specify an Amazon S3 path to a Python wheel module. For example:

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

You specify the --additional-python-modules option in the DefaultArguments or NonOverridableArguments job parameters, or in the **Job parameters** field of the AWS Glue console.

Python Modules Already Provided in AWS Glue Version 2.0

AWS Glue version 2.0 supports the following python modules out of the box:

- boto3==1.12.4
- botocore==1.15.4
- certifi==2019.11.28
- chardet==3.0.4
- cycler==0.10.0
- Cython==0.29.15
- docutils==0.15.2
- enum34==1.1.9
- fsspec==0.6.2
- idna==2.9
- jmespath==0.9.4
- joblib==0.14.1
- kiwisolver==1.1.0
- matplotlib==3.1.3
- mpmath==1.1.0
- numpy==1.18.1
- pandas==1.0.1
- patsy==0.5.1
- pmdarima==1.5.3
- ptvsd==4.3.2
- pyarrow==0.16.0
- pydevd==1.9.0
- pyhocon==0.3.54
- PyMySQL==0.9.3
- pyparsing==2.4.6
- python_dateutil==2.8.1
- pytz==2019.3
- requests==2.23.0
- s3fs==0.4.0
- s3transfer==0.3.3
- scikit-learn==0.22.1

- scipy==1.4.1
- setuptools==45.2.0
- setuptools==45.2.0
- six==1.14.0
- statsmodels==0.11.1
- subprocess32==3.5.4
- sympy==1.5.1
- tbats==1.0.9
- urllib3==1.25.8

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 \(Python: create_dev_endpoint\) \(p. 889\)](#), 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 AAAAB3NzaC1yc2EAAAQABAAQGtp04H/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 \(p. 888\)](#) object and setting the `UpdateEtlLibraries` parameter to `True` when calling [UpdateDevEndpoint \(update_dev_endpoint\) \(p. 893\)](#).

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 \(create_job\) \(p. 850\)](#), 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 \(p. 540\)](#)
- [Code Example: Data Preparation Using ResolveChoice, Lambda, and ApplyMapping \(p. 548\)](#)

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" \(p. 256\)](#).

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 \(p. 144\)](#), create a new crawler that can crawl the s3://aws-glue-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 from the Data in the Data Catalog

Next, you can easily create examine a DynamicFrame from the AWS Glue Data Catalog, and examine the schemas of the data. 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 type	image	org_id	org_name	links seats seats
null party	party/al null	AL	null null	
null https://upload.wi... party	party/democrat null	Democrat [[website,http://... null		
null https://upload.wi... party	party/democrat-li... null	Democrat-Liberal [[website,http://... null		
null legislature d56acebe-8fdc-47b... house	House of Represen... null		null 435 lower	
null party	party/independent null	Independent	null null	
null https://upload.wi... party	party/new_progres... null	New Progressive [[website,http://... null		
null https://upload.wi... party	party/popular_dem... null	Popular Democrat [[website,http://... null		
null https://upload.wi... party	party/republican null	Republican [[website,http://... null		
null https://upload.wi... party	party/republican-... null	Republican-Conser... [[website,http://... null		
null https://upload.wi... party	party/democrat null	Democrat [[website,http://... null		
null legislature 8fa6c3d2-71dc-478... house	Senate null	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: Transform the Data for 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	
10	1		202-225-1314
10	2	phone	
10	3		202-225-3772
10	4	twitter	
10	5		MikeRossUpdates
75	0	fax	
75	1		202-225-7856
75	2	phone	
75	3		202-225-2711
75	4	twitter	
75	5		SenCapito

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...	10	Mike	Ross	
e3c60f34-7d1b-4c0...	75	Shelley	Capito	

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.

Safely store and access your Amazon Redshift credentials with a AWS Glue connection. For information about how to create your own connection, see [the section called “Defining Connections in the AWS Glue Data Catalog” \(p. 110\)](#).

You are now ready to write your data to a connection by cycling through the `DynamicFrames` one at a time:

```
for df_name in dfc.keys():
    m_df = dfc.select(df_name)
    print "Writing to table: ", df_name
    glueContext.write_dynamic_frame.from_jdbc_conf(frame = m_df, connection settings here)
```

Your connection settings will differ based on your type of relational database:

- For instructions on writing to Amazon Redshift consult [the section called "Moving Data to and from Amazon Redshift" \(p. 513\)](#).
- For other databases, consult [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#).

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_relationalize.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 that was downloaded from two [Data.CMS.gov](#) data sets: "Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011" and "Inpatient Charge Data FY 2011". After downloading the data, we modified the dataset 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" \(p. 256\)](#).

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 \(p. 144\)](#), 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 Parsings

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|
|22862.23|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:

```
from awsglue.dynamicframe import DynamicFrame
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 \(p. 553\)](#)
- [PySpark Extension Types \(p. 554\)](#)
- [DynamicFrame Class \(p. 558\)](#)
- [DynamicFrameCollection Class \(p. 570\)](#)
- [DynamicFrameWriter Class \(p. 571\)](#)
- [DynamicFrameReader Class \(p. 573\)](#)
- [GlueContext Class \(p. 575\)](#)

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. The arguments use only underscores, not 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 \(p. 554\)](#) 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 \(p. 555\)](#) 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 \(p. 555\)](#) 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 \(p. 555\)](#) class to represent an enumeration of valid options.

__init__(options)

- **options** – A list of the options being enumerated.

Collection Types

- [ArrayType\(DataType\) \(p. 556\)](#)
- [ChoiceType\(DataType\) \(p. 556\)](#)
- [MapType\(DataType\) \(p. 556\)](#)
- [Field\(Object\) \(p. 556\)](#)
- [StructType\(DataType\) \(p. 557\)](#)
- [EntityType\(DataType\) \(p. 557\)](#)

[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 \(p. 554\)](#).

`__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\(object\) \(p. 557\)](#)
- [DataSink\(object\) \(p. 558\)](#)

[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. 559)
- `fromDF` (p. 559)
- `toDF` (p. 559)

`__init__`

`__init__(jdf, glue_ctx, name)`

- `jdf` – A reference to the data frame in the Java Virtual Machine (JVM).
- `glue_ctx` – A [GlueContext Class \(p. 575\)](#) 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 \(p. 575\)](#) 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 \(p. 560\)](#)
- [schema \(p. 560\)](#)
- [printSchema \(p. 560\)](#)
- [show \(p. 560\)](#)
- [repartition \(p. 560\)](#)
- [coalesce \(p. 560\)](#)

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 \(p. 561\)](#)
- [drop_fields \(p. 561\)](#)
- [filter \(p. 561\)](#)
- [join \(p. 562\)](#)
- [map \(p. 562\)](#)
- [mergeDynamicFrame \(p. 563\)](#)
- [relationalize \(p. 563\)](#)
- [rename_field \(p. 564\)](#)
- [resolveChoice \(p. 564\)](#)
- [select_fields \(p. 565\)](#)
- [spigot \(p. 566\)](#)
- [split_fields \(p. 566\)](#)

- [split_rows \(p. 566\)](#)
- [unbox \(p. 567\)](#)
- [unnest \(p. 567\)](#)
- [unnest_ddb_json \(p. 568\)](#)
- [write \(p. 569\)](#)

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 \(p. 603\)](#) 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 \(p. 599\)](#).

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 \(p. 605\)](#).

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. This argument is not currently used.
- `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, choice = "" , database = None ,
table_name = None , transformation_ctx="", info="", stageThreshold=0,
totalThreshold=0, catalog_id = None)
```

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: (`field_path`, `action`).

There are two ways to use `resolveChoice`. The first is to use the `specs` argument to specify a sequence of specific fields and how to resolve them. The other mode for `resolveChoice` is use the `choice` argument to specify a single resolution for all `ChoiceTypes`.

Values for `specs` are specified as tuples made up of (`field_path`, `action`) pairs. The `field_path` value identifies a specific ambiguous element, and the `action` value identifies the corresponding resolution. The following are the possible actions:

- `cast:type` – Attempts to cast all values to the specified type. For example, `cast:int`.
- `make_cols` – Converts each distinct type to a column with the name `columnName_type`. 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: type` – 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 `field_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 `field_path` to `"myList[].price"`, and the `action` to `"cast:double"`.

Note

Only one of the `specs` and `choice` parameters can be used. If the `specs` parameter is not `None`, then the `choice` parameter must be an empty string. Conversely if the `choice` is not an empty string, then the `specs` parameter must be `None`.

- `choice` – Specifies 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 previously for `specs`, this argument also supports the following action:
 - `match_catalog` – Attempts to cast each `ChoiceType` to the corresponding type in the specified Data Catalog table.
 - `database` – The Data Catalog database to use with the `match_catalog` action.
 - `table_name` – The Data Catalog table to use with the `match_catalog` action.
 - `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).
 - `catalog_id` – The catalog ID of the Data Catalog being accessed (the account ID of the Data Catalog). When set to `None` (default value), it uses the catalog ID of the calling account.

Example

```
df1 = df.resolveChoice(choice = "make_cols")
df2 = df.resolveChoice(specs = [("myList[].price", "make_struct"), ("columnA", "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 \(p. 504\)](#) 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()`

[unnest_ddb_json](#)

Unnests nested columns in a `DynamicFrame` that are specifically in the DynamoDB JSON structure, and returns a new unnested `DynamicFrame`. Columns that are of an array of struct types will not be unnested. Note that this is a specific type of unnesting transform that behaves differently from the regular `unnest` transform and requires the data to already be in the DynamoDB JSON structure. For more information, see [DynamoDB JSON](#).

```
unnest_ddb_json(transformation_ctx="", info="", stageThreshold=0,
totalThreshold=0)
```

- `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, the schema of a reading an export with the DynamoDB JSON structure might look like the following:

```
root
|-- Item: struct
|   |-- ColA: struct
|   |   |-- S: string
|   |-- ColB: struct
|   |   |-- S: string
|   |-- ColC: struct
|   |   |-- N: string
|   |-- ColD: struct
|   |   |-- L: array
|   |       |-- element: null
```

The `unnest_ddb_json()` transform would convert this to:

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
```

```
|--- ColD: array
|   |-- element: null
```

The following code example shows how to use the AWS Glue DynamoDB export connector, invoke a DynamoDB JSON unnest, and print the number of partitions:

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
unnested = dynamicFrame.unnest_ddb_json()
print(unnested.getNumPartitions())

job.commit()
```

write

```
write(connection_type, connection_options, format, format_options, accumulator_size)
```

Gets a [DataSink\(object\)](#) (p. 558) of the specified connection type from the [GlueContext Class \(p. 575\)](#) 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 \(p. 504\)](#) 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 \(p. 504\)](#) for the formats that are supported.

- `accumulator_size` – The accumulable size to use (optional).

— Errors —

- [assertErrorThreshold \(p. 570\)](#)
- [errorsAsDynamicFrame \(p. 570\)](#)
- [errorsCount \(p. 570\)](#)
- [stageErrorsCount \(p. 570\)](#)

[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 \(p. 558\)](#) 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 \(p. 558\)](#) objects.
- `glue_ctx` – A [GlueContext Class \(p. 575\)](#) 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__ \(p. 571\)](#)
- [from_options \(p. 571\)](#)
- [from_catalog \(p. 572\)](#)
- [from_jdbc_conf \(p. 573\)](#)

__init__

```
__init__(glue_context)
```

- **glue_context** – The [GlueContext Class \(p. 575\)](#) 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 \(p. 475\)](#).

- `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 \(p. 504\)](#) 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 \(p. 504\)](#) 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).
- `additional_options` – Additional options provided to AWS Glue.

To write to Lake Formation governed tables, you can use these additional options:

- `transactionId` – (String) The transaction ID at which to do the write to the Governed table. This transaction can not be already committed or aborted, or the write will fail.
- `callDeleteObjectsOnCancel` – (Boolean, optional) If set to `true` (default), AWS Glue automatically calls the `DeleteObjectsOnCancel` API after the object is written to Amazon S3. For more information, see [DeleteObjectsOnCancel](#) in the [AWS Lake Formation Developer Guide](#).

Example Example: Writing to a governed table in Lake Formation

```
txId = glueContext.start_transaction(read_only=False)
glueContext.write_dynamic_frame.from_catalog(
    frame=dyf,
    database = db,
    table_name = tbl,
    transformation_ctx = "datasource0",
    additional_options={"transactionId":txId})
...
```

```
glueContext.commit_transaction(txId)
```

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).

Example for write_dynamic_frame

This example writes the output locally using a `connection_type` of S3 with a POSIX path argument in `connection_options`, which allows writing to local storage.

```
glueContext.write_dynamic_frame.from_options(\  
frame = dyf_splitFields,\  
connection_options = {'path': '/home/glue/GlueLocalOutput/'},\  
connection_type = 's3',\  
format = 'json')
```

DynamicFrameReader Class

— Methods —

- [__init__ \(p. 573\)](#)
- [from_rdd \(p. 573\)](#)
- [from_options \(p. 574\)](#)
- [from_catalog \(p. 574\)](#)

[__init__](#)

[__init__\(glue_context\)](#)

- `glue_context` – The [GlueContext Class \(p. 575\)](#) 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 \(p. 512\)](#).

- **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 \(p. 504\)](#) 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 \(p. 504\)](#) for the formats that are supported.
- **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](#).

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 if not reading data from Redshift).
- **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 \(p. 509\)](#).
- **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 \(p. 512\)](#).

- To pass a catalog expression to filter based on the index columns, you can see the `catalogPartitionPredicate` option.

`catalogPartitionPredicate` — You can pass a catalog expression to filter based on the index columns. This pushes down the filtering to the server side. For more information, see [AWS Glue Partition Indexes](#). Note that `push_down_predicate` and `catalogPartitionPredicate` use different syntaxes. The former one uses Spark SQL standard syntax and the later one uses JSON parser.

For more information, see [Managing Partitions for ETL Output in AWS Glue \(p. 509\)](#).

- To read from Lake Formation governed tables, you can use these additional options:
 - `transactionId` – (String) The transaction ID at which to read the Governed table contents. If this transaction is not committed, the read will be treated as part of that transaction and will see its writes. If this transaction is committed, its writes will be visible in this read. If this transaction has aborted, an error will be returned. Cannot be specified along with `asOfTime`.

Note

Either `transactionId` or `asOfTime` must be set to access the governed table.

- `asOfTime` – (TimeStamp: yyyy-[m]m-[d]d hh:mm:ss) The time as of when to read the table contents. Cannot be specified along with `transactionId`.
- `query` – (Optional) A PartiQL query statement used as an input to the Lake Formation planner service. If not set, the default setting is to select all data from the table. For more details about PartiQL, see [PartiQL Support in Row Filter Expressions](#) in the [AWS Lake Formation Developer Guide](#).

Example Example: Using a PartiQL query statement when reading from a governed table in Lake Formation

```
txId = glueContext.start_transaction(read_only=False)
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = db,
    table_name = tbl,
    transformation_ctx = "datasource0",
    additional_options={
        "transactionId":txId,
        "query": "SELECT * from tblName WHERE partitionKey=value;"})
...
glueContext.commit_transaction(txId)
```

GlueContext Class

Wraps the Apache Spark [SparkContext](#) object, and thereby provides mechanisms for interacting with the Apache Spark platform.

`__init__`

`__init__(sparkContext)`

- `sparkContext` – The Apache Spark context to use.

Creating

- [__init__ \(p. 575\)](#)
- [getSource \(p. 576\)](#)
- [create_dynamic_frame_from_rdd \(p. 576\)](#)
- [create_dynamic_frame_from_catalog \(p. 577\)](#)
- [create_dynamic_frame_from_options \(p. 577\)](#)
- [create_sample_dynamic_frame_from_catalog \(p. 578\)](#)
- [create_sample_dynamic_frame_from_options \(p. 578\)](#)
- [add_ingestion_time_columns \(p. 579\)](#)
- [create_data_frame_from_catalog \(p. 579\)](#)
- [create_data_frame_from_options \(p. 580\)](#)
- [foreachBatch \(p. 581\)](#)

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 \(p. 475\)](#).

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 Data 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 \(p. 509\)](#).
- `additional_options` – A collection of optional name-value pairs. The possible options include those listed in [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#) except for `endpointUrl`, `streamName`, `bootstrap.servers`, `security.protocol`, `topicName`, `classification`, and `delimiter`. Another supported option is `catalogPartitionPredicate`:

`catalogPartitionPredicate` — You can pass a catalog expression to filter based on the index columns. This pushes down the filtering to the server side. For more information, see [AWS Glue Partition Indexes](#). Note that `push_down_predicate` and `catalogPartitionPredicate` use different syntaxes. The former one uses Spark SQL standard syntax and the later one uses JSON parser.

- `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 \(p. 475\)](#).

- `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 \(p. 504\)](#) 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 \(p. 504\)](#) for the formats that are supported.
- `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](#).

`create_sample_dynamic_frame_from_catalog`

```
create_sample_dynamic_frame_from_catalog(database, table_name, num,
                                         redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "",
                                         additional_options = {}, sample_options = {}, catalog_id = None)
```

Returns a sample `DynamicFrame` that is created using a Data Catalog database and table name. The `DynamicFrame` only contains first `num` records from a datasource.

- `database` – The database to read from.
- `table_name` – The name of the table to read from.
- `num` – The maximum number of records in the returned sample dynamic frame.
- `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 \(p. 509\)](#).
- `additional_options` – A collection of optional name-value pairs. The possible options include those listed in [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#) except for `endpointUrl`, `streamName`, `bootstrap.servers`, `security.protocol`, `topicName`, `classification`, and `delimiter`.
- `sample_options` – Parameters to control sampling behavior (optional). Current available parameters for Amazon S3 sources:
 - `maxSamplePartitions` – The maximum number of partitions the sampling will read. Default value is 10
 - `maxSampleFilesPerPartition` – The maximum number of files the sampling will read in one partition. Default value is 10.

These parameters help to reduce the time consumed by file listing. For example, suppose the dataset has 1000 partitions, and each partition has 10 files. If you set `maxSamplePartitions = 10`, and `maxSampleFilesPerPartition = 10`, instead of listing all 10,000 files, the sampling will only list and read the first 10 partitions with the first 10 files in each: $10 \times 10 = 100$ files in total.

- `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.

`create_sample_dynamic_frame_from_options`

```
create_sample_dynamic_frame_from_options(connection_type,
                                         connection_options={}, num, sample_options={}, format=None,
                                         format_options={}, transformation_ctx = "")
```

Returns a sample `DynamicFrame` created with the specified connection and format. The `DynamicFrame` only contains first `num` records from a datasource.

- `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 more information, see [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#).

- `num` – The maximum number of records in the returned sample dynamic frame.
- `sample_options` – Parameters to control sampling behavior (optional). Current available parameters for Amazon S3 sources:
 - `maxSamplePartitions` – The maximum number of partitions the sampling will read. Default value is 10
 - `maxSampleFilesPerPartition` – The maximum number of files the sampling will read in one partition. Default value is 10.

These parameters help to reduce the time consumed by file listing. For example, suppose the dataset has 1000 partitions, and each partition has 10 files. If you set `maxSamplePartitions = 10`, and `maxSampleFilesPerPartition = 10`, instead of listing all 10,000 files, the sampling will only list and read the first 10 partitions with the first 10 files in each: $10 \times 10 = 100$ files in total.

- `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 \(p. 504\)](#) 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 \(p. 504\)](#) for the formats that are supported.
- `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 \(p. 509\)](#).

[add_ingestion_time_columns](#)

```
add_ingestion_time_columns(dataFrame, timeGranularity = "")
```

Appends ingestion time columns like `ingest_year`, `ingest_month`, `ingest_day`, `ingest_hour`, `ingest_minute` to the input `DataFrame`. This function is automatically generated in the script generated by the AWS Glue when you specify a Data Catalog table with Amazon S3 as the target. This function automatically updates the partition with ingestion time columns on the output table. This allows the output data to be automatically partitioned on ingestion time without requiring explicit ingestion time columns in the input data.

- `dataFrame` – The `DataFrame` to append the ingestion time columns to.
- `timeGranularity` – The granularity of the time columns. Valid values are "day", "hour" and "minute". For example, if "hour" is passed in to the function, the original `DataFrame` will have "`ingest_year`", "`ingest_month`", "`ingest_day`", and "`ingest_hour`" time columns appended.

Returns the data frame after appending the time granularity columns.

Example:

```
dynamic_frame = DynamicFrame.fromDF(glueContext.add_ingestion_time_columns(dataFrame,
    "hour"))
```

[create_data_frame_from_catalog](#)

```
create_data_frame_from_catalog(database, table_name, transformation_ctx = "",
    additional_options = {})
```

Returns a `DataFrame` that is created using information from a Data Catalog table. Use this function only with AWS Glue streaming sources.

- `database` – The Data Catalog database to read from.

- `table_name` – The name of the Data Catalog table to read from.
- `transformation_ctx` – The transformation context to use (optional).
- `additional_options` – A collection of optional name-value pairs. The possible options include those listed in [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#) for streaming sources, such as `startingPosition`, `maxFetchTimeInMs`, and `startingOffsets`.

Example:

```
df = glueContext.create_data_frame.from_catalog(
    database = "MyDB",
    table_name = "streaming_table",
    transformation_ctx = "df",
    additional_options = {"startingPosition": "TRIM_HORIZON", "inferSchema": "true"})
```

create_data_frame_from_options

```
create_data_frame_from_options(connection_type, connection_options={},  
format=None, format_options={}, transformation_ctx = "")
```

Returns a DataFrame created with the specified connection and format. Use this function only with AWS Glue streaming sources.

- `connection_type` – The streaming connection type. Valid values include `kinesis` and `kafka`.
- `connection_options` – Connection options, which are different for Kinesis and Kafka. You can find the list of all connection options for each streaming data source at [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#). Note the following differences in streaming connection options:
 - Kinesis streaming sources require `streamARN`, `startingPosition`, `inferSchema`, and `classification`.
 - Kafka streaming sources require `connectionName`, `topicName`, `startingOffsets`, `inferSchema`, and `classification`.
- `format` – A format specification (optional). This is used for an Amazon S3 or an AWS Glue connection that supports multiple formats. For information about the supported formats, see [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 504\)](#).
- `format_options` – Format options for the specified format. For information about the supported format options, see [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 504\)](#).
- `transformation_ctx` – The transformation context to use (optional).

Example for Amazon Kinesis streaming source:

```
kinesis_options =  
    { "streamARN": "arn:aws:kinesis:us-east-2:77778889999:stream/fromOptionsStream",  
      "startingPosition": "TRIM_HORIZON",  
      "inferSchema": "true",  
      "classification": "json"  
    }  
data_frame_datasource0 =  
    glueContext.create_data_frame.from_options(connection_type="kinesis",  
                                              connection_options=kinesis_options)
```

Example for Kafka streaming source:

```
kafka_options =  
    { "connectionName": "ConfluentKafka",
```

```

        "topicName": "kafka-auth-topic",
        "startingOffsets": "earliest",
        "inferSchema": "true",
        "classification": "json"
    }
data_frame_datasource0 =
    glueContext.create_data_frame.from_options(connection_type="kafka",
connection_options=kafka_options)

```

forEachBatch

forEachBatch(frame, batch_function, options)

Applies the `batch_function` passed in to every micro batch that is read from the Streaming source.

- `frame` – The DataFrame containing the current micro batch.
- `batch_function` – A function that will be applied for every micro batch.
- `options` – A collection of key-value pairs that holds information about how to process micro batches. The following options are required:
 - `windowSize` – The amount of time to spend processing each batch.
 - `checkpointLocation` – The location where checkpoints are stored for the streaming ETL job.
 - `batchMaxRetries` – The maximum number of times to retry the batch if it fails. The default value is 3. This option is only configurable for Glue version 2.0 and above.

Example:

```

glueContext.forEachBatch(
    frame = data_frame_datasource0,
    batch_function = processBatch,
    options = {
        "windowSize": "100 seconds",
        "checkpointLocation": "s3://kafka-auth-dataplane/confluent-test/output/checkpoint/"
    }
)

def processBatch(data_frame, batchId):
    if (data_frame.count() > 0):
        datasource0 = DynamicFrame.fromDF(
            glueContext.add_ingestion_time_columns(data_frame, "hour"),
            glueContext, "from_data_frame"
        )
        additionalOptions_datasink1 = {"enableUpdateCatalog": True}
        additionalOptions_datasink1["partitionKeys"] = ["ingest_yr", "ingest_mo",
"ingest_day"]
        datasink1 = glueContext.write_dynamic_frame.from_catalog(
            frame = datasource0,
            database = "tempdb",
            table_name = "kafka-auth-table-output",
            transformation_ctx = "datasink1",
            additional_options = additionalOptions_datasink1
        )

```

Working with Datasets in Amazon S3

- [purge_table \(p. 582\)](#)
- [purge_s3_path \(p. 582\)](#)
- [transition_table \(p. 583\)](#)

- [transition_s3_path \(p. 584\)](#)

[purge_table](#)

```
purge_table(catalog_id=None, database="", table_name="", options={}, transformation_ctx="")
```

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 turn on [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.

- **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.
- **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.

[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 turn on [object versioning](#) on the Amazon S3 bucket. When an object is deleted from a bucket that doesn't have object versioning turned on, the object can't be recovered. For more information about how to recover deleted objects in a bucket with versioning, 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.

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 Amazon Web Services 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 Amazon Web Services 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"})
```

Extracting

- [extract_jdbc_conf \(p. 584\)](#)

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.

Transactions

- [start_transaction \(p. 585\)](#)
- [commit_transaction \(p. 585\)](#)
- [cancel_transaction \(p. 585\)](#)

[start_transaction](#)

start_transaction(read_only)

Start a new transaction. Internally calls the Lake Formation [startTransaction](#) API.

- `read_only` – (Boolean) Indicates whether this transaction should be read only or read and write. Writes made using a read-only transaction ID will be rejected. Read-only transactions do not need to be committed.

Returns the transaction ID.

[commit_transaction](#)

commit_transaction(transaction_id, wait_for_commit = True)

Attempts to commit the specified transaction. `commit_transaction` may return before the transaction has finished committing. Internally calls the Lake Formation [commitTransaction](#) API.

- `transaction_id` – (String) The transaction to commit.
- `wait_for_commit` – (Boolean) Determines whether the `commit_transaction` returns immediately. The default value is true. If false, `commit_transaction` polls and waits until the transaction is committed. The amount of wait time is restricted to 1 minute using exponential backoff with a maximum of 6 retry attempts.

Returns a Boolean to indicate whether the commit is done or not.

[cancel_transaction](#)

cancel_transaction(transaction_id)

Attempts to cancel the specified transaction. Returns a `TransactionCommittedException` exception if the transaction was previously committed. Internally calls the Lake Formation [CancelTransaction](#) API.

- `transaction_id` – (String) The transaction to cancel.

Writing

- [getSink \(p. 586\)](#)
- [write_dynamic_frame_from_options \(p. 586\)](#)
- [write_from_options \(p. 587\)](#)
- [write_dynamic_frame_from_catalog \(p. 587\)](#)
- [write_dynamic_frame_from_jdbc_conf \(p. 588\)](#)
- [write_from_jdbc_conf \(p. 588\)](#)

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 name-value pairs used to specify the connection options. Some of the possible values are:
 - `user` and `password`: For authorization
 - `url`: The endpoint for the data store
 - `dbtable`: The name of the target table
 - `bulkSize`: Degree of parallelism for insert operations

The options that you can specify depends on the connection type. See [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#) for additional values and examples.

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 \(p. 475\)](#).

- **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 \(p. 504\)](#) 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 \(p. 504\)](#) 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 \(p. 475\)](#).

- **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 \(p. 504\)](#) 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 \(p. 504\)](#) 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 information from a Data Catalog database and table.

- **frame** – The DynamicFrame to write.
- **Database** – The Data Catalog database that contains the table.

- `table_name` – The name of the Data Catalog table associated with the target.
- `redshift_tmp_dir` – An Amazon Redshift temporary directory to use (optional).
- `transformation_ctx` – The transformation context to use (optional).
- `additional_options` – A collection of optional name-value pairs.
- `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 \(p. 475\)](#).
- `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 \(p. 475\)](#).
- `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.

AWS Glue PySpark Transforms Reference

AWS Glue has created the following transform Classes to use in PySpark ETL operations.

- [GlueTransform Base Class \(p. 589\)](#)
- [ApplyMapping Class \(p. 591\)](#)
- [DropFields Class \(p. 592\)](#)
- [DropNullFields Class \(p. 596\)](#)

- [ErrorsAsDynamicFrame Class \(p. 597\)](#)
- [FillMissingValues Class \(p. 598\)](#)
- [Filter Class \(p. 599\)](#)
- [FindIncrementalMatches Class \(p. 601\)](#)
- [FindMatches Class \(p. 602\)](#)
- [FlatMap Class \(p. 603\)](#)
- [Join Class \(p. 604\)](#)
- [Map Class \(p. 605\)](#)
- [MapToCollection Class \(p. 608\)](#)
- [mergeDynamicFrame \(p. 563\)](#)
- [Relationalize Class \(p. 609\)](#)
- [RenameField Class \(p. 611\)](#)
- [ResolveChoice Class \(p. 612\)](#)
- [SelectFields Class \(p. 614\)](#)
- [SelectFromCollection Class \(p. 615\)](#)
- [Spigot Class \(p. 617\)](#)
- [SplitFields Class \(p. 618\)](#)
- [SplitRows Class \(p. 620\)](#)
- [Unbox Class \(p. 622\)](#)
- [UnnestFrame Class \(p. 624\)](#)

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\(cls, *args, **kwargs\) \(p. 589\)](#)
- [name\(cls\) \(p. 589\)](#)
- [describeArgs\(cls\) \(p. 590\)](#)
- [describeReturn\(cls\) \(p. 590\)](#)
- [describeTransform\(cls\) \(p. 590\)](#)
- [describeErrors\(cls\) \(p. 590\)](#)
- [describe\(cls\) \(p. 591\)](#)

`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__ \(p. 591\)](#)
- [apply \(p. 592\)](#)
- [name \(p. 592\)](#)
- [describeArgs \(p. 592\)](#)
- [describeReturn \(p. 592\)](#)
- [describeTransform \(p. 592\)](#)
- [describeErrors \(p. 592\)](#)
- [describe \(p. 592\)](#)

`__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 ".", square brackets "[]" or parentheses "()" in the name, the mapping will not work unless you place back-ticks "`~`" around it. For example, to map `this.old.name` (`string`) to `thisNewName` (`string`) and `[id]` (`long`) to `id` (`long`), you would use the following tuples:

```
("`this.old.name`", "string", "thisNewName", "string"), ("`[id]`", "long", "id", "long")
```

- `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 \(p. 589\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 591\)](#).

[Example for ApplyMapping](#)

This example calls `ApplyMapping` for a `DynamicFrame` that uses the following schema:

```
order_schema = StructType([ StructField("order_id", StringType()),  
    StructField("customer_id", StringType()),  
    StructField("essential_item", StringType()), StructField("timestamp", StringType()),  
    StructField("zipcode", StringType()) ])
```

This example changes some of the `String` columns to `Long` format to save storage space. It also shortens the name of the column `zipcode` to `zip`.

```
dyf_applyMapping = ApplyMapping.apply( frame = dyf_orders, mappings =  
    [ ("order_id", "String", "order_id", "Long"),  
    ("customer_id", "String", "customer_id", "Long"),  
    ("essential_item", "String", "essential_item", "String"), ("timestamp", "String",  
    "timestamp", "Long"), ("zipcode", "String", "zip", "Long") ])
```

[DropFields Class](#)

Drops fields within a `DynamicFrame`.

Methods

- [__call__ \(p. 593\)](#)
- [apply \(p. 593\)](#)
- [name \(p. 593\)](#)
- [describeArgs \(p. 593\)](#)
- [describeReturn \(p. 593\)](#)
- [describeTransform \(p. 593\)](#)
- [describeErrors \(p. 593\)](#)
- [describe \(p. 593\)](#)

[`__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 \(p. 589\)](#).

[`name\(cls\)`](#)

Inherited from `GlueTransform` [name \(p. 589\)](#).

[`describeArgs\(cls\)`](#)

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

[`describeReturn\(cls\)`](#)

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

[`describeTransform\(cls\)`](#)

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

[`describeErrors\(cls\)`](#)

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

[`describe\(cls\)`](#)

Inherited from `GlueTransform` [describe \(p. 591\)](#).

Examples

- [Example dataset \(p. 594\)](#)
- [Drop a top-level field \(p. 594\)](#)
- [Drop a nested field \(p. 595\)](#)
- [Drop a nested field in an array \(p. 595\)](#)

Dataset used for DropFields examples

The following dataset is used for the DropFields examples:

```
{name: Sally, age: 23, location: {state: WY, county: Fremont}, friends: []}
  {name: Varun, age: 34, location: {state: NE, county: Douglas}, friends: [{name: Arjun,
  age: 3}]}
    {name: George, age: 52, location: {state: NY}, friends: [{name: Fred}, {name: Amy, age:
  15}]}
      {name: Haruki, age: 21, location: {state: AK, county: Denali}}
        {name: Sheila, age: 63, friends: [{name: Nancy, age: 22}]}
```

This dataset has the following schema:

```
root
  |-- name: string
  |-- age: int
  |-- location: struct
  |   |-- state: string
  |   |-- county: string
  |-- friends: array
  |   |-- element: struct
  |   |   |-- name: string
  |   |   |-- age: int
```

Example: Drop a top-level field

Use code similar to the following to drop the age field:

```
df_no_age = DropFields.apply(df, paths=['age'])
```

Resulting dataset:

```
{name: Sally, location: {state: WY, county: Fremont}, friends: []}
  {name: Varun, location: {state: NE, county: Douglas}, friends: [{name: Arjun, age: 3}]}
    {name: George, location: {state: NY}, friends: [{name: Fred}, {name: Amy, age: 15}]}
      {name: Haruki, location: {state: AK, county: Denali}}
        {name: Sheila, friends: [{name: Nancy, age: 22}]}
```

Resulting schema:

```
root
  |-- name: string
  |-- location: struct
  |   |-- state: string
  |   |-- county: string
  |-- friends: array
  |   |-- element: struct
  |   |   |-- name: string
```

```
| | | -- age: int
```

Example: Drop a nested field

To drop a nested field, you can qualify the field with a `'.'`.

```
df_no_county = DropFields.apply(df, paths=['location.county'])
```

Resulting dataset:

```
{name: Sally, age: 23, location: {state: WY}, friends: []}
  {name: Varun, age: 34, location: {state: NE}, friends: [{name: Arjun, age: 3}]}
  {name: George, age: 52, location: {state: NY}, friends: [{name: Fred}, {name: Amy, age: 15}]}
  {name: Haruki, age: 21, location: {state: AK}}
  {name: Sheila, age: 63, friends: [{name: Nancy, age: 22}]}
```

If you drop the last element of a `struct` type, the transform removes the entire `struct`.

```
df_no_county = DropFields.apply(df, paths=['location.state'])
```

Resulting schema:

```
root
  |-- name: string
  |-- age: int
  |-- friends: array
    |   |-- element: struct
    |   |   |-- name: string
    |   |   |-- age: int
```

Example: Drop a nested field from an array

No special syntax is needed to drop a field from inside a `struct` nested inside an `array`. For example, we can drop the `age` field from the `friends` array with the following:

```
df_no_friend_age = DropFields.apply(df, paths=['friends.age'])
```

Resulting dataset:

```
{name: Sally, age: 23, location: {state: WY, county: Fremont}}
  {name: Varun, age: 34, location: {state: NE, county: Douglas}, friends: [{name: Arjun}]}
  {name: George, age: 52, location: {state: NY}, friends: [{name: Fred}, {name: Amy}]}
  {name: Haruki, age: 21, location: {state: AK, county: Denali}}
  {name: Sheila, age: 63, friends: [{name: Nancy}]}
```

Resulting schema:

```
root
  |-- name: string
  |-- age: int
  |-- location: struct
    |   |-- state: string
    |   |-- county: string
  |-- friends: array
    |   |-- element: struct
    |   |   |-- name: string
```

Example for DropFields

The backticks (`) around `.zip` in the following example are needed because the column name contains a period (.)�.

```
dyf_dropfields = DropFields.apply(frame = dyf_join, paths = "`.zip`")
```

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__ \(p. 596\)](#)
- [apply \(p. 596\)](#)
- [name \(p. 596\)](#)
- [describeArgs \(p. 596\)](#)
- [describeReturn \(p. 597\)](#)
- [describeTransform \(p. 597\)](#)
- [describeErrors \(p. 597\)](#)
- [describe \(p. 597\)](#)

`__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`

[Example for DropNullFields](#)

The `DropNullFields` function automatically identifies the columns with `null` values and drops them from the resulting `DynamicFrame`.

```
dyf_dropNullfields = DropNullFields.apply(frame = dyf_unnest)
```

[ErrorsAsDynamicFrame Class](#)

Returns a `DynamicFrame` that contains nested error records leading up to the creation of the source `DynamicFrame`.

[Methods](#)

- [__call__ \(p. 597\)](#)
- [apply \(p. 597\)](#)
- [name \(p. 598\)](#)
- [describeArgs \(p. 598\)](#)
- [describeReturn \(p. 598\)](#)
- [describeTransform \(p. 598\)](#)
- [describeErrors \(p. 598\)](#)
- [describe \(p. 598\)](#)

[__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`

[FillMissingValues Class](#)

The `FillMissingValues` class locates null values and empty strings in a specified `DynamicFrame` and uses machine learning methods, such as linear regression and random forest, to predict the missing values. The ETL job uses the values in the input dataset to train the machine learning model, which then predicts what the missing values should be.

Tip

If you use incremental data sets, then each incremental set is used as the training data for the machine learning model, so the results might not be as accurate.

To import:

```
from awsglueml.transforms import FillMissingValues
```

Methods

- [apply \(p. 598\)](#)

`apply(frame, missing_values_column, output_column = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

Fills a dynamic frame's missing values in a specified column and returns a new frame with estimates in a new column. For rows without missing values, the specified column's value is duplicated to the new column.

- `frame` – The `DynamicFrame` in which to fill missing values. Required.

- `missing_values_column` – The column containing missing values (null values and empty strings). Required.
- `output_column` – The name of the new column that will contain estimated values for all rows whose value was missing. Optional; the default is the name of `missing_values_column` suffixed by `_filled`.
- `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 one additional column that contains estimations for rows with missing values and the present value for other rows.

Filter Class

Builds a new `DynamicFrame` by selecting records from the input `DynamicFrame` that satisfy a specified predicate function.

Methods

- [__call__ \(p. 599\)](#)
- [apply \(p. 600\)](#)
- [name \(p. 600\)](#)
- [describeArgs \(p. 600\)](#)
- [describeReturn \(p. 600\)](#)
- [describeTransform \(p. 600\)](#)
- [describeErrors \(p. 600\)](#)
- [describe \(p. 600\)](#)
- [Example Code \(p. 600\)](#)

`__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 \(p. 589\)](#).

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 591\)](#).

AWS Glue Python Example

This example filters sample data using the `Filter` transform and a simple Lambda function. The dataset that is used in this example consists of Medicare Provider payment data that was downloaded from two [Data.CMS.gov](#) data sets: "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 \(p. 548\)](#).

Begin by creating a `DynamicFrame` for the data:

```
%pyspark
import sys
from awsglue.context import GlueContext
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
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`

FindIncrementalMatches Class

Identifies matching records in the existing and incremental `DynamicFrame` and creates a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

To import:

```
from awsglueml.transforms import FindIncrementalMatches
```

Methods

- [apply \(p. 602\)](#)

```
apply(existingFrame, incrementalFrame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = None, computeMatchConfidenceScores = 0)
```

Identifies matching records in the input `DynamicFrame` and creates a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

- `existingFrame` – The existing and pre-matched `DynamicFrame` to apply the `FindIncrementalMatches` transform. Required.
- `incrementalFrame` – The incremental `DynamicFrame` to apply the `FindIncrementalMatches` transform to match against the `existingFrame`. Required.
- `transformId` – The unique ID associated with the `FindIncrementalMatches` transform to apply on records in the `DynamicFrames`. Required.
- `transformation_ctx` – A unique string that is used to identify stats/state information. Optional.
- `info` – A string to be 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.
- `enforcedMatches` – The `DynamicFrame` used to enforce matches. Optional. The default is `None`.
- `computeMatchConfidenceScores` – A Boolean value indicating whether to compute a confidence score for each group of matching records. Optional. The default is `false`.

Returns a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

FindMatches Class

Identifies matching records in the input `DynamicFrame` and creates a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

To import:

```
from awsglueml.transforms import FindMatches
```

Methods

- [apply \(p. 602\)](#)

```
apply(frame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = None, computeMatchConfidenceScores = 0)
```

Identifies matching records in the input `DynamicFrame` and creates a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

- `frame` – The `DynamicFrame` to apply the `FindMatches` transform. Required.

- `transformId` – The unique ID associated with the `FindMatches` transform to apply on records in the `DynamicFrame`. Required.
- `transformation_ctx` – A unique string that is used to identify stats/state information. Optional.
- `info` – A string to be 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.
- `enforcedMatches` – The `DynamicFrame` used to enforce matches. Optional. The default is `None`.
- `computeMatchConfidenceScores` – A Boolean value indicating whether to compute a confidence score for each group of matching records. Optional. The default is `false`.

Returns a new `DynamicFrame` with a unique identifier assigned to each group of matching records.

FlatMap Class

Applies a transform to each `DynamicFrame` in a collection and flattens the results.

Methods

- [__call__ \(p. 603\)](#)
- [apply \(p. 603\)](#)
- [name \(p. 603\)](#)
- [describeArgs \(p. 604\)](#)
- [describeReturn \(p. 604\)](#)
- [describeTransform \(p. 604\)](#)
- [describeErrors \(p. 604\)](#)
- [describe \(p. 604\)](#)

[`__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 \(p. 589\)](#).

[`name\(cls\)`](#)

Inherited from `GlueTransform` [name \(p. 589\)](#).

[describeArgs\(`cls`\)](#)

Inherited from `GlueTransform` [describeArgs](#) (p. 590).

[describeReturn\(`cls`\)](#)

Inherited from `GlueTransform` [describeReturn](#) (p. 590).

[describeTransform\(`cls`\)](#)

Inherited from `GlueTransform` [describeTransform](#) (p. 590).

[describeErrors\(`cls`\)](#)

Inherited from `GlueTransform` [describeErrors](#) (p. 590).

[describe\(`cls`\)](#)

Inherited from `GlueTransform` [describe](#) (p. 591).

Join Class

Performs an equality join on two `DynamicFrames`.

Methods

- [__call__](#) (p. 604)
- [apply](#) (p. 604)
- [name](#) (p. 605)
- [describeArgs](#) (p. 605)
- [describeReturn](#) (p. 605)
- [describeTransform](#) (p. 605)
- [describeErrors](#) (p. 605)
- [describe](#) (p. 605)

[__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. 589)

[name\(cls\)](#)

Inherited from [GlueTransform](#) [name \(p. 589\)](#)

[describeArgs\(cls\)](#)

Inherited from [GlueTransform](#) [describeArgs \(p. 590\)](#)

[describeReturn\(cls\)](#)

Inherited from [GlueTransform](#) [describeReturn \(p. 590\)](#)

[describeTransform\(cls\)](#)

Inherited from [GlueTransform](#) [describeTransform \(p. 590\)](#)

[describeErrors\(cls\)](#)

Inherited from [GlueTransform](#) [describeErrors \(p. 590\)](#)

[describe\(cls\)](#)

Inherited from [GlueTransform](#) [describe \(p. 591\)](#)

Example for Join

The Join function manages duplicate columns. Each dataset has a column named zip. AWS Glue adds a period (.) to one of the duplicate column names to avoid errors.

```
dyf_join = Join.apply(dyf_json, dyf_selectFields, 'zip', 'zip')
dyf_join.toDF().show()

+-----+-----+
| customers| .zip| zip|
+-----+-----+
|[108 Park Street...|75091|75091|
|[66 P Street, NY...|75023|75023|
|[708 Fed Ln, CA,...|90093|90093|
+-----+-----+
```

Map Class

Builds a new [DynamicFrame](#) by applying a function to all records in the input [DynamicFrame](#).

Methods

- [__call__ \(p. 606\)](#)
- [apply \(p. 606\)](#)
- [name \(p. 606\)](#)
- [describeArgs \(p. 606\)](#)
- [describeReturn \(p. 606\)](#)
- [describeTransform \(p. 606\)](#)
- [describeErrors \(p. 606\)](#)
- [describe \(p. 606\)](#)

- [Example Code \(p. 607\)](#)

`__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 \(p. 589\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 589\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 591\)](#).

AWS Glue Python Example

This example uses the `Map` transform to merge several fields into one `struct` type. The dataset that is used in this example consists of Medicare Provider payment data that was downloaded from two [Data.CMS.gov](#) data sets: "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 \(p. 548\)](#).

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__ \(p. 609\)](#)
- [apply \(p. 609\)](#)
- [name \(p. 609\)](#)
- [describeArgs \(p. 609\)](#)
- [describeReturn \(p. 609\)](#)
- [describeTransform \(p. 609\)](#)
- [describeErrors \(p. 609\)](#)

- [describe \(p. 609\)](#)

`__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 \(p. 589\)](#)

`name(cls)`

Inherited from `GlueTransform` [name \(p. 589\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 591\)](#).

Relationalize Class

Flattens nested schema in a `DynamicFrame` and pivots out array columns from the flattened frame.

Methods

- [__call__ \(p. 610\)](#)
- [apply \(p. 610\)](#)
- [name \(p. 610\)](#)
- [describeArgs \(p. 610\)](#)
- [describeReturn \(p. 610\)](#)

- [describeTransform \(p. 610\)](#)
- [describeErrors \(p. 610\)](#)
- [describe \(p. 610\)](#)

`__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. Currently unused.
- `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. 589)`.

`name(cls)`

Inherited from `GlueTransform name (p. 589)`.

`describeArgs(cls)`

Inherited from `GlueTransform describeArgs (p. 590)`.

`describeReturn(cls)`

Inherited from `GlueTransform describeReturn (p. 590)`.

`describeTransform(cls)`

Inherited from `GlueTransform describeTransform (p. 590)`.

`describeErrors(cls)`

Inherited from `GlueTransform describeErrors (p. 590)`.

`describe(cls)`

Inherited from `GlueTransform describe (p. 591)`.

Example for Relationalize

The Relationalize function can flatten nested structures and create multiple dynamic frames. In this example, the customer column is a nested structure, and Relationalize converts it into multiple flattened DynamicFrames.

```
dyf_relationize = dyf_orders.relationize("root", "/home/glue/GlueLocalOutput")
```

The result is a collection of DynamicFrames. Use the SelectFromCollection function to retrieve a specific DynamicFrame from the result.

RenameField Class

Renames a node within a `DynamicFrame`.

Methods

- [__call__ \(p. 611\)](#)
- [apply \(p. 611\)](#)
- [name \(p. 612\)](#)
- [describeArgs \(p. 612\)](#)
- [describeReturn \(p. 612\)](#)
- [describeTransform \(p. 612\)](#)
- [describeErrors \(p. 612\)](#)
- [describe \(p. 612\)](#)

`__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 \(p. 589\)](#).

[name\(cls\)](#)

Inherited from [GlueTransform](#) [name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from [GlueTransform](#) [describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from [GlueTransform](#) [describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from [GlueTransform](#) [describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform](#) [describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from [GlueTransform](#) [describe \(p. 591\)](#).

Example for RenameField

This example simplifies the names of fields in DynamicFrames created by the Relationalize transform, and then drops the added index and id fields.

```
dyf_renameField_1 = RenameField.apply(dyf_flattened, "`customers.val.address`", "address")
dyf_renameField_2 = RenameField.apply( dyf_renameField_1, "`customers.val.id`",
    "cust_id" )

dyf_dropfields_rf = DropFields.apply( frame = dyf_renameField_2, paths = ["index", "id"] )

dyf_dropfields_rf.toDF().show()
+-----+-----+
| address|cust_id|
+-----+-----+
| 66 P Street, NY| 343|
| 708 Fed Ln, CA| 932|
| 807 Deccan Dr, CA| 102|
|108 Park Street, TX| 623|
| 763 Marsh Ln, TX| 231|
+-----+-----+
```

ResolveChoice Class

Resolves a choice type within a [DynamicFrame](#).

Methods

- [__call__ \(p. 613\)](#)
- [apply \(p. 614\)](#)
- [name \(p. 614\)](#)
- [describeArgs \(p. 614\)](#)

- [describeReturn \(p. 614\)](#)
- [describeTransform \(p. 614\)](#)
- [describeErrors \(p. 614\)](#)
- [describe \(p. 614\)](#)

`__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 values 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 \(p. 589\)](#).

[name\(cls\)](#)

Inherited from [GlueTransform name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from [GlueTransform describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from [GlueTransform describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from [GlueTransform describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from [GlueTransform describe \(p. 591\)](#).

Example for [ResolveChoice](#)

This example shows how to handle column type ambiguities by casting all values in the `cust_id` field to the data type `String`.

```
dyf_resolveChoice = dyf_input.resolveChoice(specs = [('cust_id', 'cast:String')])
```

SelectFields Class

Gets fields in a [DynamicFrame](#).

Methods

- [__call__ \(p. 615\)](#)
- [apply \(p. 615\)](#)
- [name \(p. 615\)](#)
- [describeArgs \(p. 615\)](#)
- [describeReturn \(p. 615\)](#)
- [describeTransform \(p. 615\)](#)
- [describeErrors \(p. 615\)](#)
- [describe \(p. 615\)](#)

`__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. 589).

`name(cls)`

Inherited from `GlueTransform name` (p. 589).

`describeArgs(cls)`

Inherited from `GlueTransform describeArgs` (p. 590).

`describeReturn(cls)`

Inherited from `GlueTransform describeReturn` (p. 590).

`describeTransform(cls)`

Inherited from `GlueTransform describeTransform` (p. 590).

`describeErrors(cls)`

Inherited from `GlueTransform describeErrors` (p. 590).

`describe(cls)`

Inherited from `GlueTransform describe` (p. 591).

Example for `SelectFields`

This example selects only the zip code column, but you can add more columns because the argument `paths` accepts a list.

```
dyf_selectFields = SelectFields.apply(frame = dyf_filter, paths=['zip'])
```

SelectFromCollection Class

Selects one `DynamicFrame` in a `DynamicFrameCollection`.

Methods

- [__call__ \(p. 616\)](#)
- [apply \(p. 616\)](#)
- [name \(p. 616\)](#)
- [describeArgs \(p. 616\)](#)
- [describeReturn \(p. 616\)](#)
- [describeTransform \(p. 616\)](#)
- [describeErrors \(p. 616\)](#)
- [describe \(p. 616\)](#)

[__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 \(p. 589\)](#).

[name\(cls\)](#)

Inherited from [GlueTransform name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from [GlueTransform describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from [GlueTransform describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from [GlueTransform describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from [GlueTransform describe \(p. 591\)](#).

[Example for SelectFromCollection](#)

The `SelectFromCollection` function retrieves a specific DynamicFrame from a collection of DynamicFrames.

This example uses the following DynamicFrame collection as input.

```
dyf_relationize.keys()  
  
dict_keys(['root', 'root_customers'])
```

The following command retrieves the first DynamicFrame, with the key `root`.

```
dyf_selectFromCollection = SelectFromCollection.apply(dyf_relationize, 'root')  
  
dyf_selectFromCollection.toDF().show()  
+-----+  
|customers| zip|  
+-----+  
| 1|75091|  
| 2|75023|  
| 3|90093|  
+-----+
```

This next command retrieves the second DynamicFrame from the collection, with the key `root_customers`.

```
dyf_selectFromCollection = SelectFromCollection.apply(dyf_relationize, 'root_customers')  
  
dyf_selectFromCollection.toDF().show()  
+-----+-----+-----+  
| id|index|customers.val.address|customers.val.id|  
+-----+-----+-----+  
| 2| 0| 66 P Street, NY| 343|  
| 3| 0| 708 Fed Ln, CA| 932|  
| 3| 1| 807 Deccan Dr, CA| 102|  
| 1| 0| 108 Park Street, TX| 623|  
| 1| 1| 763 Marsh Ln, TX| 231|  
+-----+-----+-----+
```

Spigot Class

Writes sample records to a specified destination during a transformation.

Methods

- [__call__ \(p. 617\)](#)
- [apply \(p. 618\)](#)
- [name \(p. 618\)](#)
- [describeArgs \(p. 618\)](#)
- [describeReturn \(p. 618\)](#)
- [describeTransform \(p. 618\)](#)
- [describeErrors \(p. 618\)](#)
- [describe \(p. 618\)](#)

[__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 \(p. 589\)](#)

[name\(cls\)](#)

Inherited from `GlueTransform` [name \(p. 589\)](#)

[describeArgs\(cls\)](#)

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#)

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#)

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#)

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#)

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 591\)](#)

[Example for Spigot](#)

```
dyf_result = Spigot.apply(dyf_input, '/home/glue/GlueLocalOutput/Spigot/', 'top10')
```

[SplitFields Class](#)

Splits a `DynamicFrame` into two new ones, by specified fields.

Methods

- [__call__ \(p. 619\)](#)
- [apply \(p. 619\)](#)
- [name \(p. 619\)](#)
- [describeArgs \(p. 619\)](#)
- [describeReturn \(p. 619\)](#)
- [describeTransform \(p. 619\)](#)

- [describeErrors \(p. 619\)](#)
- [describe \(p. 619\)](#)

`__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 \(p. 589\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 589\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 591\)](#).

Example for SplitFields

This example uses the following DynamicFrame as input, and splits it into two DynamicFrames.

```
dyf_dropNullfields.toDF().show()

+-----+-----+-----+
|warehouse_loc|data.strawberry|data.pineapple|data.mango|
+-----+-----+-----+
| TX_WAREHOUSE| 220| 560| 350|
| CA_WAREHOUSE| 34| 123| 42|
| CO_WAREHOUSE| 340| 180| 2|
+-----+-----+-----+
```

```
dyf_splitFields = SplitFields.apply(frame = dyf_dropNullfields, paths =
["`data.strawberry`",
 "`data.pineapple`"], name1 = "a", name2 = "b")
```

You can view the first DynamicFrame result using the following commands.

```
dyf_retrieve_a = SelectFromCollection.apply(dyf_splitFields, "a")

dyf_retrieve_a.toDF().show()
+-----+
|data.strawberry|data.pineapple|
+-----+
| 220| 560|
| 34| 123|
| 340| 180|
+-----+
```

You can view the second DynamicFrame result using the following commands.

```
dyf_retrieve_b = SelectFromCollection.apply(dyf_splitFields, "b")

dyf_retrieve_b.toDF().show()
+-----+
|warehouse_loc|data.mango|
+-----+
| TX_WAREHOUSE| 350|
| CA_WAREHOUSE| 42|
| CO_WAREHOUSE| 2|
+-----+
```

SplitRows Class

Splits a DynamicFrame in two by specified rows.

Methods

- [__call__ \(p. 621\)](#)
- [apply \(p. 621\)](#)
- [name \(p. 621\)](#)
- [describeArgs \(p. 621\)](#)
- [describeReturn \(p. 621\)](#)
- [describeTransform \(p. 621\)](#)

- [describeErrors \(p. 621\)](#)
- [describe \(p. 621\)](#)

`__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 \(p. 589\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 589\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

`describeReturn(cls)`

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

`describeTransform(cls)`

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

`describeErrors(cls)`

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

`describe(cls)`

Inherited from `GlueTransform` [describe \(p. 591\)](#).

Example for SplitRows

```
dyf_splitRows = SplitRows.apply(frame = dyf_dropNullfields, comparison_dict =  
    {"`data.pineapple`": {">": "100",  
    "<": "200"}}, name1 = 'pa_200_less', name2 = 'pa_200_more')
```

Unbox Class

Unboxes a string field in a `DynamicFrame`.

Methods

- [__call__ \(p. 622\)](#)
- [apply \(p. 622\)](#)
- [name \(p. 623\)](#)
- [describeArgs \(p. 623\)](#)
- [describeReturn \(p. 623\)](#)
- [describeTransform \(p. 623\)](#)
- [describeErrors \(p. 623\)](#)
- [describe \(p. 623\)](#)

`__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 \(p. 504\)](#) 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. 589)`.

[name\(cls\)](#)

Inherited from [GlueTransform name \(p. 589\)](#).

[describeArgs\(cls\)](#)

Inherited from [GlueTransform describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from [GlueTransform describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from [GlueTransform describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from [GlueTransform describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from [GlueTransform describe \(p. 591\)](#).

[Example for Unbox](#)

The following commands create the `dyf_warehouse` DynamicFrame that is used in this example.

```
warehouse_inventory_list = [ ['TX_WAREHOUSE', '{\ \"strawberry\":220,\ \"pineapple\":560,\ \"mango\":350,\ \"pears\":null}', ],\ ['CA_WAREHOUSE', '{\ \"strawberry\":34,\ \"pineapple\":123,\ \"mango\":42,\ \"pears\":null}'],\ ['CO_WAREHOUSE', '{\ \"strawberry\":340,\ \"pineapple\":180,\ \"mango\":2,\ \"pears\":null}' ] ]

warehouse_schema = StructType([StructField("warehouse_loc", StringType()), StructField("data", StringType())])

df_warehouse = spark.createDataFrame(warehouse_inventory_list, schema = warehouse_schema)

dyf_warehouse = DynamicFrame.fromDF(df_warehouse, glueContext, "dyf_warehouse")
```

The `Unbox` function in the following example converts the lists within the `String` datatype into a structure.

```
dyf_warehouse.printSchema()

root
|-- warehouse_location: string
|-- data: string

dyf_unbox = Unbox.apply(frame = dyf_warehouse, path = "data", format="json")
dyf_unbox.printSchema()

root
|-- warehouse_loc: string
|-- data: struct
|   |-- strawberry: int
|   |-- pineapple: int
|   |-- mango: int
|   |-- pears: null
```

```
dyf_unbox.toDF().show()

+-----+-----+
|warehouse_loc| data|
+-----+-----+
| TX_WAREHOUSE|[ 220, 560, 350,]|
| CA_WAREHOUSE|[ 34, 123, 42,]|
| CO_WAREHOUSE|[ 340, 180, 2,]|
+-----+-----+
```

UnnestFrame Class

Unnests a `DynamicFrame`, flattens nested objects to top-level elements, and generates joinkeys for array objects.

Methods

- [__call__ \(p. 624\)](#)
- [apply \(p. 624\)](#)
- [name \(p. 624\)](#)
- [describeArgs \(p. 624\)](#)
- [describeReturn \(p. 625\)](#)
- [describeTransform \(p. 625\)](#)
- [describeErrors \(p. 625\)](#)
- [describe \(p. 625\)](#)

`__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 \(p. 589\)](#).

`name(cls)`

Inherited from `GlueTransform` [name \(p. 589\)](#).

`describeArgs(cls)`

Inherited from `GlueTransform` [describeArgs \(p. 590\)](#).

[describeReturn\(cls\)](#)

Inherited from `GlueTransform` [describeReturn \(p. 590\)](#).

[describeTransform\(cls\)](#)

Inherited from `GlueTransform` [describeTransform \(p. 590\)](#).

[describeErrors\(cls\)](#)

Inherited from `GlueTransform` [describeErrors \(p. 590\)](#).

[describe\(cls\)](#)

Inherited from `GlueTransform` [describe \(p. 591\)](#).

Example for UnnestFrame

The following command shows the schema of the `dyf_unbox` DynamicFrame that is used in this example.

```
dyf_unbox.printSchema()

root
|-- warehouse_loc: string
|-- data: struct
| |-- strawberry: int
| |-- pineapple: int
| |-- mango: int
| |-- pears: null
```

The `Unnest` function in the following example flattens the nested structure data into a more relational table format.

```
dyf_unnest = UnnestFrame.apply(frame = dyf_unbox)

dyf_unnest.printSchema()

|-- warehouse_loc: string
|-- data.strawberry: int
|-- data.pineapple: int
|-- data.mango: int
|-- data.pears: null
```

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 \(p. 630\)](#)
 - [Testing a Scala ETL Program in a Zeppelin Notebook on a Development Endpoint \(p. 630\)](#)
 - [Testing a Scala ETL Program in a Scala REPL \(p. 631\)](#)

- [Scala Script Example - Streaming ETL \(p. 631\)](#)
- [APIs in the AWS Glue Scala Library \(p. 632\)](#)
 - [com.amazonaws.services.glue \(p. 632\)](#)
 - [com.amazonaws.services.glue.ml \(p. 633\)](#)
 - [com.amazonaws.services.glue.types \(p. 633\)](#)
 - [com.amazonaws.services.glue.util \(p. 633\)](#)
 - [AWS Glue Scala ChoiceOption APIs \(p. 633\)](#)
 - [ChoiceOption Trait \(p. 634\)](#)
 - [ChoiceOption Object \(p. 634\)](#)
 - [def apply \(p. 634\)](#)
 - [case class ChoiceOptionWithResolver \(p. 634\)](#)
 - [case class MatchCatalogSchemaChoiceOption \(p. 634\)](#)
 - [Abstract DataSink Class \(p. 634\)](#)
 - [def writeDynamicFrame \(p. 635\)](#)
 - [def pyWriteDynamicFrame \(p. 635\)](#)
 - [def setCatalogInfo \(p. 635\)](#)
 - [def supportsFormat \(p. 635\)](#)
 - [def setFormat \(p. 635\)](#)
 - [def withFormat \(p. 635\)](#)
 - [def setAccumulableSize \(p. 635\)](#)
 - [def getOutputErrorRecordsAccumulable \(p. 636\)](#)
 - [def errorsAsDynamicFrame \(p. 636\)](#)
 - [DataSink Object \(p. 636\)](#)
 - [def recordMetrics \(p. 636\)](#)
 - [AWS Glue Scala DataSource Trait \(p. 636\)](#)
 - [AWS Glue Scala DynamicFrame APIs \(p. 637\)](#)
 - [AWS Glue Scala DynamicFrame Class \(p. 638\)](#)
 - [val errorsCount \(p. 638\)](#)
 - [def applyMapping \(p. 639\)](#)
 - [def assertErrorThreshold \(p. 640\)](#)
 - [def count \(p. 640\)](#)
 - [def dropField \(p. 640\)](#)
 - [def dropFields \(p. 640\)](#)
 - [def dropNulls \(p. 640\)](#)
 - [def errorsAsDynamicFrame \(p. 641\)](#)
 - [def filter \(p. 641\)](#)
 - [def getName \(p. 641\)](#)
 - [def getNumPartitions \(p. 641\)](#)
 - [def getSchemaComputed \(p. 641\)](#)
 - [def isSchemaComputed \(p. 641\)](#)
 - [def javaToPython \(p. 641\)](#)
 - [def join \(p. 642\)](#)
 - [def map \(p. 642\)](#)
 - [def mergeDynamicFrames \(p. 642\)](#)
 - [def printSchema \(p. 643\)](#)
 - [def recomputeSchema \(p. 643\)](#)

- [def relationalize \(p. 643\)](#)
- [def renameField \(p. 644\)](#)
- [def repartition \(p. 645\)](#)
- [def resolveChoice \(p. 645\)](#)
- [def schema \(p. 646\)](#)
- [def selectField \(p. 646\)](#)
- [def selectFields \(p. 646\)](#)
- [def show \(p. 647\)](#)
- [def spigot \(p. 647\)](#)
- [def splitFields \(p. 647\)](#)
- [def splitRows \(p. 648\)](#)
- [def stageErrorsCount \(p. 648\)](#)
- [def toDF \(p. 649\)](#)
- [def unbox \(p. 649\)](#)
- [def unnest \(p. 650\)](#)
- [def unnestDDBJson \(p. 650\)](#)
- [def withFrameSchema \(p. 652\)](#)
- [def withName \(p. 652\)](#)
- [def withTransformationContext \(p. 652\)](#)
- [The DynamicFrame Object \(p. 652\)](#)
 - [def apply \(p. 652\)](#)
 - [def emptyDynamicFrame \(p. 652\)](#)
 - [def fromPythonRDD \(p. 652\)](#)
 - [def ignoreErrors \(p. 653\)](#)
 - [def inlineErrors \(p. 653\)](#)
 - [def newFrameWithErrors \(p. 653\)](#)
- [AWS Glue Scala DynamicRecord Class \(p. 653\)](#)
 - [def addField \(p. 654\)](#)
 - [def dropField \(p. 654\)](#)
 - [def setError \(p. 654\)](#)
 - [def isError \(p. 654\)](#)
 - [def getError \(p. 654\)](#)
 - [def clearError \(p. 654\)](#)
 - [def write \(p. 655\)](#)
 - [def readFields \(p. 655\)](#)
 - [def clone \(p. 655\)](#)
 - [def schema \(p. 655\)](#)
 - [def getRoot \(p. 655\)](#)
 - [def toJson \(p. 655\)](#)
 - [def getFieldNode \(p. 655\)](#)
 - [def getField \(p. 655\)](#)
 - [def hashCode \(p. 656\)](#)
 - [def equals \(p. 656\) 627](#)
 - [DynamicRecord Object \(p. 656\)](#)
 - [def apply \(p. 656\)](#)

- [RecordTraverser Trait \(p. 656\)](#)
 - [AWS Glue Scala GlueContext APIs \(p. 656\)](#)
 - [def addIngestionTimeColumns \(p. 657\)](#)
 - [def createDataFrameFromOptions \(p. 657\)](#)
 - [foreachBatch \(p. 658\)](#)
 - [def getCatalogSink \(p. 659\)](#)
 - [def getCatalogSource \(p. 659\)](#)
 - [def getJDBCSink \(p. 660\)](#)
 - [def getSink \(p. 661\)](#)
 - [def getSinkWithFormat \(p. 661\)](#)
 - [def getSource \(p. 662\)](#)
 - [def getSourceWithFormat \(p. 663\)](#)
 - [def getSparkSession \(p. 664\)](#)
 - [def startTransaction \(p. 664\)](#)
 - [def commitTransaction \(p. 664\)](#)
 - [def cancelTransaction \(p. 664\)](#)
 - [def this \(p. 665\)](#)
 - [def this \(p. 665\)](#)
 - [def this \(p. 665\)](#)
 - [MappingSpec \(p. 665\)](#)
 - [MappingSpec Case Class \(p. 665\)](#)
 - [MappingSpec Object \(p. 666\)](#)
 - [val orderingByTarget \(p. 666\)](#)
 - [def apply \(p. 666\)](#)
 - [def apply \(p. 666\)](#)
 - [def apply \(p. 667\)](#)
 - [AWS Glue Scala ResolveSpec APIs \(p. 667\)](#)
 - [ResolveSpec Object \(p. 667\)](#)
 - [def \(p. 667\)](#)
 - [def \(p. 667\)](#)
 - [ResolveSpec Case Class \(p. 668\)](#)
 - [ResolveSpec def Methods \(p. 668\)](#)
 - [AWS Glue Scala ArrayNode APIs \(p. 668\)](#)
 - [ArrayNode Case Class \(p. 668\)](#)
 - [ArrayNode def Methods \(p. 668\)](#)
 - [AWS Glue Scala BinaryNode APIs \(p. 669\)](#)
 - [BinaryNode Case Class \(p. 669\)](#)
 - [BinaryNode val Fields \(p. 669\)](#)
 - [BinaryNode def Methods \(p. 669\)](#)
 - [AWS Glue Scala BooleanNode APIs \(p. 669\)](#)
 - [BooleanNode Case Class \(p. 669\)](#)
 - [BooleanNode val Fields \(p. 669\)](#)
 - [BooleanNode def Methods \(p. 670\)](#)
 - [AWS Glue Scala ByteNode APIs \(p. 670\)](#)
 - [ByteNode Case Class \(p. 670\)](#)
-

- [ByteNode val Fields \(p. 670\)](#)
 - [ByteNode def Methods \(p. 670\)](#)
 - [AWS Glue Scala DateNode APIs \(p. 670\)](#)
 - [DateNode Case Class \(p. 670\)](#)
 - [DateNode val Fields \(p. 670\)](#)
 - [DateNode def Methods \(p. 670\)](#)
 - [AWS Glue Scala DecimalNode APIs \(p. 670\)](#)
 - [DecimalNode Case Class \(p. 670\)](#)
 - [DecimalNode val Fields \(p. 671\)](#)
 - [DecimalNode def Methods \(p. 671\)](#)
 - [AWS Glue Scala DoubleNode APIs \(p. 671\)](#)
 - [DoubleNode Case Class \(p. 671\)](#)
 - [DoubleNode val Fields \(p. 671\)](#)
 - [DoubleNode def Methods \(p. 671\)](#)
 - [AWS Glue Scala DynamicNode APIs \(p. 671\)](#)
 - [DynamicNode Class \(p. 671\)](#)
 - [DynamicNode def Methods \(p. 671\)](#)
 - [DynamicNode Object \(p. 672\)](#)
 - [DynamicNode def Methods \(p. 672\)](#)
 - [AWS Glue Scala FloatNode APIs \(p. 672\)](#)
 - [FloatNode Case Class \(p. 672\)](#)
 - [FloatNode val Fields \(p. 672\)](#)
 - [FloatNode def Methods \(p. 672\)](#)
 - [FillMissingValues Class \(p. 672\)](#)
 - [def apply \(p. 673\)](#)
 - [FindMatches Class \(p. 673\)](#)
 - [def apply \(p. 673\)](#)
 - [FindIncrementalMatches Class \(p. 674\)](#)
 - [def apply \(p. 674\)](#)
 - [AWS Glue Scala IntegerNode APIs \(p. 675\)](#)
 - [IntegerNode Case Class \(p. 675\)](#)
 - [IntegerNode val Fields \(p. 675\)](#)
 - [IntegerNode def Methods \(p. 675\)](#)
 - [AWS Glue Scala LongNode APIs \(p. 675\)](#)
 - [LongNode Case Class \(p. 675\)](#)
 - [LongNode val Fields \(p. 675\)](#)
 - [LongNode def Methods \(p. 675\)](#)
 - [AWS Glue Scala MapLikeNode APIs \(p. 675\)](#)
 - [MapLikeNode Class \(p. 675\)](#)
 - [MapLikeNode def Methods \(p. 676\)](#)
 - [AWS Glue Scala MapNode APIs \(p. 676\)](#)
 - [MapNode Case Class \(p. 676\)](#)
 - [MapNode def Methods \(p. 676\)](#)
 - [AWS Glue Scala NullNode APIs \(p. 677\)](#)
 - [NullNode Class \(p. 677\)](#)
-

- [NullNode Case Object \(p. 677\)](#)
- [AWS Glue Scala ObjectNode APIs \(p. 677\)](#)
 - [ObjectNode Object \(p. 677\)](#)
 - [ObjectNode def Methods \(p. 677\)](#)
 - [ObjectNode Case Class \(p. 678\)](#)
 - [ObjectNode def Methods \(p. 678\)](#)
- [AWS Glue Scala ScalarNode APIs \(p. 678\)](#)
 - [ScalarNode Class \(p. 678\)](#)
 - [ScalarNode def Methods \(p. 678\)](#)
 - [ScalarNode Object \(p. 679\)](#)
 - [ScalarNode def Methods \(p. 679\)](#)
- [AWS Glue Scala ShortNode APIs \(p. 679\)](#)
 - [ShortNode Case Class \(p. 679\)](#)
 - [ShortNode val Fields \(p. 679\)](#)
 - [ShortNode def Methods \(p. 679\)](#)
- [AWS Glue Scala StringNode APIs \(p. 679\)](#)
 - [StringNode Case Class \(p. 679\)](#)
 - [StringNode val Fields \(p. 680\)](#)
 - [StringNode def Methods \(p. 680\)](#)
- [AWS Glue Scala TimestampNode APIs \(p. 680\)](#)
 - [TimestampNode Case Class \(p. 680\)](#)
 - [TimestampNode val Fields \(p. 680\)](#)
 - [TimestampNode def Methods \(p. 680\)](#)
- [AWS Glue Scala GlueArgParser APIs \(p. 680\)](#)
 - [GlueArgParser Object \(p. 680\)](#)
 - [GlueArgParser def Methods \(p. 680\)](#)
- [AWS Glue Scala Job APIs \(p. 681\)](#)
 - [Job Object \(p. 681\)](#)
 - [Job def Methods \(p. 681\)](#)

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 \(p. 197\)](#). 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 \(p. 252\)](#).

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 \(p. 264\)](#).

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 \(p. 274\)](#), 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`.

Scala Script Example - Streaming ETL

Example

The following example script connects to Amazon Kinesis Data Streams, uses a schema from the Data Catalog to parse a data stream, joins the stream to a static dataset on Amazon S3, and outputs the joined results to Amazon S3 in parquet format.

```
// This script connects to an Amazon Kinesis stream, uses a schema from the data catalog to
// parse the stream,
// joins the stream to a static dataset on Amazon S3, and outputs the joined results to
// Amazon S3 in parquet format.
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import java.util.Calendar
import org.apache.spark.SparkContext
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.from_json
import org.apache.spark.sql.streaming.Trigger
import scala.collection.JavaConverters._

object streamJoiner {
    def main(sysArgs: Array[String]) {
        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)
        val sparkSession: SparkSession = glueContext.getSparkSession
        import sparkSession.implicits._
        // @params: [JOB_NAME]
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val staticData = sparkSession.read          // read() returns type DataFrameReader
            .format("csv")
            .option("header", "true")
            .load("s3://awsexamplebucket-streaming-demo2/inputs/productsStatic.csv") // load()
        returns a DataFrame
    }
}
```

```

    val datasource0 = sparkSession.readStream // readstream() returns type
DataStreamReader
    .format("kinesis")
    .option("streamName", "stream-join-demo")
    .option("endpointUrl", "https://kinesis.us-east-1.amazonaws.com")
    .option("startingPosition", "TRIM_HORIZON")
    .load // load() returns a DataFrame

    val selectfields1 = datasource0.select(from_json($"data".cast("string"),
glueContext.getCatalogSchemaAsSparkSchema("stream-demos", "stream-join-demo2")) as
"data").select("data.*")

    val datasink2 = selectfields1.writeStream.foreachBatch { (dataFrame: Dataset[Row],
batchId: Long) => { //foreachBatch() returns type DataStreamWriter
        val joined = dataFrame.join(staticData, "product_id")
        val year: Int = Calendar.getInstance().get(Calendar.YEAR)
        val month :Int = Calendar.getInstance().get(Calendar.MONTH) + 1
        val day: Int = Calendar.getInstance().get(Calendar.DATE)
        val hour: Int = Calendar.getInstance().get(Calendar.HOUR_OF_DAY)

        if (dataFrame.count() > 0) {
            joined.write // joined.write returns type DataFrameWriter
                .mode(SaveMode.Append)
                .format("parquet")
                .option("quote", " ")
                .save("s3://awsexamplebucket-streaming-demo2/output/" + "/year=" +
"%04d".format(year) + "/month=" + "%02d".format(month) + "/day=" + "%02d".format(day) + "/
hour=" + "%02d".format(hour) + "/")
        }
    }
} // end foreachBatch()
.trigger(Trigger.ProcessingTime("100 seconds"))
.option("checkpointLocation", "s3://awsexamplebucket-streaming-demo2/checkpoint/")
.start().awaitTermination() // start() returns type StreamingQuery
Job.commit()
}
}

```

APIs in the AWS Glue Scala 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 \(p. 633\)](#)
- [DataSink \(p. 634\)](#)
- [DataSource trait \(p. 636\)](#)
- [DynamicFrame \(p. 637\)](#)
- [DynamicRecord \(p. 653\)](#)
- [GlueContext \(p. 656\)](#)
- [MappingSpec \(p. 665\)](#)
- [ResolveSpec \(p. 667\)](#)

com.amazonaws.services.glue.ml

The **com.amazonaws.services.glue.ml** package in the AWS Glue Scala library contains the following APIs:

- [FillMissingValues \(p. 672\)](#)
- [FindIncrementalMatches \(p. 674\)](#)
- [FindMatches \(p. 673\)](#)

com.amazonaws.services.glue.types

The **com.amazonaws.services.glue.types** package in the AWS Glue Scala library contains the following APIs:

- [ArrayNode \(p. 668\)](#)
- [BinaryNode \(p. 669\)](#)
- [BooleanNode \(p. 669\)](#)
- [ByteNode \(p. 670\)](#)
- [DateNode \(p. 670\)](#)
- [DecimalNode \(p. 670\)](#)
- [DoubleNode \(p. 671\)](#)
- [DynamicNode \(p. 671\)](#)
- [FloatNode \(p. 672\)](#)
- [IntegerNode \(p. 675\)](#)
- [LongNode \(p. 675\)](#)
- [MapLikeNode \(p. 675\)](#)
- [MapNode \(p. 676\)](#)
- [NullNode \(p. 677\)](#)
- [ObjectNode \(p. 677\)](#)
- [ScalarNode \(p. 678\)](#)
- [ShortNode \(p. 679\)](#)
- [StringNode \(p. 679\)](#)
- [TimestampNode \(p. 680\)](#)

com.amazonaws.services.glue.util

The **com.amazonaws.services.glue.util** package in the AWS Glue Scala library contains the following APIs:

- [GlueArgParser \(p. 680\)](#)
- [Job \(p. 681\)](#)

AWS Glue Scala ChoiceOption APIs

Topics

- [ChoiceOption Trait \(p. 634\)](#)
- [ChoiceOption Object \(p. 634\)](#)
- [case class ChoiceOptionWithResolver \(p. 634\)](#)

- [case class MatchCatalogSchemaChoiceOption \(p. 634\)](#)

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 \(p. 635\)](#)
- [def pyWriteDynamicFrame \(p. 635\)](#)
- [def setCatalogInfo \(p. 635\)](#)
- [def supportsFormat \(p. 635\)](#)
- [def setFormat \(p. 635\)](#)
- [def withFormat \(p. 635\)](#)
- [def setAccumulableSize \(p. 635\)](#)
- [def getOutputErrorRecordsAccumulable \(p. 636\)](#)

- [def errorsAsDynamicFrame \(p. 636\)](#)
- [DataSink Object \(p. 636\)](#)

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

    /** @param num: the number of records for sampling.
     *  @param options: optional parameters to control sampling behavior. Current available
     *  parameter for Amazon S3 sources in options:
     *  * 1. maxSamplePartitions: the maximum number of partitions the sampling will read.
     *  * 2. maxSampleFilesPerPartition: the maximum number of files the sampling will read in
     *  one partition.
     */
    def getSampleDynamicFrame(num:Int, options: JsonOptions = JsonOptions.empty): 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 \(p. 638\)](#)
 - [val errorsCount \(p. 638\)](#)
 - [def applyMapping \(p. 639\)](#)
 - [def assertErrorThreshold \(p. 640\)](#)
 - [def count \(p. 640\)](#)
 - [def dropField \(p. 640\)](#)
 - [def dropFields \(p. 640\)](#)
 - [def dropNulls \(p. 640\)](#)
 - [def errorsAsDynamicFrame \(p. 641\)](#)
 - [def filter \(p. 641\)](#)
 - [def getName \(p. 641\)](#)
 - [def getNumPartitions \(p. 641\)](#)
 - [def getSchemaComputed \(p. 641\)](#)
 - [def isSchemaComputed \(p. 641\)](#)
 - [def javaToPython \(p. 641\)](#)
 - [def join \(p. 642\)](#)
 - [def map \(p. 642\)](#)
 - [def mergeDynamicFrames \(p. 642\)](#)
 - [def printSchema \(p. 643\)](#)
 - [def recomputeSchema \(p. 643\)](#)
 - [def relationalize \(p. 643\)](#)
 - [def renameField \(p. 644\)](#)
 - [def repartition \(p. 645\)](#)
 - [def resolveChoice \(p. 645\)](#)
 - [def schema \(p. 646\)](#)
 - [def selectField \(p. 646\)](#)
 - [def selectFields \(p. 646\)](#)
 - [def show \(p. 647\)](#)
 - [def spigot \(p. 647\)](#)
 - [def splitFields \(p. 647\)](#)
 - [def splitRows \(p. 648\)](#)
 - [def stageErrorsCount \(p. 648\)](#)
 - [def toDF \(p. 649\)](#)
 - [def unbox \(p. 649\)](#)
 - [def unnest \(p. 650\)](#)
 - [def unnestDDBJson \(p. 650\)](#)
 - [def withFrameSchema \(p. 652\)](#)

- [def withName \(p. 652\)](#)
- [def withTransformationContext \(p. 652\)](#)
- [The DynamicFrame Object \(p. 652\)](#)
 - [def apply \(p. 652\)](#)
 - [def emptyDynamicFrame \(p. 652\)](#)
 - [def fromPythonRDD \(p. 652\)](#)
 - [def ignoreErrors \(p. 653\)](#)
 - [def inlineErrors \(p. 653\)](#)
 - [def newFrameWithErrors \(p. 653\)](#)

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 \(p. 653\)](#) 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 \(p. 665\)](#) 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 \(p. 650\)](#) 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"]}
```

Run the following code.

```
 {{{
   df.relationalize("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 `relationalize` 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 \(p. 650\)](#) 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 \(p. 639\)](#) 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.

Example

This example takes a DynamicFrame created from the persons table in the legislators database in the AWS Glue Data Catalog and splits the DynamicFrame into two, with the specified fields going into the first DynamicFrame and the remaining fields going into a second DynamicFrame. The example then chooses the first DynamicFrame from the result.

```
val InputFrame = glueContext.getCatalogSource(database="legislators", tableName="persons", transformationContext="InputFrame").getDynamicFrame()

val SplitField_collection = InputFrame.splitFields(paths=Seq("family_name", "name", "links.note", "links.url", "gender", "image", "identifiers.scheme", "identifiers.identifier", "other_names.lang", "other_names.note", "other_names.name"), transformationContext="SplitField_collection")

val ResultFrame = SplitField_collection(0)
```

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(">=", "="))
}}}
```

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 \(p. 645\)](#).

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 unnestDDBJson

```

unnestDDBJson(transformationContext : String = "",
                callSite : CallSite = CallSite("Not Provided"),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0): DynamicFrame

```

Unnests nested columns in a `DynamicFrame` that are specifically in the DynamoDB JSON structure, and returns a new unnested `DynamicFrame`. Columns that are of an array of struct types will not be unnested. Note that this is a specific type of unnesting transform that behaves differently from the

regular unnest transform and requires the data to already be in the DynamoDB JSON structure. For more information, see [DynamoDB JSON](#).

For example, the schema of a reading an export with the DynamoDB JSON structure might look like the following:

```
root
|-- Item: struct
|   |-- ColA: struct
|   |   |-- S: string
|   |-- ColB: struct
|   |   |-- S: string
|   |-- ColC: struct
|   |   |-- N: string
|   |-- ColD: struct
|   |   |-- L: array
|   |       |-- element: null
```

The `unnestDDBJson()` transform would convert this to:

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|   |-- element: null
```

The following code example shows how to use the AWS Glue DynamoDB export connector, invoke a DynamoDB JSON unnest, and print the number of partitions:

```
import com.amazonaws.services.glue.GlueContext
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.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getSourceWithFormat(
            connectionType = "dynamodb",
            options = JsonOptions(Map(
                "dynamodb.export" -> "ddb",
                "dynamodb.tableArn" -> "<test_source>",
                "dynamodb.s3.bucket" -> "<bucket name>",
                "dynamodb.s3.prefix" -> "<bucket prefix>",
                "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
            )))
        .getDynamicFrame()

        val unnested = dynamicFrame.unnestDDBJson()
        print(unnested.getNumPartitions())

        Job.commit()
    }
}
```

```
}
```

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 \(p. 654\)](#)
- [def dropField \(p. 654\)](#)
- [def setError \(p. 654\)](#)
- [def isError \(p. 654\)](#)
- [def getError \(p. 654\)](#)
- [def clearError \(p. 654\)](#)
- [def write \(p. 655\)](#)
- [def readFields \(p. 655\)](#)
- [def clone \(p. 655\)](#)
- [def schema \(p. 655\)](#)
- [def getRoot \(p. 655\)](#)
- [def toJson \(p. 655\)](#)
- [def getFieldNode \(p. 655\)](#)
- [def getField \(p. 655\)](#)
- [def hashCode \(p. 656\)](#)
- [def equals \(p. 656\)](#)
- [DynamicRecord Object \(p. 656\)](#)
- [RecordTraverser Trait \(p. 656\)](#)

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 \(p. 671\)](#) to the specified path.

- `path` — The path for the field to be added.
- `dynamicNode` — The [DynamicNode \(p. 671\)](#) to be added at the specified path.

def dropField

```
def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

Drops a [DynamicNode \(p. 671\)](#) 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[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.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. 671))` 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 \(p. 653\)](#).

- **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 \(p. 637\)](#) 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 \(p. 636\)](#) and [DataSink \(p. 634\)](#) 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 addIngestionTimeColumns

```
def addIngestionTimeColumns(  
    df : DataFrame,  
    timeGranularity : String = "") : DataFrame
```

Appends ingestion time columns like `ingest_year`, `ingest_month`, `ingest_day`, `ingest_hour`, `ingest_minute` to the input `DataFrame`. This function is automatically generated in the script generated by the AWS Glue when you specify a Data Catalog table with Amazon S3 as the target. This function automatically updates the partition with ingestion time columns on the output table. This allows the output data to be automatically partitioned on ingestion time without requiring explicit ingestion time columns in the input data.

- `dataFrame` – The `DataFrame` to append the ingestion time columns to.
- `timeGranularity` – The granularity of the time columns. Valid values are "day", "hour" and "minute". For example, if "hour" is passed in to the function, the original `DataFrame` will have "`ingest_year`", "`ingest_month`", "`ingest_day`", and "`ingest_hour`" time columns appended.

Returns the data frame after appending the time granularity columns.

Example:

```
glueContext.addIngestionTimeColumns(dataFrame, "hour")
```

def createDataFrameFromOptions

```
def createDataFrameFromOptions( connectionType : String,  
    connectionOptions : JsonOptions,  
    transformationContext : String = "",  
    format : String = null,  
    formatOptions : JsonOptions = JsonOptions.empty  
) : DataSource
```

Returns a `DataFrame` created with the specified connection and format. Use this function only with AWS Glue streaming sources.

- `connectionType` – The streaming connection type. Valid values include `kinesis` and `kafka`.
- `connectionOptions` – Connection options, which are different for Kinesis and Kafka. You can find the list of all connection options for each streaming data source at [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#). Note the following differences in streaming connection options:
 - Kinesis streaming sources require `streamARN`, `startingPosition`, `inferSchema`, and `classification`.

- Kafka streaming sources require `connectionName`, `topicName`, `startingOffsets`, `inferSchema`, and `classification`.
- `transformationContext` – The transformation context to use (optional).
- `format` – A format specification (optional). This is used for an Amazon S3 or an AWS Glue connection that supports multiple formats. For information about the supported formats, see [Format Options for ETL Inputs and Outputs in AWS Glue \(p. 504\)](#)
- `formatOptions` – Format options for the specified format. For information about the supported format options, see [Format Options \(p. 504\)](#).

Example for Amazon Kinesis streaming source:

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
connectionType = "kinesis",
connectionOptions = JsonOptions("""{"streamName": "example_stream", "startingPosition": "TRIM_HORIZON", "inferSchema": "true", "classification": "json"}""")
```

Example for Kafka streaming source:

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
connectionType = "kafka",
connectionOptions = JsonOptions("""{"connectionName": "example_connection", "topicName": "example_topic", "startingPosition": "earliest", "inferSchema": "false", "classification": "json", "schema": "`column1` STRING, `column2` STRING"}""")
```

foreachBatch

foreachBatch(frame, batch_function, options)

Applies the `batch_function` passed in to every micro batch that is read from the Streaming source.

- `frame` – The DataFrame containing the current micro batch.
- `batch_function` – A function that will be applied for every micro batch.
- `options` – A collection of key-value pairs that holds information about how to process micro batches. The following options are required:
 - `windowSize` – The amount of time to spend processing each batch.
 - `checkpointLocation` – The location where checkpoints are stored for the streaming ETL job.
 - `batchMaxRetries` – The maximum number of times to retry the batch if it fails. The default value is 3. This option is only configurable for Glue version 2.0 and above.

Example:

```
glueContext.foreachBatch(data_frame_datasource0, (dataFrame: Dataset[Row], batchId: Long)
=>
{
  if (dataFrame.count() > 0)
  {
    val datasource0 = DynamicFrame(glueContext.addIngestionTimeColumns(dataFrame,
"hour"), glueContext)
    // @type: DataSink
    // @args: [database = "tempdb", table_name = "fromoptionsoutput",
    stream_batch_time = "100 seconds",
    //         stream_checkpoint_location = "s3://from-options-testing-eu-central-1/
    fromOptionsOutput/checkpoint/",
```

```
//      transformation_ctx = "datasink1"]
// @return: datasink1
// @inputs: [frame = datasource0]
val options_datasink1 = JsonOptions(
    Map("partitionKeys" -> Seq("ingest_year", "ingest_month", "ingest_day",
"ingest_hour"),
    "enableUpdateCatalog" -> true))
val datasink1 = glueContext.getCatalogSink(
    database = "tempdb",
    tableName = "fromoptionsoutput",
    redshiftTmpDir = "",
    transformationContext = "datasink1",
    additionalOptions = options_datasink1).writeDynamicFrame(datasource0)
}
}, JsonOptions("""{"windowSize" : "100 seconds",
"checkpointLocation" : "s3://from-options-testing-eu-central-1/fromOptionsOutput/
checkpoint/"}}"))

```

def getCatalogSink

```
def getCatalogSink( database : String,
    tableName : String,
    redshiftTmpDir : String = "",
    transformationContext : String = ""
    additionalOptions: JsonOptions = JsonOptions.empty,
    catalogId: String = null
) : DataSink
```

Creates a [DataSink \(p. 634\)](#) 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 \(p. 636\)](#) 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 \(p. 509\)](#).
- `additionalOptions` — A collection of optional name-value pairs. The possible options include those listed in [Connection Types and Options for ETL in AWS Glue \(p. 475\)](#) except for `endpointUrl`, `streamName`, `bootstrap.servers`, `security.protocol`, `topicName`, `classification`, and `delimiter`. Another supported option is `catalogPartitionPredicate`:

`catalogPartitionPredicate` — You can pass a catalog expression to filter based on the index columns. This pushes down the filtering to the server side. For more information, see [AWS Glue Partition Indexes](#). Note that `push_down_predicate` and `catalogPartitionPredicate` use different syntaxes. The former one uses Spark SQL standard syntax and the later one uses JDBC parser.
- `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`.

Example for streaming source

```
val data_frame_datasource0 = glueContext.getCatalogSource(
    database = "tempdb",
    tableName = "test-stream-input",
    redshiftTmpDir = "",
    transformationContext = "datasource0",
    additionalOptions = JsonOptions("""{
        "startingPosition": "TRIM_HORIZON", "inferSchema": "false"}""")  
).getDataFrame()
```

def getJDBCSink

```
def getJDBCSink( catalogConnection : String,
                 options : JsonOptions,
                 redshiftTmpDir : String = "",
                 transformationContext : String = "",
                 catalogId: String = null
               ) : DataSink
```

Creates a [DataSink \(p. 634\)](#) 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.

Example code:

```
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,
            connectionOptions : JsonOptions,
            transformationContext : String = ""
          ) : DataSink
```

Creates a [DataSink \(p. 634\)](#) that writes data to a destination like Amazon Simple Storage Service (Amazon S3), JDBC, or the AWS Glue Data Catalog.

- *connectionType* — The type of the connection. See [the section called “Connection Parameters” \(p. 475\)](#).
- *connectionOptions* — A string of JSON name-value pairs that provide additional information to establish the connection with the data sink. See [the section called “Connection Parameters” \(p. 475\)](#).
- *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 \(p. 634\)](#) 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. See [the section called “Connection Parameters” \(p. 475\)](#).
- *options* — A string of JSON name-value pairs that provide additional information to establish a connection with the data sink. See [the section called “Connection Parameters” \(p. 475\)](#).

- `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 \(p. 504\)](#).

Returns the `DataSink`.

def getSource

```
def getSource( connectionType : String,
              connectionOptions : JsonOptions,
              transformationContext : String = ""
              pushDownPredicate
            ) : DataSource
```

Creates a [DataSource trait \(p. 636\)](#) that reads data from a source like Amazon S3, JDBC, or the AWS Glue Data Catalog. Also supports Kafka and Kinesis streaming data sources.

- `connectionType` — The type of the data source. See [the section called “Connection Parameters” \(p. 475\)](#).
- `connectionOptions` — A string of JSON name-value pairs that provide additional information for establishing a connection with the data source. For more information, see [the section called “Connection Parameters” \(p. 475\)](#).

A Kinesis streaming source requires the following connection options: `streamARN`, `startingPosition`, `inferSchema`, and `classification`.

A Kafka streaming source requires the following connection options: `connectionName`, `topicName`, `startingOffsets`, `inferSchema`, and `classification`.

- `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`.

Example for Amazon Kinesis streaming source:

```
val kinesisOptions = jsonOptions()
data_frame_datasource0 = glueContext.getSource("kinesis", kinesisOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
  new JsonOptions(
    s"""{"streamARN": "arn:aws:kinesis:eu-central-1:123456789012:stream/
fromOptionsStream",
      |"startingPosition": "TRIM_HORIZON",
      |"inferSchema": "true",
      |"classification": "json"}""".stripMargin)
}
```

Example for Kafka streaming source:

```
val kafkaOptions = jsonOptions()
val data_frame_datasource0 = glueContext.getSource("kafka", kafkaOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
```

```

new JsonOptions(
    s"""{"connectionName": "ConfluentKafka",
       | "topicName": "kafka-auth-topic",
       | "startingOffsets": "earliest",
       | "inferSchema": "true",
       | "classification": "json"}""".stripMargin)
}

```

def getSourceWithFormat

```

def getSourceWithFormat( connectionType : String,
                        options : JsonOptions,
                        transformationContext : String = "",
                        format : String = null,
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSource

```

Creates a [DataSource trait \(p. 636\)](#) 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. See [the section called “Connection Parameters” \(p. 475\)](#).
- **options** – A string of JSON name-value pairs that provide additional information for establishing a connection with the data source. See [the section called “Connection Parameters” \(p. 475\)](#).
- **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 \(p. 504\)](#).

Returns the `DataSource`.

Examples

Create a `DynamicFrame` from a data source that is a comma-separated values (CSV) file on Amazon S3:

```

val datasource0 = glueContext.getSourceWithFormat(
  connectionType="s3",
  options =JsonOptions(s"""{"paths": [ "s3://csv/nycflights.csv"]}"""),
  transformationContext = "datasource0",
  format = "csv",
  formatOptions=JsonOptions(s"""{"withHeader":true,"separator": ","}""")
).getDynamicFrame()

```

Create a `DynamicFrame` from a data source that is a PostgreSQL using a JDBC connection:

```

val datasource0 = glueContext.getSourceWithFormat(
  connectionType="postgresql",
  options =JsonOptions(s"""{
    "url": "jdbc:postgresql://databasePostgres-1.rds.amazonaws.com:5432/testdb",
    "dbtable": "public.company",
    "redshiftTmpDir": "",
    "user": "username",
    "password": "password123"
}"""),
  transformationContext = "datasource0").getDynamicFrame()

```

Create a DynamicFrame from a data source that is a MySQL using a JDBC connection:

```
val datasource0 = glueContext.getSourceWithFormat(  
    connectionType="mysql",  
    options =JsonOptions(s"""  
        "url":"jdbc:mysql://databaseMysql-1.rds.amazonaws.com:3306/testdb",  
        "dbtable": "athenatest_nycflights13_csv",  
        "redshiftTmpDir": "",  
        "user": "username",  
        "password": "password123"  
    """),  
    transformationContext = "datasource0").getDynamicFrame()
```

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 startTransaction

```
def startTransaction(readOnly: Boolean):String
```

Start a new transaction. Internally calls the Lake Formation [startTransaction](#) API.

- `readOnly` – (Boolean) Indicates whether this transaction should be read only or read and write. Writes made using a read-only transaction ID will be rejected. Read-only transactions do not need to be committed.

Returns the transaction ID.

def commitTransaction

```
def commitTransaction(transactionId: String, waitForCommit: Boolean): Boolean
```

Attempts to commit the specified transaction. `commitTransaction` may return before the transaction has finished committing. Internally calls the Lake Formation [commitTransaction](#) API.

- `transactionId` – (String) The transaction to commit.
- `waitForCommit` – (Boolean) Determines whether the `commitTransaction` returns immediately. The default value is true. If false, `commitTransaction` polls and waits until the transaction is committed. The amount of wait time is restricted to 1 minute using exponential backoff with a maximum of 6 retry attempts.

Returns a Boolean to indicate whether the commit is done or not.

def cancelTransaction

```
def cancelTransaction(transactionId: String): Unit
```

Attempts to cancel the specified transaction. Internally calls the Lake Formation [CancelTransaction](#) API.

- **transactionId** – (String) The transaction to cancel.

Returns a `TransactionCommittedException` exception if the transaction was previously committed.

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: DataTyp
                        ) 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 \(p. 667\)](#)
- [ResolveSpec Case Class \(p. 668\)](#)

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 \(p. 671\)](#)
- [DynamicNode Object \(p. 672\)](#)

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 )
```

FillMissingValues Class

Package: com.amazonaws.services.glue.ml

```
object FillMissingValues
```

def apply

```
def apply(frame: DynamicFrame,  
         missingValuesColumn: String,  
         outputColumn: String = "",  
         transformationContext: String = "",  
         callSite: CallSite = CallSite("Not provided", ""),  
         stageThreshold: Long = 0,  
         totalThreshold: Long = 0): DynamicFrame
```

Fills a dynamic frame's missing values in a specified column and returns a new frame with estimates in a new column. For rows without missing values, the specified column's value is duplicated to the new column.

- **frame** — The DynamicFrame in which to fill missing values. Required.
- **missingValuesColumn** — The column containing missing values (null values and empty strings). Required.
- **outputColumn** — The name of the new column that will contain estimated values for all rows whose value was missing. Optional; the default is the value of `missingValuesColumn` suffixed by `_filled`.
- **transformationContext** — A unique string that is used to identify state information (optional).
- **callSite** — Used to provide context information for error reporting. (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 dynamic frame with one additional column that contains estimations for rows with missing values and the present value for other rows.

FindMatches Class

Package: com.amazonaws.services.glue.ml

```
object FindMatches
```

def apply

```
def apply(frame: DynamicFrame,  
         transformId: String,  
         transformationContext: String = "",  
         callSite: CallSite = CallSite("Not provided", ""),  
         stageThreshold: Long = 0,  
         totalThreshold: Long = 0,  
         enforcedMatches: DynamicFrame = null): DynamicFrame,  
         computeMatchConfidenceScores: Boolean
```

Find matches in an input frame and return a new frame with a new column containing a unique ID per match group.

- **frame** — The DynamicFrame in which to find matches. Required.

- `transformId` — A unique ID associated with the `FindMatches` transform to apply on the input frame. Required.
- `transformationContext` — Identifier for this `DynamicFrame`. The `transformationContext` is used as a key for the job bookmark state that is persisted across runs. Optional.
- `callSite` — Used to provide context information for error reporting. These values are automatically set when calling from Python. Optional.
- `stageThreshold` — The maximum number of error records allowed from the computation of this `DynamicFrame` before throwing an exception, excluding records present in the previous `DynamicFrame`. Optional. The default is zero.
- `totalThreshold` — The maximum number of total errors records before an exception is thrown, including those from previous frames. Optional. The default is zero.
- `enforcedMatches` — The frame for enforced matches. Optional. The default is `null`.
- `computeMatchConfidenceScores` — A Boolean value indicating whether to compute a confidence score for each group of matching records. Optional. The default is `false`.

Returns a new dynamic frame with a unique identifier assigned to each group of matching records.

FindIncrementalMatches Class

Package: `com.amazonaws.services.glue.ml`

```
object FindIncrementalMatches
```

def apply

```
apply(existingFrame: DynamicFrame,
      incrementalFrame: DynamicFrame,
      transformId: String,
      transformationContext: String = "",
      callSite: CallSite = CallSite("Not provided", ""),
      stageThreshold: Long = 0,
      totalThreshold: Long = 0,
      enforcedMatches: DynamicFrame = null): DynamicFrame,
      computeMatchConfidenceScores: Boolean
```

Find matches across the existing and incremental frames and return a new frame with a column containing a unique ID per match group.

- `existingframe` — An existing frame which has been assigned a matching ID for each group. Required.
- `incrementalframe` — An incremental frame used to find matches against the existing frame. Required.
- `transformId` — A unique ID associated with the `FindIncrementalMatches` transform to apply on the input frames. Required.
- `transformationContext` — Identifier for this `DynamicFrame`. The `transformationContext` is used as a key for the job bookmark state that is persisted across runs. Optional.
- `callSite` — Used to provide context information for error reporting. These values are automatically set when calling from Python. Optional.
- `stageThreshold` — The maximum number of error records allowed from the computation of this `DynamicFrame` before throwing an exception, excluding records present in the previous `DynamicFrame`. Optional. The default is zero.
- `totalThreshold` — The maximum number of total errors records before an exception is thrown, including those from previous frames. Optional. The default is zero.

- `enforcedMatches` — The frame for enforced matches. Optional. The default is `null`.
- `computeMatchConfidenceScores` — A Boolean value indicating whether to compute a confidence score for each group of matching records. Optional. The default is `false`.

Returns a new dynamic frame with a unique identifier assigned to each group of matching records.

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 \(p. 677\)](#)
- [NullNode Case Object \(p. 677\)](#)

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 \(p. 677\)](#)
- [ObjectNode Case Class \(p. 678\)](#)

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 \(p. 678\)](#)
- [ScalarNode Object \(p. 679\)](#)

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 \(p. 691\)](#).

Note

AWS Glue version 2.0 FindMatches jobs use the Amazon S3 bucket `aws-glue-temp-<accountID>-<region>` to store temporary files while the transform is processing data. You can delete this data after the run has completed, either manually or by setting an Amazon S3 Lifecycle rule.

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 incremental matches

The Find matches transform can also be configured to find matches across the existing and incremental frames and return as output a column containing a unique ID per match group.

For more information, see: [Finding Incremental Matches \(p. 699\)](#)

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.

Note

Currently, `FindMatches` transforms that use a custom encryption key aren't supported in the following Regions:

- Asia Pacific (Osaka) - `ap-northeast-3`

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 \(p. 684\)](#). Proceed to step 4.
4. Download the labeling file and label the file as described in the [Labeling \(p. 684\)](#) 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 \(p. 686\)](#).

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 \(p. 682\)](#).

Topics

- [Machine Learning Measurements \(p. 686\)](#)
- [Deciding Between Precision and Recall \(p. 687\)](#)
- [Deciding Between Accuracy and Cost \(p. 688\)](#)
- [Estimating the quality of matches using match confidence scores \(p. 688\)](#)
- [Teaching the Find Matches Transform \(p. 690\)](#)

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 among the total number of records that it identifies as positive (true positives and false positives). 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."

Estimating the quality of matches using match confidence scores

Match confidence scores provide an estimate of the quality of matches found by `FindMatches` to distinguish between matched records in which the machine learning model is highly confident, uncertain, or unlikely. A match confidence score will be between 0 and 1, where a higher score means higher similarity. Examining match confidence scores lets you distinguish between clusters of matches in which the system is highly confident (which you may decide to merge), clusters about which the system is

uncertain (which you may decide to have reviewed by a human), and clusters that the system deems to be unlikely (which you may decide to reject).

You may want to adjust your training data in situations where you see a high match confidence score, but determine there are not matches, or where you see a low score but determine there are, in fact, matches.

Confidence scores are particularly useful when there are large sized industrial datasets, where it is infeasible to review every FindMatches decision.

Match confidence scores are available in AWS Glue version 2.0 or later.

Generating match confidence scores

You can generate match confidence scores by setting the Boolean value of `computeMatchConfidenceScores` to True when calling the `FindMatches` or `FindIncrementalMatches` API.

AWS Glue adds a new column `match_confidence_score` to the output.

Match Scoring Examples

For example, consider the following matched records:

Score >= 0.9

Summary of matched records:

primary_id	match_id	match_confidence_score
3281355037663	85899345947	0.9823658302132061
1546188247619	85899345947	0.9823658302132061

Details:

raw_id	phone[source]	website	poi_id	display_position	primary_name[locale_name]	street1 street2 street3
[ae]6e5001c0-0efppL1 u +4326581661 yelp https://www.commercialbank.at yelp:iaeJqB5D0ICdIqHFPPL1 1 geo:47.711590000,16.344020000 Commercialbank Mattersburg en_US Hauptstr. 59 null null Forchtenstein 1 AT 7212						
9 1546188247619 85899345947 yelp https://www.commercialbank.at yelp:uhQk6v2jj51Z4N8lXn-q0 geo:47.787420000,16.455440000 Commercialbank Mattersburg en_US Hauptstr. 9 null null Markt 1 AT 7024						
Hauptstr. 9 3281355037663 85899345947 yelp https://www.commercialbank.at yelp:uhQk6v2jj51Z4N8lXn-q0 geo:47.787420000,16.455440000 Commercialbank Mattersburg en_US Hauptstr. 9 null null Markt 1 AT 7024						

From this example, we can see that two records are very similar and share `display_position`, `primary_name`, and `street` name.

Score >= 0.8 and score < 0.9

Summary of matched records:

primary_id	match_id	match_confidence_score
309237680432	85899345928	0.8309852373674638
3590592666790	85899345928	0.8309852373674638
343597390617	85899345928	0.8309852373674638
249108124906	85899345928	0.8309852373674638
463856477937	85899345928	0.8309852373674638

Details:

primary_id	raw_id	phone[source]	website	poi_id	display_position	primary_name[locale_name]	street1 street2 street3	city state country postal_code street_in_one_line
NNIMVA35Tm4Imraqkyvr_w	null yelp null yelp::NNIMVA35Tm4Imraqkyvr_w geo:50.541800000,7.102920000 Eiscafe Dolomiten en_US Ahrhutstr. 49 null null Bad Neuenahr-Ahrweiler RP DE 53474 Ahrhutstr. 49							
343597390617 85899345928	0.8309852373674638 yelp null yelp::0.8309852373674638 geo:50.541800000,7.102920000 Eiscafe Dolomiten en_US Markt 5 null null Gebenstein HE DE 34393 Markt 5							
463856477937 85899345928	0.8309852373674638 yelp null yelp::0.8309852373674638 geo:50.541800000,7.102920000 Eiscafe Dolomiten en_US Alexanderstr. 105 null null Eisenach TH DE 99817 Alexanderstr. 105							
309237680432 85899345928	0.8309852373674638 yelp null yelp::0.8309852373674638 geo:50.541800000,7.102920000 Eiscafe Dolomiten en_US Rheinstr. 15 null null Linz RP DE 53545 Rheinstr.							
1513590572600779 85899345928	0.8309852373674638 yelp null yelp::0.8309852373674638 geo:50.541800000,7.102920000 Eiscafe Dolomiten en_US Rheinstr. 15 null null Linz RP DE 53545 Rheinstr.							

From this example, we can see that these records share the same primary_name, and country.

Score >= 0.6 and score < 0.7

Summary of matched records:

primary_id	match_id	match_confidence_score
2164663519676	85899345930	0.6971099896480333
317827595278	85899345930	0.6971099896480333
472446424341	85899345930	0.6971099896480333
3118146262932	85899345930	0.6971099896480333
214748380804	85899345930	0.6971099896480333

Details:

From this example, we can see that these records share only the same primary_name.

For more information, see:

- Step 5: Add and Run a Job with Your Machine Learning Transform (p. 697)
 - PySpark: [FindMatches Class](#) (p. 602)
 - PySpark: [FindIncrementalMatches Class](#) (p. 601)
 - Scala: [FindMatches Class](#) (p. 673)
 - Scala: [FindIncrementalMatches Class](#) (p. 674)

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` \(p. 682\)](#).

Topics

- [Transform Properties \(p. 691\)](#)
- [Adding and Editing Machine Learning Transforms \(p. 692\)](#)
- [Viewing Transform Details \(p. 693\)](#)

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 **ML transforms** from the left-side navigation menu.

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**.

Label count

The number of labels in the labeling file that was provided to help teach the transform.

Status

Indicates whether the transform is **Ready** or **Needs training**. To run a machine learning transform successfully in a job, it must be **Ready**.

Date created

The date the transform was created.

Last modified

The date the transform was last updated.

Description

The description supplied for the transform, if one was provided.

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**.

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 \(p. 682\)](#).

Using Data Encryption with Machine Learning Transforms

When adding a machine learning transform to AWS Glue, you can optionally specify a security configuration that is associated with the data source or data target. If the Amazon S3 bucket used to store the data is encrypted with a security configuration, specify the same security configuration when creating the transform.

You can also choose to use server-side encryption with AWS KMS (SSE-KMS) to encrypt the model and labels to prevent unauthorized persons from inspecting it. If you choose this option, you're prompted to choose the AWS KMS key by name, or you can choose **Enter a key ARN**. If you choose to enter the ARN for the KMS key, a second field appears where you can enter the KMS key ARN.

Note

Currently, `FindMatches` transforms that use a custom encryption key aren't supported in the following Regions:

- Asia Pacific (Osaka) - `ap-northeast-3`

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.
- **Spark version** is related to the AWS Glue version you chose in the **Task run properties** when adding the transform. AWS Glue 1.0 and Spark 2.4 is recommended for most customers. For more information, see [AWS Glue Versions](#).

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.
- The **Column importance** table show the column names and importance score for each column. Column importance helps you understand how columns contribute to your model, by identifying which columns in your records are being used the most to do the matching. This data may prompt you to add to or change your labelset to raise or lower the importance of columns.

The Importance column provides a numerical score for each column, as a decimal not greater than 1.0.

For information about understanding quality estimates versus true quality, see [Quality Estimates Versus End-to-End \(True\) Quality \(p. 694\)](#).

For more information about tuning your transform, see [Tuning Machine Learning Transforms in AWS Glue \(p. 686\)](#).

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 the cost/accuracy tradeoff for your matches 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 \(p. 695\)](#)
- [Step 2: Add a Machine Learning Transform \(p. 695\)](#)
- [Step 3: Teach Your Machine Learning Transform \(p. 696\)](#)
- [Step 4: Estimate the Quality of Your Machine Learning Transform \(p. 696\)](#)
- [Step 5: Add and Run a Job with Your Machine Learning Transform \(p. 697\)](#)
- [Step 6: Verify Output Data from Amazon S3 \(p. 698\)](#)

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-dblp-acm` with output to database `demo-db-dblp-acm`. When running the wizard, create the database `demo-db-dblp-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/dblp-acm/records/dblp_acm_records.csv`.

If successful, the crawler creates the table `dblp_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-dblp-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-dblp-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 `dblp_acm_records_csv` in database `demo-db-dblp-acm`.
 - d. For **Primary key**, choose the primary key column for the table, `id`.
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 `dblp_acm_labels.csv`. For more information about the labeling process, see [Labeling \(p. 684\)](#).

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 \(p. 694\)](#).

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 = "t fm-123456789012", emitFusion = false,
        survivorComparisonField = "<primary_id>", transformation_ctx = "findmatches1"]
        // @return: findmatches1
        // @inputs: [frame = datasource0]
        val findmatches1 = FindMatches.apply(frame = datasource0, transformId
        = "t fm-123456789012", transformationContext = "findmatches1",
        computeMatchConfidenceScores = true)

        // 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-dblp-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.

	B	C	D	E	F	G	H
1	Title	authors	venue	year	source	primary_id	match_id
2	Semantic Integration of Environmental Models for Application to Global Information	E.D. Scott Mackay	SIGMOD Record	1999	DBLP	3092	0
3	Semantic integration of environmental models for application to global information	E.D. Scott Mackay	ACM SIGMOD Recor	1999	ACM	3590	0
4	Estimation of Query-Result Distribution and Its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E.	VLDB	1996	DBLP	3435	1
5	Estimation of Query-Result Distribution and Its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E.	Very Large Data Bas	1996	ACM	2491	1
6	Incremental Maintenance for Non-Distributive Aggregate Functions	Theofilakis Palpanas, Richa	VLDB	2002	DBLP	4638	2
7	Cost-based Selection of Join Operations in Object-Oriented Databases	Tang, Richard J., et al.	VLDB	2002	DBLP	3735	3
8	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented D	Georges Ouzounis, Jean-Robert	Very Large Data Bas	1996	ACM	5526	3
9	Benchmarking Spatial Join Operations with Spatial Output	Erk G. Hoel, Hanan Samet	VLDB	1995	DBLP	9739	4
10	Benchmarking Spatial Join Operations with Spatial Output	Erk G. Hoel, Hanan Samet	Very Large Data Bas	1995	ACM	8124	4
11	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	International Confe	1999	ACM	5647	5
12	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	SIGMOD Conference	1999	DBLP	3432	5
13	Managing the World Wide Web: An Information System Approach - Book Review	Albert Olszak	SIGMOD Record	2000	DBLP	6799	6
14	Enhanced Abstract Data Types in Object-relational Databases	Praveen Sethadri	VLDB J.	1998	DBLP	3617	7
15	Enhanced abstract data types in object-relational databases	Praveen Sethadri	The VLDB Journal &	1998	ACM	4906	7
16	Report on DART '96 Databases: Active and Real-Time (Concepts meet Practice)	Nandit Soparkar, Krithi Ramamirtham	SIGMOD Record	1997	DBLP	7937	8
17	Report on DART '96 databases: active and real-time (concepts meet practice)	Krithi Ramamirtham, Nandit S	ACM SIGMOD Recor	1997	ACM	8193	8
18	UNISQL's next-generation object-relational database management system	Albert D'Andrea, Phil Janus	ACM SIGMOD Recor	1996	ACM	8491	9
19	UNISQL's Next-Generation Object-Relational Database Management System	Phil Janus, Albert D'Andrea	SIGMOD Record	1996	DBLP	4869	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. The `match_confidence_score` is a number between 0 and 1 that provides an estimate of the quality of matches found by `Find matches`.

- 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`.

Finding Incremental Matches

The `Find matches` feature allows you to identify duplicate or matching records in your dataset, even when the records don't have a common unique identifier and no fields match exactly. The initial release of the `Find matches` transform identified matching records within a single dataset. When you add new data to the dataset, you had to merge it with the existing clean dataset and rerun matching against the complete merged dataset.

The incremental matching feature makes it simpler to match to incremental records against existing matched datasets. Suppose that you want to match prospects data with existing customer datasets. The incremental match capability provides you the flexibility to match hundreds of thousands of new prospects with an existing database of prospects and customers by merging the results into a single database or table. By matching only between the new and existing datasets, the `find incremental matches` optimization reduces computation time, which also reduces cost.

The usage of incremental matching is similar to `Find matches` as described in [Tutorial: Creating a Machine Learning Transform with AWS Glue \(p. 695\)](#). This topic identifies only the differences with incremental matching.

For more information, see the blog post on [Incremental data matching](#).

Running an incremental matching job

For the following procedure, suppose the following:

- You have crawled the existing dataset into table `first_records` and created and trained a `Find matches` transform with AWS Glue version 2.0. This is the only version of AWS Glue that supports incremental matches.
- The ETL language is Scala. Note that Python is also supported.
- The model already generated is called `demo-xform`.

- Crawl the incremental dataset to the table `second_records`.

2. On the AWS Glue console, in the navigation pane, choose **Jobs**.
3. 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 **demo-etl**.
 - b. For **IAM role**, choose an IAM role with permission to the Amazon S3 source data, labeling file, and [AWS Glue API operations](#).
 - c. For **ETL language**, choose **Scala**.
 - d. For **Script file name**, choose **demo-etl**. This is the file name of the Scala script.
 - e. For **Data source**, choose **first_records**. 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. Select the incremental matching option, and for **Data Source** select the table named **second_records**.
 - h. For **Transform**, choose **demo-xform**, the machine learning transform used by the job.
 - i. Choose **Create tables in your data target** or **Use tables in the data catalog and update your data target**.
4. Choose **Save job and edit script** to display the script editor page.
5. Choose **Run job** to start the job run.

AWS Glue API

Contents

- [Security APIs in AWS Glue \(p. 712\)](#)
 - [Data Types \(p. 712\)](#)
 - [DataCatalogEncryptionSettings Structure \(p. 712\)](#)
 - [EncryptionAtRest Structure \(p. 713\)](#)
 - [ConnectionPasswordEncryption Structure \(p. 713\)](#)
 - [EncryptionConfiguration Structure \(p. 714\)](#)
 - [S3Encryption Structure \(p. 714\)](#)
 - [CloudWatchEncryption Structure \(p. 714\)](#)
 - [JobBookmarksEncryption Structure \(p. 714\)](#)
 - [SecurityConfiguration Structure \(p. 715\)](#)
 - [GluePolicy Structure \(p. 715\)](#)
 - [Operations \(p. 715\)](#)
 - [GetDataCatalogEncryptionSettings Action \(Python: get_data_catalog_encryption_settings\) \(p. 716\)](#)
 - [PutDataCatalogEncryptionSettings Action \(Python: put_data_catalog_encryption_settings\) \(p. 716\)](#)
 - [PutResourcePolicy Action \(Python: put_resource_policy\) \(p. 717\)](#)
 - [GetResourcePolicy Action \(Python: get_resource_policy\) \(p. 718\)](#)
 - [DeleteResourcePolicy Action \(Python: delete_resource_policy\) \(p. 718\)](#)
 - [CreateSecurityConfiguration Action \(Python: create_security_configuration\) \(p. 719\)](#)
 - [DeleteSecurityConfiguration Action \(Python: delete_security_configuration\) \(p. 720\)](#)
 - [GetSecurityConfiguration Action \(Python: get_security_configuration\) \(p. 720\)](#)
 - [GetSecurityConfigurations Action \(Python: get_security_configurations\) \(p. 721\)](#)
 - [GetResourcePolicies Action \(Python: get_resource_policies\) \(p. 721\)](#)
- [Catalog API \(p. 722\)](#)
 - [Database API \(p. 722\)](#)
 - [Data Types \(p. 722\)](#)
 - [Database Structure \(p. 722\)](#)
 - [DatabaseInput Structure \(p. 723\)](#)
 - [PrincipalPermissions Structure \(p. 724\)](#)
 - [DataLakePrincipal Structure \(p. 724\)](#)
 - [DatabaseIdentifier Structure \(p. 724\)](#)
 - [Operations \(p. 725\)](#)
 - [CreateDatabase Action \(Python: create_database\) \(p. 725\)](#)
 - [UpdateDatabase Action \(Python: update_database\) \(p. 725\)](#)
 - [DeleteDatabase Action \(Python: delete_database\) \(p. 726\)](#)
 - [GetDatabase Action \(Python: get_database\) \(p. 727\)](#)
 - [GetDatabases Action \(Python: get_databases\) \(p. 727\)](#)
 - [Table API \(p. 728\)](#)
 - [Data Types \(p. 728\)](#)
 - [Table Structure \(p. 729\)](#)
 - [TableInput Structure \(p. 730\)](#)

- [Column Structure \(p. 731\)](#)
 - [StorageDescriptor Structure \(p. 732\)](#)
 - [SchemaReference Structure \(p. 733\)](#)
 - [SerDeInfo Structure \(p. 733\)](#)
 - [Order Structure \(p. 734\)](#)
 - [SkewedInfo Structure \(p. 734\)](#)
 - [TableVersion Structure \(p. 734\)](#)
 - [TableError Structure \(p. 734\)](#)
 - [TableVersionError Structure \(p. 735\)](#)
 - [SortCriterion Structure \(p. 735\)](#)
 - [TableIdentifier Structure \(p. 735\)](#)
 - [KeySchemaElement Structure \(p. 736\)](#)
 - [PartitionIndex Structure \(p. 736\)](#)
 - [PartitionIndexDescriptor Structure \(p. 736\)](#)
 - [BackfillError Structure \(p. 737\)](#)
 - [Operations \(p. 737\)](#)
 - [CreateTable Action \(Python: `create_table`\) \(p. 738\)](#)
 - [UpdateTable Action \(Python: `update_table`\) \(p. 739\)](#)
 - [DeleteTable Action \(Python: `delete_table`\) \(p. 739\)](#)
 - [BatchDeleteTable Action \(Python: `batch_delete_table`\) \(p. 740\)](#)
 - [GetTable Action \(Python: `get_table`\) \(p. 741\)](#)
 - [GetTables Action \(Python: `get_tables`\) \(p. 742\)](#)
 - [GetTableVersion Action \(Python: `get_table_version`\) \(p. 743\)](#)
 - [GetTableVersions Action \(Python: `get_table_versions`\) \(p. 744\)](#)
 - [DeleteTableVersion Action \(Python: `delete_table_version`\) \(p. 745\)](#)
 - [BatchDeleteTableVersion Action \(Python: `batch_delete_table_version`\) \(p. 745\)](#)
 - [SearchTables Action \(Python: `search_tables`\) \(p. 746\)](#)
 - [GetPartitionIndexes Action \(Python: `get_partition_indexes`\) \(p. 747\)](#)
 - [CreatePartitionIndex Action \(Python: `create_partition_index`\) \(p. 748\)](#)
 - [DeletePartitionIndex Action \(Python: `delete_partition_index`\) \(p. 749\)](#)
 - [GetColumnStatisticsForTable Action \(Python: `get_column_statistics_for_table`\) \(p. 749\)](#)
 - [UpdateColumnStatisticsForTable Action \(Python: `update_column_statistics_for_table`\) \(p. 750\)](#)
 - [DeleteColumnStatisticsForTable Action \(Python: `delete_column_statistics_for_table`\) \(p. 751\)](#)
 - [Partition API \(p. 752\)](#)
 - [Data Types \(p. 752\)](#)
 - [Partition Structure \(p. 752\)](#)
 - [PartitionInput Structure \(p. 753\)](#)
 - [PartitionSpecWithSharedStorageDescriptor Structure \(p. 753\)](#)
 - [PartitionListComposingSpec Structure \(p. 754\)](#)
 - [PartitionSpecProxy Structure \(p. 754\)](#)
 - [PartitionValueList Struct \(p. 754\)](#)
 - [Segment Structure \(p. 755\)](#)
 - [PartitionError Structure \(p. 755\)](#)
-

- [BatchUpdatePartitionFailureEntry Structure \(p. 755\)](#)
- [BatchUpdatePartitionRequestEntry Structure \(p. 755\)](#)
- [Operations \(p. 756\)](#)
- [CreatePartition Action \(Python: create_partition\) \(p. 756\)](#)
- [BatchCreatePartition Action \(Python: batch_create_partition\) \(p. 757\)](#)
- [UpdatePartition Action \(Python: update_partition\) \(p. 757\)](#)
- [DeletePartition Action \(Python: delete_partition\) \(p. 758\)](#)
- [BatchDeletePartition Action \(Python: batch_delete_partition\) \(p. 759\)](#)
- [GetPartition Action \(Python: get_partition\) \(p. 759\)](#)
- [GetPartitions Action \(Python: get_partitions\) \(p. 760\)](#)
- [BatchGetPartition Action \(Python: batch_get_partition\) \(p. 763\)](#)
- [BatchUpdatePartition Action \(Python: batch_update_partition\) \(p. 764\)](#)
- [GetColumnStatisticsForPartition Action \(Python: get_column_statistics_for_partition\) \(p. 765\)](#)
- [UpdateColumnStatisticsForPartition Action \(Python: update_column_statistics_for_partition\) \(p. 766\)](#)
- [DeleteColumnStatisticsForPartition Action \(Python: delete_column_statistics_for_partition\) \(p. 767\)](#)
- [Connection API \(p. 767\)](#)
 - [Data Types \(p. 767\)](#)
 - [Connection Structure \(p. 768\)](#)
 - [ConnectionInput Structure \(p. 770\)](#)
 - [PhysicalConnectionRequirements Structure \(p. 771\)](#)
 - [GetConnectionsFilter Structure \(p. 771\)](#)
 - [Operations \(p. 772\)](#)
 - [CreateConnection Action \(Python: create_connection\) \(p. 772\)](#)
 - [DeleteConnection Action \(Python: delete_connection\) \(p. 773\)](#)
 - [GetConnection Action \(Python: get_connection\) \(p. 773\)](#)
 - [GetConnections Action \(Python: get_connections\) \(p. 774\)](#)
 - [UpdateConnection Action \(Python: update_connection\) \(p. 775\)](#)
 - [BatchDeleteConnection Action \(Python: batch_delete_connection\) \(p. 775\)](#)
- [User-Defined Function API \(p. 776\)](#)
 - [Data Types \(p. 776\)](#)
 - [UserDefinedFunction Structure \(p. 776\)](#)
 - [UserDefinedFunctionInput Structure \(p. 777\)](#)
 - [Operations \(p. 777\)](#)
 - [CreateUserDefinedFunction Action \(Python: create_user_defined_function\) \(p. 777\)](#)
 - [UpdateUserDefinedFunction Action \(Python: update_user_defined_function\) \(p. 778\)](#)
 - [DeleteUserDefinedFunction Action \(Python: delete_user_defined_function\) \(p. 779\)](#)
 - [GetUserDefinedFunction Action \(Python: get_user_defined_function\) \(p. 779\)](#)
 - [GetUserDefinedFunctions Action \(Python: get_user_defined_functions\) \(p. 780\)](#)
- [Importing an Athena Catalog to AWS Glue \(p. 781\)](#)
 - [Data Types \(p. 781\)](#)
 - [CatalogImportStatus Structure \(p. 781\)](#)

- [Operations \(p. 781\)](#)
 - [ImportCatalogToGlue Action \(Python: import_catalog_to_glue\) \(p. 781\)](#)
 - [GetCatalogImportStatus Action \(Python: get_catalog_import_status\) \(p. 782\)](#)
 - [Crawlers and Classifiers API \(p. 782\)](#)
 - [Classifier API \(p. 782\)](#)
 - [Data Types \(p. 782\)](#)
 - [Classifier Structure \(p. 783\)](#)
 - [GrokClassifier Structure \(p. 783\)](#)
 - [XMLClassifier Structure \(p. 784\)](#)
 - [JsonClassifier Structure \(p. 784\)](#)
 - [CsvClassifier Structure \(p. 785\)](#)
 - [CreateGrokClassifierRequest Structure \(p. 786\)](#)
 - [UpdateGrokClassifierRequest Structure \(p. 786\)](#)
 - [CreateXMLClassifierRequest Structure \(p. 786\)](#)
 - [UpdateXMLClassifierRequest Structure \(p. 787\)](#)
 - [CreateJsonClassifierRequest Structure \(p. 787\)](#)
 - [UpdateJsonClassifierRequest Structure \(p. 787\)](#)
 - [CreateCsvClassifierRequest Structure \(p. 788\)](#)
 - [UpdateCsvClassifierRequest Structure \(p. 788\)](#)
 - [Operations \(p. 789\)](#)
 - [CreateClassifier Action \(Python: create_classifier\) \(p. 789\)](#)
 - [DeleteClassifier Action \(Python: delete_classifier\) \(p. 790\)](#)
 - [GetClassifier Action \(Python: get_classifier\) \(p. 790\)](#)
 - [GetClassifiers Action \(Python: get_classifiers\) \(p. 790\)](#)
 - [UpdateClassifier Action \(Python: update_classifier\) \(p. 791\)](#)
 - [Crawler API \(p. 792\)](#)
 - [Data Types \(p. 792\)](#)
 - [Crawler Structure \(p. 792\)](#)
 - [Schedule Structure \(p. 793\)](#)
 - [CrawlerTargets Structure \(p. 794\)](#)
 - [S3Target Structure \(p. 794\)](#)
 - [JdbcTarget Structure \(p. 795\)](#)
 - [MongoDBTarget Structure \(p. 795\)](#)
 - [DynamoDBTarget Structure \(p. 795\)](#)
 - [DeltaTarget Structure \(p. 796\)](#)
 - [CatalogTarget Structure \(p. 796\)](#)
 - [CrawlerMetrics Structure \(p. 796\)](#)
 - [SchemaChangePolicy Structure \(p. 797\)](#)
 - [LastCrawlInfo Structure \(p. 797\)](#)
 - [RecrawlPolicy Structure \(p. 798\)](#)
 - [LineageConfiguration Structure \(p. 798\)](#)
 - [LakeFormationConfiguration Structure \(p. 798\)](#)
 - [Operations \(p. 799\) 704](#)
 - [CreateCrawler Action \(Python: create_crawler\) \(p. 799\)](#)
 - [DeleteCrawler Action \(Python: delete_crawler\) \(p. 800\)](#)
-

- [GetCrawler Action \(Python: get_crawler\) \(p. 801\)](#)
- [GetCrawlers Action \(Python: get_crawlers\) \(p. 801\)](#)
- [GetCrawlerMetrics Action \(Python: get_crawler_metrics\) \(p. 802\)](#)
- [UpdateCrawler Action \(Python: update_crawler\) \(p. 802\)](#)
- [StartCrawler Action \(Python: start_crawler\) \(p. 804\)](#)
- [StopCrawler Action \(Python: stop_crawler\) \(p. 804\)](#)
- [BatchGetCrawlers Action \(Python: batch_get_crawlers\) \(p. 804\)](#)
- [ListCrawlers Action \(Python: list_crawlers\) \(p. 805\)](#)
- [Crawler Scheduler API \(p. 806\)](#)
 - [Data Types \(p. 806\)](#)
 - [Schedule Structure \(p. 806\)](#)
 - [Operations \(p. 806\)](#)
 - [UpdateCrawlerSchedule Action \(Python: update_crawler_schedule\) \(p. 806\)](#)
 - [StartCrawlerSchedule Action \(Python: start_crawler_schedule\) \(p. 807\)](#)
 - [StopCrawlerSchedule Action \(Python: stop_crawler_schedule\) \(p. 807\)](#)
- [Autogenerating ETL Scripts API \(p. 808\)](#)
 - [Data Types \(p. 808\)](#)
 - [CodeGenNode Structure \(p. 808\)](#)
 - [CodeGenNodeArg Structure \(p. 808\)](#)
 - [CodeGenEdge Structure \(p. 809\)](#)
 - [Location Structure \(p. 809\)](#)
 - [CatalogEntry Structure \(p. 809\)](#)
 - [MappingEntry Structure \(p. 810\)](#)
 - [Operations \(p. 810\)](#)
 - [CreateScript Action \(Python: create_script\) \(p. 810\)](#)
 - [GetDataflowGraph Action \(Python: get_dataflow_graph\) \(p. 811\)](#)
 - [GetMapping Action \(Python: get_mapping\) \(p. 811\)](#)
 - [GetPlan Action \(Python: get_plan\) \(p. 812\)](#)
- [Visual Job API \(Preview\) \(p. 813\)](#)
 - [Data Types \(p. 813\)](#)
 - [CodeGenConfigurationNode Structure \(p. 814\)](#)
 - [JDBCConnectorOptions Structure \(p. 817\)](#)
 - [StreamingDataPreviewOptions Structure \(p. 818\)](#)
 - [AthenaConnectorSource Structure \(p. 818\)](#)
 - [JDBCConnectorSource Structure \(p. 819\)](#)
 - [SparkConnectorSource Structure \(p. 819\)](#)
 - [CatalogSource Structure \(p. 820\)](#)
 - [CatalogKinesisSource Structure \(p. 820\)](#)
 - [DirectKinesisSource Structure \(p. 821\)](#)
 - [KinesisStreamingSourceOptions Structure \(p. 821\)](#)
 - [CatalogKafkaSource Structure \(p. 823\)](#)
 - [DirectKafkaSource Structure \(p. 823\)](#)
 - [KafkaStreamingSourceOptions Structure \(p. 824\)
705](#)
 - [RedshiftSource Structure \(p. 825\)](#)
 - [S3CatalogSource Structure \(p. 826\)](#)

- [S3SourceAdditionalOptions Structure \(p. 826\)](#)
- [S3CSVSource Structure \(p. 826\)](#)
- [S3JsonSource Structure \(p. 828\)](#)
- [S3ParquetSource Structure \(p. 829\)](#)
- [DynamoDBELTConnectorSource Structure \(p. 830\)](#)
- [DDBELTConnectionOptions Structure \(p. 830\)](#)
- [JDBCConnectorTarget Structure \(p. 831\)](#)
- [SparkConnectorTarget Structure \(p. 831\)](#)
- [BasicCatalogTarget Structure \(p. 832\)](#)
- [RedshiftTarget Structure \(p. 833\)](#)
- [S3CatalogTarget Structure \(p. 833\)](#)
- [S3GlueParquetTarget Structure \(p. 834\)](#)
- [CatalogSchemaChangePolicy Structure \(p. 834\)](#)
- [S3DirectTarget Structure \(p. 834\)](#)
- [DirectSchemaChangePolicy Structure \(p. 835\)](#)
- [ApplyMapping Structure \(p. 835\)](#)
- [Mapping Structure \(p. 836\)](#)
- [SelectFields Structure \(p. 837\)](#)
- [DropFields Structure \(p. 837\)](#)
- [RenameField Structure \(p. 837\)](#)
- [Spigot Structure \(p. 838\)](#)
- [Join Structure \(p. 838\)](#)
- [JoinColumn Structure \(p. 838\)](#)
- [SplitFields Structure \(p. 839\)](#)
- [SelectFromCollection Structure \(p. 839\)](#)
- [FillMissingValues Structure \(p. 839\)](#)
- [Filter Structure \(p. 840\)](#)
- [FilterExpression Structure \(p. 840\)](#)
- [FilterValue Structure \(p. 841\)](#)
- [CustomCode Structure \(p. 841\)](#)
- [SparkSQL Structure \(p. 841\)](#)
- [SqlAlias Structure \(p. 842\)](#)
- [DropNullFields Structure \(p. 842\)](#)
- [NullCheckBoxList Structure \(p. 843\)](#)
- [NullValueField Structure \(p. 843\)](#)
- [Datatype Structure \(p. 843\)](#)
- [Merge Structure \(p. 843\)](#)
- [Union Structure \(p. 844\)](#)
- [Jobs API \(p. 844\)](#)
 - [Jobs \(p. 845\)](#)
 - [Data Types \(p. 845\)](#)
 - [Job Structure \(p. 845\)](#)
 - [ExecutionProperty Structure 706 \(p. 847\)](#)
 - [NotificationProperty Structure \(p. 847\)](#)
 - [JobCommand Structure \(p. 847\)](#)

- [ConnectionsList Structure \(p. 848\)](#)
- [JobUpdate Structure \(p. 848\)](#)
- [Operations \(p. 850\)](#)
- [CreateJob Action \(Python: create_job\) \(p. 850\)](#)
- [UpdateJob Action \(Python: update_job\) \(p. 853\)](#)
- [GetJob Action \(Python: get_job\) \(p. 853\)](#)
- [GetJobs Action \(Python: get_jobs\) \(p. 854\)](#)
- [DeleteJob Action \(Python: delete_job\) \(p. 854\)](#)
- [ListJobs Action \(Python: list_jobs\) \(p. 855\)](#)
- [BatchGetJobs Action \(Python: batch_get_jobs\) \(p. 856\)](#)
- [Job Runs \(p. 856\)](#)
 - [Data Types \(p. 856\)](#)
 - [JobRun Structure \(p. 856\)](#)
 - [Predecessor Structure \(p. 859\)](#)
 - [JobBookmarkEntry Structure \(p. 859\)](#)
 - [BatchStopJobRunSuccessfulSubmission Structure \(p. 860\)](#)
 - [BatchStopJobRunError Structure \(p. 860\)](#)
 - [Operations \(p. 860\)](#)
 - [StartJobRun Action \(Python: start_job_run\) \(p. 860\)](#)
 - [BatchStopJobRun Action \(Python: batch_stop_job_run\) \(p. 862\)](#)
 - [GetJobRun Action \(Python: get_job_run\) \(p. 863\)](#)
 - [GetJobRuns Action \(Python: get_job_runs\) \(p. 863\)](#)
 - [GetJobBookmark Action \(Python: get_job_bookmark\) \(p. 864\)](#)
 - [GetJobBookmarks Action \(Python: get_job_bookmarks\) \(p. 865\)](#)
 - [ResetJobBookmark Action \(Python: reset_job_bookmark\) \(p. 865\)](#)
- [Triggers \(p. 866\)](#)
 - [Data Types \(p. 866\)](#)
 - [Trigger Structure \(p. 866\)](#)
 - [TriggerUpdate Structure \(p. 867\)](#)
 - [Predicate Structure \(p. 867\)](#)
 - [Condition Structure \(p. 868\)](#)
 - [Action Structure \(p. 868\)](#)
 - [EventBatchingCondition Structure \(p. 869\)](#)
 - [Operations \(p. 869\)](#)
 - [CreateTrigger Action \(Python: create_trigger\) \(p. 869\)](#)
 - [StartTrigger Action \(Python: start_trigger\) \(p. 871\)](#)
 - [GetTrigger Action \(Python: get_trigger\) \(p. 871\)](#)
 - [GetTriggers Action \(Python: get_triggers\) \(p. 872\)](#)
 - [UpdateTrigger Action \(Python: update_trigger\) \(p. 872\)](#)
 - [StopTrigger Action \(Python: stop_trigger\) \(p. 873\)](#)
 - [DeleteTrigger Action \(Python: delete_trigger\) \(p. 874\)](#)
 - [ListTriggers Action \(Python: list_triggers\) \(p. 874\)](#)
 - [BatchGetTriggers Action \(Python: batch_get_triggers\) \(p. 875\)](#)
- [Interactive sessions API \(p. 875\)](#)
 - [Data Types \(p. 875\)](#)

- [Session Structure \(p. 876\)](#)
 - [SessionCommand Structure \(p. 877\)](#)
 - [Statement Structure \(p. 877\)](#)
 - [StatementOutput Structure \(p. 878\)](#)
 - [StatementOutputData Structure \(p. 878\)](#)
 - [Operations \(p. 878\)](#)
 - [CreateSession Action \(Python: create_session\) \(p. 879\)](#)
 - [StopSession Action \(Python: stop_session\) \(p. 881\)](#)
 - [DeleteSession Action \(Python: delete_session\) \(p. 881\)](#)
 - [GetSession Action \(Python: get_session\) \(p. 882\)](#)
 - [ListSessions Action \(Python: list_sessions\) \(p. 882\)](#)
 - [RunStatement Action \(Python: run_statement\) \(p. 883\)](#)
 - [CancelStatement Action \(Python: cancel_statement\) \(p. 884\)](#)
 - [GetStatement Action \(Python: get_statement\) \(p. 884\)](#)
 - [ListStatements Action \(Python: list_statements\) \(p. 885\)](#)
 - [Development Endpoints API \(p. 886\)](#)
 - [Data Types \(p. 886\)](#)
 - [DevEndpoint Structure \(p. 886\)](#)
 - [DevEndpointCustomLibraries Structure \(p. 888\)](#)
 - [Operations \(p. 889\)](#)
 - [CreateDevEndpoint Action \(Python: create_dev_endpoint\) \(p. 889\)](#)
 - [UpdateDevEndpoint Action \(Python: update_dev_endpoint\) \(p. 893\)](#)
 - [DeleteDevEndpoint Action \(Python: delete_dev_endpoint\) \(p. 894\)](#)
 - [GetDevEndpoint Action \(Python: get_dev_endpoint\) \(p. 894\)](#)
 - [GetDevEndpoints Action \(Python: get_dev_endpoints\) \(p. 895\)](#)
 - [BatchGetDevEndpoints Action \(Python: batch_get_dev_endpoints\) \(p. 895\)](#)
 - [ListDevEndpoints Action \(Python: list_dev_endpoints\) \(p. 896\)](#)
 - [Schema Registry \(p. 897\)](#)
 - [Data Types \(p. 897\)](#)
 - [RegistryId Structure \(p. 897\)](#)
 - [RegistryListItem Structure \(p. 897\)](#)
 - [MetadataInfo Structure \(p. 898\)](#)
 - [OtherMetadataValueListItem Structure \(p. 898\)](#)
 - [SchemaListItem Structure \(p. 899\)](#)
 - [SchemaVersionListItem Structure \(p. 899\)](#)
 - [MetadataKeyValuePair Structure \(p. 900\)](#)
 - [SchemaVersionErrorItem Structure \(p. 900\)](#)
 - [ErrorDetails Structure \(p. 900\)](#)
 - [SchemaVersionNumber Structure \(p. 900\)](#)
 - [Schemald Structure \(p. 901\)](#)
 - [Operations \(p. 901\)](#)
 - [CreateRegistry Action \(Python: create_registry\) \(p. 902\)](#)
 - [CreateSchema Action \(Python: create_schema\) \(p. 903\)
708](#)
 - [GetSchema Action \(Python: get_schema\) \(p. 905\)](#)
 - [ListSchemaVersions Action \(Python: list_schema_versions\) \(p. 907\)](#)
-

- [GetSchemaVersion Action \(Python: get_schema_version\) \(p. 907\)](#)
- [GetSchemaVersionsDiff Action \(Python: get_schema_versions_diff\) \(p. 908\)](#)
- [ListRegistries Action \(Python: list_registries\) \(p. 909\)](#)
- [ListSchemas Action \(Python: list_schemas\) \(p. 910\)](#)
- [RegisterSchemaVersion Action \(Python: register_schema_version\) \(p. 911\)](#)
- [UpdateSchema Action \(Python: update_schema\) \(p. 912\)](#)
- [CheckSchemaVersionValidity Action \(Python: check_schema_version_validity\) \(p. 913\)](#)
- [UpdateRegistry Action \(Python: update_registry\) \(p. 913\)](#)
- [GetSchemaByDefinition Action \(Python: get_schema_by_definition\) \(p. 914\)](#)
- [GetRegistry Action \(Python: get_registry\) \(p. 915\)](#)
- [PutSchemaVersionMetadata Action \(Python: put_schema_version_metadata\) \(p. 916\)](#)
- [QuerySchemaVersionMetadata Action \(Python: query_schema_version_metadata\) \(p. 917\)](#)
- [RemoveSchemaVersionMetadata Action \(Python: remove_schema_version_metadata\) \(p. 918\)](#)
- [DeleteRegistry Action \(Python: delete_registry\) \(p. 919\)](#)
- [DeleteSchema Action \(Python: delete_schema\) \(p. 920\)](#)
- [DeleteSchemaVersions Action \(Python: delete_schema_versions\) \(p. 920\)](#)
- [Workflows \(p. 921\)](#)
 - [Data Types \(p. 921\)](#)
 - [JobNodeDetails Structure \(p. 922\)](#)
 - [CrawlerNodeDetails Structure \(p. 922\)](#)
 - [TriggerNodeDetails Structure \(p. 922\)](#)
 - [Crawl Structure \(p. 922\)](#)
 - [Node Structure \(p. 923\)](#)
 - [Edge Structure \(p. 923\)](#)
 - [Workflow Structure \(p. 924\)](#)
 - [WorkflowGraph Structure \(p. 924\)](#)
 - [WorkflowRun Structure \(p. 925\)](#)
 - [WorkflowRunStatistics Structure \(p. 926\)](#)
 - [StartingEventBatchCondition Structure \(p. 926\)](#)
 - [Blueprint Structure \(p. 926\)](#)
 - [BlueprintDetails Structure \(p. 927\)](#)
 - [LastActiveDefinition Structure \(p. 927\)](#)
 - [BlueprintRun Structure \(p. 928\)](#)
 - [Operations \(p. 929\)](#)
 - [CreateWorkflow Action \(Python: create_workflow\) \(p. 929\)](#)
 - [UpdateWorkflow Action \(Python: update_workflow\) \(p. 930\)](#)
 - [DeleteWorkflow Action \(Python: delete_workflow\) \(p. 931\)](#)
 - [GetWorkflow Action \(Python: get_workflow\) \(p. 932\)](#)
 - [ListWorkflows Action \(Python: list_workflows\) \(p. 932\)](#)
 - [BatchGetWorkflows Action \(Python: batch_get_workflows\) \(p. 933\)](#)
 - [GetWorkflowRun Action \(Python: get_workflow_run\) \(p. 933\)](#)
 - [GetWorkflowRuns Action \(Python: get_workflow_runs\) \(p. 934\)](#)
 - [GetWorkflowRunProperties Action \(Python: get_workflow_run_properties\) \(p. 935\)
709](#)
 - [PutWorkflowRunProperties Action \(Python: put_workflow_run_properties\) \(p. 935\)](#)
 - [CreateBlueprint Action \(Python: create_blueprint\) \(p. 936\)](#)

- [UpdateBlueprint Action \(Python: update_blueprint\) \(p. 937\)](#)
- [DeleteBlueprint Action \(Python: delete_blueprint\) \(p. 937\)](#)
- [ListBlueprints Action \(Python: list_blueprints\) \(p. 938\)](#)
- [BatchGetBlueprints Action \(Python: batch_get_blueprints\) \(p. 938\)](#)
- [StartBlueprintRun Action \(Python: start_blueprint_run\) \(p. 939\)](#)
- [GetBlueprintRun Action \(Python: get_blueprint_run\) \(p. 940\)](#)
- [GetBlueprintRuns Action \(Python: get_blueprint_runs\) \(p. 940\)](#)
- [StartWorkflowRun Action \(Python: start_workflow_run\) \(p. 941\)](#)
- [StopWorkflowRun Action \(Python: stop_workflow_run\) \(p. 942\)](#)
- [ResumeWorkflowRun Action \(Python: resume_workflow_run\) \(p. 942\)](#)
- [Machine Learning API \(p. 943\)](#)
 - [Data Types \(p. 943\)](#)
 - [TransformParameters Structure \(p. 944\)](#)
 - [EvaluationMetrics Structure \(p. 944\)](#)
 - [MLTransform Structure \(p. 944\)](#)
 - [FindMatchesParameters Structure \(p. 946\)](#)
 - [FindMatchesMetrics Structure \(p. 947\)](#)
 - [ConfusionMatrix Structure \(p. 948\)](#)
 - [GlueTable Structure \(p. 948\)](#)
 - [TaskRun Structure \(p. 949\)](#)
 - [TransformFilterCriteria Structure \(p. 949\)](#)
 - [TransformSortCriteria Structure \(p. 950\)](#)
 - [TaskRunFilterCriteria Structure \(p. 950\)](#)
 - [TaskRunSortCriteria Structure \(p. 951\)](#)
 - [TaskRunProperties Structure \(p. 951\)](#)
 - [FindMatchesTaskRunProperties Structure \(p. 951\)](#)
 - [ImportLabelsTaskRunProperties Structure \(p. 952\)](#)
 - [ExportLabelsTaskRunProperties Structure \(p. 952\)](#)
 - [LabelingSetGenerationTaskRunProperties Structure \(p. 952\)](#)
 - [SchemaColumn Structure \(p. 952\)](#)
 - [TransformEncryption Structure \(p. 953\)](#)
 - [MLUserDataEncryption Structure \(p. 953\)](#)
 - [ColumnImportance Structure \(p. 953\)](#)
 - [Operations \(p. 954\)](#)
 - [CreateMLTransform Action \(Python: create_ml_transform\) \(p. 954\)](#)
 - [UpdateMLTransform Action \(Python: update_ml_transform\) \(p. 957\)](#)
 - [DeleteMLTransform Action \(Python: delete_ml_transform\) \(p. 958\)](#)
 - [GetMLTransform Action \(Python: get_ml_transform\) \(p. 959\)](#)
 - [GetMLTransforms Action \(Python: get_ml_transforms\) \(p. 961\)](#)
 - [ListMLTransforms Action \(Python: list_ml_transforms\) \(p. 961\)](#)
 - [StartMLEvaluationTaskRun Action \(Python: start_ml_evaluation_task_run\) \(p. 962\)](#)
 - [StartMLLabelingSetGenerationTaskRun Action \(Python: start_ml_labeling_set_generation_task_run\) \(p. 963\)](#)
 - [GetMLTaskRun Action \(Python: get_ml_task_run\) \(p. 964\)](#)
 - [GetMLTaskRuns Action \(Python: get_ml_task_runs\) \(p. 965\)](#)
 - [CancelMLTaskRun Action \(Python: cancel_ml_task_run\) \(p. 966\)](#)

- [StartExportLabelsTaskRun Action \(Python: start_export_labels_task_run\) \(p. 966\)](#)
- [StartImportLabelsTaskRun Action \(Python: start_import_labels_task_run\) \(p. 967\)](#)
- [Sensitive Data Detection API \(p. 968\)](#)
 - [Data Types \(p. 968\)](#)
 - [CustomEntityType Structure \(p. 968\)](#)
 - [Operations \(p. 969\)](#)
 - [CreateCustomEntityType Action \(Python: create_custom_entity_type\) \(p. 969\)](#)
 - [DeleteCustomEntityType Action \(Python: delete_custom_entity_type\) \(p. 970\)](#)
 - [GetCustomEntityType Action \(Python: get_custom_entity_type\) \(p. 970\)](#)
 - [BatchGetCustomEntityTypes Action \(Python: batch_get_custom_entity_types\) \(p. 971\)](#)
 - [ListCustomEntityTypes Action \(Python: list_custom_entity_types\) \(p. 972\)](#)
- [Tagging APIs in AWS Glue \(p. 972\)](#)
 - [Data Types \(p. 972\)](#)
 - [Tag Structure \(p. 972\)](#)
 - [Operations \(p. 973\)](#)
 - [TagResource Action \(Python: tag_resource\) \(p. 973\)](#)
 - [UntagResource Action \(Python: untag_resource\) \(p. 973\)](#)
 - [GetTags Action \(Python: get_tags\) \(p. 974\)](#)
- [Common Data Types \(p. 974\)](#)
 - [Tag Structure \(p. 974\)](#)
 - [DecimalNumber Structure \(p. 975\)](#)
 - [ErrorDetail Structure \(p. 975\)](#)
 - [PropertyPredicate Structure \(p. 975\)](#)
 - [ResourceUri Structure \(p. 976\)](#)
 - [ColumnStatistics Structure \(p. 976\)](#)
 - [ColumnStatisticsError Structure \(p. 976\)](#)
 - [ColumnError Structure \(p. 977\)](#)
 - [ColumnStatisticsData Structure \(p. 977\)](#)
 - [BooleanColumnStatisticsData Structure \(p. 977\)](#)
 - [DateColumnStatisticsData Structure \(p. 978\)](#)
 - [DecimalColumnStatisticsData Structure \(p. 978\)](#)
 - [DoubleColumnStatisticsData Structure \(p. 978\)](#)
 - [LongColumnStatisticsData Structure \(p. 979\)](#)
 - [StringColumnStatisticsData Structure \(p. 979\)](#)
 - [BinaryColumnStatisticsData Structure \(p. 979\)](#)
 - [String Patterns \(p. 980\)](#)
- [Exceptions \(p. 981\)](#)
 - [AccessDeniedException Structure \(p. 981\)](#)
 - [AlreadyExistsException Structure \(p. 981\)](#)
 - [ConcurrentModificationException Structure \(p. 981\)](#)
 - [ConcurrentRunsExceededException Structure \(p. 981\)](#)
 - [CrawlerNotRunningException Structure \(p. 982\)](#)
 - [CrawlerRunningException Structure \(p. 982\)](#)
 - [CrawlerStoppingException Structure \(p. 982\)](#)
 - [EntityNotFoundException Structure \(p. 982\)](#)

- [GlueEncryptionException Structure \(p. 982\)](#)
- [IdempotentParameterMismatchException Structure \(p. 983\)](#)
- [IllegalWorkflowStateException Structure \(p. 983\)](#)
- [InternalServiceException Structure \(p. 983\)](#)
- [InvalidExecutionEngineException Structure \(p. 983\)](#)
- [InvalidInputException Structure \(p. 983\)](#)
- [InvalidStateException Structure \(p. 984\)](#)
- [InvalidTaskStatusTransitionException Structure \(p. 984\)](#)
- [JobDefinitionErrorException Structure \(p. 984\)](#)
- [JobRunInTerminalStateException Structure \(p. 984\)](#)
- [JobRunInvalidStateTransitionException Structure \(p. 984\)](#)
- [JobRunNotInTerminalStateException Structure \(p. 985\)](#)
- [LateRunnerException Structure \(p. 985\)](#)
- [NoScheduleException Structure \(p. 985\)](#)
- [OperationTimeoutException Structure \(p. 985\)](#)
- [ResourceNotReadyException Structure \(p. 985\)](#)
- [ResourceNumberLimitExceededException Structure \(p. 986\)](#)
- [SchedulerNotRunningException Structure \(p. 986\)](#)
- [SchedulerRunningException Structure \(p. 986\)](#)
- [SchedulerTransitioningException Structure \(p. 986\)](#)
- [UnrecognizedRunnerException Structure \(p. 986\)](#)
- [ValidationException Structure \(p. 987\)](#)
- [VersionMismatchException Structure \(p. 987\)](#)

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 \(p. 712\)](#)
- [EncryptionAtRest Structure \(p. 713\)](#)
- [ConnectionPasswordEncryption Structure \(p. 713\)](#)
- [EncryptionConfiguration Structure \(p. 714\)](#)
- [S3Encryption Structure \(p. 714\)](#)
- [CloudWatchEncryption Structure \(p. 714\)](#)
- [JobBookmarksEncryption Structure \(p. 714\)](#)
- [SecurityConfiguration Structure \(p. 715\)](#)
- [GluePolicy Structure \(p. 715\)](#)

DataCatalogEncryptionSettings Structure

Contains configuration information for maintaining Data Catalog security.

Fields

- **EncryptionAtRest** – An [EncryptionAtRest \(p. 713\)](#) object.
Specifies the encryption-at-rest configuration for the Data Catalog.
- **ConnectionPasswordEncryption** – A [ConnectionPasswordEncryption \(p. 713\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).

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 \(p. 714\)](#) objects.
The encryption configuration for Amazon Simple Storage Service (Amazon S3) data.
- **CloudWatchEncryption** – A [CloudWatchEncryption \(p. 714\)](#) object.
The encryption configuration for Amazon CloudWatch.
- **JobBookmarksEncryption** – A [JobBookmarksEncryption \(p. 714\)](#) 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="SSSES3").
The encryption mode to use for Amazon S3 data.
- **KmsKeyArn** – UTF-8 string, matching the [Custom string pattern #19 \(p. 980\)](#).
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 #19 \(p. 980\)](#).
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 #19 \(p. 980\)](#).
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 \(p. 980\)](#).
The name of the security configuration.
- **CreatedTimeStamp** – Timestamp.
The time at which this security configuration was created.
- **EncryptionConfiguration** – An [EncryptionConfiguration \(p. 714\)](#) object.
The encryption configuration associated with this security configuration.

GluePolicy Structure

A structure for returning a resource policy.

Fields

- **PolicyInJson** – UTF-8 string, at least 2 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 \(p. 980\)](#).
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.

Operations

- [GetDataCatalogEncryptionSettings Action \(Python: get_data_catalog_encryption_settings\) \(p. 716\)](#)
- [PutDataCatalogEncryptionSettings Action \(Python: put_data_catalog_encryption_settings\) \(p. 716\)](#)
- [PutResourcePolicy Action \(Python: put_resource_policy\) \(p. 717\)](#)

- [GetResourcePolicy Action \(Python: get_resource_policy\) \(p. 718\)](#)
- [DeleteResourcePolicy Action \(Python: delete_resource_policy\) \(p. 718\)](#)
- [CreateSecurityConfiguration Action \(Python: create_security_configuration\) \(p. 719\)](#)
- [DeleteSecurityConfiguration Action \(Python: delete_security_configuration\) \(p. 720\)](#)
- [GetSecurityConfiguration Action \(Python: get_security_configuration\) \(p. 720\)](#)
- [GetSecurityConfigurations Action \(Python: get_security_configurations\) \(p. 721\)](#)
- [GetResourcePolicies Action \(Python: get_resource_policies\) \(p. 721\)](#)

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 \(p. 980\)](#).

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 \(p. 712\)](#) 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 \(p. 980\)](#).

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 \(p. 712\)](#) 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, at least 2 bytes long.
Contains the policy document to set, in JSON format.
- ResourceArn – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).
Do not use. For internal use only.
- PolicyHashCondition – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 does not depend on the existence of a policy.

- EnableHybrid – UTF-8 string (valid values: TRUE | FALSE).

If 'TRUE', indicates that you are using both methods to grant cross-account access to Data Catalog resources:

- By directly updating the resource policy with PutResourcePolicy
- By using the **Grant permissions** command on the AWS Management Console.

Must be set to 'TRUE' if you have already used the Management Console to grant cross-account access, otherwise the call fails. Default is 'FALSE'.

Response

- PolicyHash – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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

- `ResourceArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The ARN of the AWS Glue resource for which to retrieve the resource policy. If not supplied, the Data Catalog resource policy is returned. Use `GetResourcePolicies` to view all existing resource policies. For more information see [Specifying AWS Glue Resource ARNs](#).

Response

- `PolicyInJson` – UTF-8 string, at least 2 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 \(p. 980\)](#).
 - 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`
- `InvalidInputException`

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 \(p. 980\)](#).

The hash value returned when this policy was set.

- **ResourceArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The ARN of the AWS Glue resource for the resource policy to be deleted.

Response

- *No Response parameters.*

Errors

- [EntityNotFoundException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [InvalidInputException](#)
- [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 \(p. 980\)](#).

The name for the new security configuration.

- **EncryptionConfiguration** – *Required:* An [EncryptionConfiguration \(p. 714\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the security configuration to retrieve.

Response

- `SecurityConfiguration` – A [SecurityConfiguration \(p. 715\)](#) 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 \(p. 715\)](#) 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`

GetResourcePolicies Action (Python: `get_resource_policies`)

Retrieves the resource policies set on individual resources by AWS Resource Access Manager during cross-account permission grants. Also retrieves the Data Catalog resource policy.

If you enabled metadata encryption in Data Catalog settings, and you do not have permission on the AWS KMS key, the operation can't return the Data Catalog resource policy.

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

- **GetResourcePoliciesResponseList** – An array of [GluePolicy \(p. 715\)](#) objects.
A list of the individual resource policies and the account-level resource policy.
- **NextToken** – UTF-8 string.
A continuation token, if the returned list does not contain the last resource policy available.

Errors

- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [InvalidInputException](#)
- [GlueEncryptionException](#)

Catalog API

The Catalog API describes the data types and API related to working with catalogs in AWS Glue.

Topics

- [Database API \(p. 722\)](#)
- [Table API \(p. 728\)](#)
- [Partition API \(p. 752\)](#)
- [Connection API \(p. 767\)](#)
- [User-Defined Function API \(p. 776\)](#)
- [Importing an Athena Catalog to AWS Glue \(p. 781\)](#)

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 \(p. 722\)](#)
- [DatabaseInput Structure \(p. 723\)](#)
- [PrincipalPermissions Structure \(p. 724\)](#)
- [DataLakePrincipal Structure \(p. 724\)](#)
- [DatabaseIdentifier Structure \(p. 724\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 724\)](#) objects.

Creates a set of default permissions on the table for principals.

- **TargetDatabase** – A [DatabaseIdentifier \(p. 724\)](#) object.

A `DatabaseIdentifier` structure that describes a target database for resource linking.

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the Data Catalog in which the database resides.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 724\)](#) objects.

Creates a set of default permissions on the table for principals.

- `TargetDatabase` – A [DatabaseIdentifier \(p. 724\)](#) object.

A `DatabaseIdentifier` structure that describes a target database for resource linking.

PrincipalPermissions Structure

Permissions granted to a principal.

Fields

- `Principal` – A [DataLakePrincipal \(p. 724\)](#) 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.

DatabaseIdentifier Structure

A structure that describes a target database for resource linking.

Fields

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the Data Catalog in which the database resides.

- `DatabaseName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the catalog database.

Operations

- [CreateDatabase Action \(Python: create_database\) \(p. 725\)](#)
- [UpdateDatabase Action \(Python: update_database\) \(p. 725\)](#)
- [DeleteDatabase Action \(Python: delete_database\) \(p. 726\)](#)
- [GetDatabase Action \(Python: get_database\) \(p. 727\)](#)
- [GetDatabases Action \(Python: get_databases\) \(p. 727\)](#)

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 \(p. 980\)](#).

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 \(p. 723\)](#) object.

The metadata for the database.

Response

- *No Response parameters.*

Errors

- [InvalidInputException](#)
- [AlreadyExistsException](#)
- [ResourceNumberLimitExceededException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)
- [ConcurrentModificationException](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the database to update in the catalog. For Hive compatibility, this is folded to lowercase.

- **DatabaseInput** – *Required:* A [DatabaseInput \(p. 723\)](#) 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`
- `ConcurrentModificationException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the database to delete. For Hive compatibility, this must be all lowercase.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the database to retrieve. For Hive compatibility, this should be all lowercase.

Response

- `Database` – A [Database \(p. 722\)](#) 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 \(p. 980\)](#).

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 100.

The maximum number of databases to return in one response.

- **ResourceShareType** – UTF-8 string (valid values: FOREIGN | ALL).

Allows you to specify that you want to list the databases shared with your account. The allowable values are FOREIGN or ALL.

- If set to FOREIGN, will list the databases shared with your account.
- If set to ALL, will list the databases shared with your account, as well as the databases in your local account.

Response

- **DatabaseList** – *Required:* An array of [Database \(p. 722\)](#) 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

- [InvalidArgumentException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)

Table API

The Table API describes data types and operations associated with tables.

Data Types

- [Table Structure \(p. 729\)](#)
- [TableInput Structure \(p. 730\)](#)
- [Column Structure \(p. 731\)](#)
- [StorageDescriptor Structure \(p. 732\)](#)
- [SchemaReference Structure \(p. 733\)](#)
- [SerDeInfo Structure \(p. 733\)](#)
- [Order Structure \(p. 734\)](#)
- [SkewedInfo Structure \(p. 734\)](#)
- [TableVersion Structure \(p. 734\)](#)
- [TableError Structure \(p. 734\)](#)
- [TableVersionError Structure \(p. 735\)](#)
- [SortCriterion Structure \(p. 735\)](#)
- [TableIdentifier Structure \(p. 735\)](#)
- [KeySchemaElement Structure \(p. 736\)](#)
- [PartitionIndex Structure \(p. 736\)](#)
- [PartitionIndexDescriptor Structure \(p. 736\)](#)

- [BackfillError Structure \(p. 737\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 732\)](#) object.

A storage descriptor containing information about the physical storage of this table.

- **PartitionKeys** – An array of [Column \(p. 731\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).
The person or entity who created the table.
- **IsRegisteredWithLakeFormation** – Boolean.
Indicates whether the table has been registered with AWS Lake Formation.
- **TargetTable** – A [TableIdentifier \(p. 735\)](#) object.
A `TableIdentifier` structure that describes a target table for resource linking.
- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the Data Catalog in which the table resides.
- **VersionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the table version.

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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 732\)](#) object.

A storage descriptor containing information about the physical storage of this table.

- **PartitionKeys** – An array of [Column \(p. 731\)](#) 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 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

These key-value pairs define properties associated with the table.

- **TargetTable** – A [TableIdentifier \(p. 735\)](#) object.

A `TableIdentifier` structure that describes a target table for resource linking.

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 \(p. 980\)](#).

The name of the Column.

- **Type** – UTF-8 string, not more than 131072 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The data type of the Column.

- **Comment** – Comment string, not more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 731\)](#) 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 \(p. 980\)](#).

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.

- **AdditionalLocations** – An array of UTF-8 strings.

A list of locations that point to the path where a Delta table is located.

- **InputFormat** – Format string, not more than 128 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 733\)](#) 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 \(p. 734\)](#) 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 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 512000 bytes long.

The user-supplied properties in key-value form.

- **SkewedInfo** – A [SkewedInfo \(p. 734\)](#) 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.

- **SchemaReference** – A [SchemaReference \(p. 733\)](#) object.

An object that references a schema stored in the AWS Glue Schema Registry.

When creating a table, you can pass an empty list of columns for the schema, and instead use a schema reference.

SchemaReference Structure

An object that references a schema stored in the AWS Glue Schema Registry.

Fields

- **SchemaId** – A [SchemaId \(p. 901\)](#) object.

A structure that contains schema identity fields. Either this or the `SchemaVersionId` has to be provided.

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The unique ID assigned to a version of the schema. Either this or the `SchemaId` has to be provided.

- **SchemaVersionNumber** – Number (long), not less than 1 or more than 100000.

The version number of the schema.

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 \(p. 980\)](#).

Name of the SerDe.

- **SerializationLibrary** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 729\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the table. For Hive compatibility, this must be entirely lowercase.

- **ErrorDetail** – An [ErrorDetail \(p. 975\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 975\)](#) 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.

TableIdentifier Structure

A structure that describes a target table for resource linking.

Fields

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the Data Catalog in which the table resides.

- **DatabaseName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the catalog database that contains the target table.

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the target table.

KeySchemaElement Structure

A partition key pair consisting of a name and a type.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of a partition key.

- **Type** – *Required*: UTF-8 string, not more than 131072 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The type of a partition key.

PartitionIndex Structure

A structure for a partition index.

Fields

- **Keys** – *Required*: An array of UTF-8 strings, at least 1 string.

The keys for the partition index.

- **IndexName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the partition index.

PartitionIndexDescriptor Structure

A descriptor for a partition index in a table.

Fields

- **IndexName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the partition index.

- **Keys** – *Required*: An array of [KeySchemaElement \(p. 736\)](#) objects, at least 1 structure.

A list of one or more keys, as `KeySchemaElement` structures, for the partition index.

- **IndexStatus** – *Required*: UTF-8 string (valid values: CREATING | ACTIVE | DELETING | FAILED).

The status of the partition index.

The possible statuses are:

- **CREATING:** The index is being created. When an index is in a CREATING state, the index or its table cannot be deleted.
- **ACTIVE:** The index creation succeeds.
- **FAILED:** The index creation fails.
- **DELETING:** The index is deleted from the list of indexes.
- **BackfillErrors** – An array of [BackfillError \(p. 737\)](#) objects.

A list of errors that can occur when registering partition indexes for an existing table.

BackfillError Structure

A list of errors that can occur when registering partition indexes for an existing table.

These errors give the details about why an index registration failed and provide a limited number of partitions in the response, so that you can fix the partitions at fault and try registering the index again. The most common set of errors that can occur are categorized as follows:

- **EncryptedPartitionError:** The partitions are encrypted.
- **InvalidPartitionTypeDataError:** The partition value doesn't match the data type for that partition column.
- **MissingPartitionValueError:** The partitions are encrypted.
- **UnsupportedPartitionCharacterError:** Characters inside the partition value are not supported. For example: U+0000 , U+0001, U+0002.
- **InternalError:** Any error which does not belong to other error codes.

Fields

- **Code** – UTF-8 string (valid values: ENCRYPTED_PARTITION_ERROR | INTERNAL_ERROR | INVALID_PARTITION_TYPE_DATA_ERROR | MISSING_PARTITION_VALUE_ERROR | UNSUPPORTED_PARTITION_CHARACTER_ERROR).

The error code for an error that occurred when registering partition indexes for an existing table.

- **Partitions** – An array of [PartitionValueList \(p. 754\)](#) objects.

A list of a limited number of partitions in the response.

Operations

- [CreateTable Action \(Python: create_table\) \(p. 738\)](#)
- [UpdateTable Action \(Python: update_table\) \(p. 739\)](#)
- [DeleteTable Action \(Python: delete_table\) \(p. 739\)](#)
- [BatchDeleteTable Action \(Python: batch_delete_table\) \(p. 740\)](#)
- [GetTable Action \(Python: get_table\) \(p. 741\)](#)
- [GetTables Action \(Python: get_tables\) \(p. 742\)](#)
- [GetTableVersion Action \(Python: get_table_version\) \(p. 743\)](#)
- [GetTableVersions Action \(Python: get_table_versions\) \(p. 744\)](#)
- [DeleteTableVersion Action \(Python: delete_table_version\) \(p. 745\)](#)
- [BatchDeleteTableVersion Action \(Python: batch_delete_table_version\) \(p. 745\)](#)

- [SearchTables Action \(Python: search_tables\) \(p. 746\)](#)
- [GetPartitionIndexes Action \(Python: get_partition_indexes\) \(p. 747\)](#)
- [CreatePartitionIndex Action \(Python: create_partition_index\) \(p. 748\)](#)
- [DeletePartitionIndex Action \(Python: delete_partition_index\) \(p. 749\)](#)
- [GetColumnStatisticsForTable Action \(Python: get_column_statistics_for_table\) \(p. 749\)](#)
- [UpdateColumnStatisticsForTable Action \(Python: update_column_statistics_for_table\) \(p. 750\)](#)
- [DeleteColumnStatisticsForTable Action \(Python: delete_column_statistics_for_table\) \(p. 751\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The catalog database in which to create the new table. For Hive compatibility, this name is entirely lowercase.

- **TableInput** – *Required:* A [TableInput \(p. 730\)](#) object.

The TableInput object that defines the metadata table to create in the catalog.

- **PartitionIndexes** – An array of [PartitionIndex \(p. 736\)](#) objects, not more than 3 structures.

A list of partition indexes, PartitionIndex structures, to create in the table.

- **TransactionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The ID of the transaction.

Response

- *No Response parameters.*

Errors

- [AlreadyExistsException](#)
- [InvalidInputException](#)
- [EntityNotFoundException](#)
- [ResourceNumberLimitExceededException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)
- [ConcurrentModificationException](#)

- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the catalog database in which the table resides. For Hive compatibility, this name is entirely lowercase.

- `TableInput` – *Required:* A [TableInput \(p. 730\)](#) 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.

- `TransactionId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to update the table contents.

- `VersionId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The version ID at which to update the table contents.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`
- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the table to be deleted. For Hive compatibility, this name is entirely lowercase.

- `TransactionId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to delete the table contents.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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.

- **TransactionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to delete the table contents.

Response

- **Errors** – An array of [TableError \(p. 734\)](#) objects.

A list of errors encountered in attempting to delete the specified tables.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the table for which to retrieve the definition. For Hive compatibility, this name is entirely lowercase.

- `TransactionId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to read the table contents.

- `QueryAsOfTime` – Timestamp.

The time as of when to read the table contents. If not set, the most recent transaction commit time will be used. Cannot be specified along with `TransactionId`.

Response

- `Table` – A [Table \(p. 729\)](#) object.

The `Table` object that defines the specified table.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 100.

The maximum number of tables to return in a single response.

- **TransactionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to read the table contents.

- **QueryAsOfTime** – Timestamp.

The time as of when to read the table contents. If not set, the most recent transaction commit time will be used. Cannot be specified along with `TransactionId`.

Response

- **TableList** – An array of [Table \(p. 729\)](#) 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`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 734\)](#) object.

The requested table version.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 100.

The maximum number of table versions to return in one response.

Response

- **TableVersions** – An array of [TableVersion \(p. 734\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 735\)](#) 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 \(p. 980\)](#).

A unique identifier, consisting of `account_id`.

- **NextToken** – UTF-8 string.

A continuation token, included if this is a continuation call.

- **Filters** – An array of [PropertyPredicate \(p. 975\)](#) objects.

A list of key-value pairs, and a comparator used to filter the search results. Returns all entities matching the predicate.

The `Comparator` member of the `PropertyPredicate` struct is used only for time fields, and can be omitted for other field types. Also, when comparing string values, such as when `Key=Name`, a fuzzy match algorithm is used. The `Key` field (for example, the value of the `Name` field) is split on certain punctuation characters, for example, `,`, `:`, `#`, etc. into tokens. Then each token is exact-match compared with the `Value` member of `PropertyPredicate`. For example, if `Key=Name` and `Value=link`, tables named `customer-link` and `xx-link-yy` are returned, but `xxlinkyy` is not returned.

- `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 \(p. 735\)](#) 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.

- `ResourceShareType` – UTF-8 string (valid values: `FOREIGN` | `ALL`).

Allows you to specify that you want to search the tables shared with your account. The allowable values are `FOREIGN` or `ALL`.

- If set to `FOREIGN`, will search the tables shared with your account.
- If set to `ALL`, will search the tables shared with your account, as well as the tables in your local account.

Response

- `NextToken` – UTF-8 string.

A continuation token, present if the current list segment is not the last.

- `TableList` – An array of [Table \(p. 729\)](#) objects.

A list of the requested `Table` objects. The `SearchTables` response returns only the tables that you have access to.

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

GetPartitionIndexes Action ([Python: get_partition_indexes](#))

Retrieves the partition indexes associated with a table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The catalog ID where the table resides.

- `DatabaseName` – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a database from which you want to retrieve partition indexes.

- **TableName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a table for which you want to retrieve the partition indexes.

- **NextToken** – UTF-8 string.

A continuation token, included if this is a continuation call.

Response

- **PartitionIndexDescriptorList** – An array of [PartitionIndexDescriptor \(p. 736\)](#) objects.

A list of index descriptors.

- **NextToken** – UTF-8 string.

A continuation token, present if the current list segment is not the last.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`

CreatePartitionIndex Action (Python: `create_partition_index`)

Creates a specified partition index in an existing table.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The catalog ID where the table resides.

- **DatabaseName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a database in which you want to create a partition index.

- **TableName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a table in which you want to create a partition index.

- **PartitionIndex** – *Required:* A [PartitionIndex \(p. 736\)](#) object.

Specifies a `PartitionIndex` structure to create a partition index in an existing table.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartitionIndex Action (Python: `delete_partition_index`)

Deletes a specified partition index from an existing table.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The catalog ID where the table resides.

- `DatabaseName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a database from which you want to delete a partition index.

- `TableName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of a table from which you want to delete a partition index.

- `IndexName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the partition index to be deleted.

Response

- *No Response parameters.*

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`
- `GlueEncryptionException`

GetColumnStatisticsForTable Action (Python: `get_column_statistics_for_table`)

Retrieves table statistics of columns.

The Identity and Access Management (IAM) permission required for this operation is `GetTable`.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- `ColumnNames` – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the column names.

Response

- `ColumnStatisticsList` – An array of [ColumnStatistics \(p. 976\)](#) objects.

List of ColumnStatistics that failed to be retrieved.

- `Errors` – An array of [ColumnError \(p. 977\)](#) objects.

List of ColumnStatistics that failed to be retrieved.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForTable Action (Python: `update_column_statistics_for_table`)

Creates or updates table statistics of columns.

The Identity and Access Management (IAM) permission required for this operation is `UpdateTable`.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- **ColumnStatisticsList** – *Required*: An array of [ColumnStatistics \(p. 976\)](#) objects, not more than 25 structures.

A list of the column statistics.

Response

- **Errors** – An array of [ColumnStatisticsError \(p. 976\)](#) objects.

List of ColumnStatisticsErrors.

Errors

- [EntityNotFoundException](#)
- [InvalidInputException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [GlueEncryptionException](#)

DeleteColumnStatisticsForTable Action (Python: `delete_column_statistics_for_table`)

Retrieves table statistics of columns.

The Identity and Access Management (IAM) permission required for this operation is `DeleteTable`.

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- **ColumnName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the column.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

Partition API

The Partition API describes data types and operations used to work with partitions.

Data Types

- [Partition Structure \(p. 752\)](#)
- [PartitionInput Structure \(p. 753\)](#)
- [PartitionSpecWithSharedStorageDescriptor Structure \(p. 753\)](#)
- [PartitionListComposingSpec Structure \(p. 754\)](#)
- [PartitionSpecProxy Structure \(p. 754\)](#)
- [PartitionValueList Structure \(p. 754\)](#)
- [Segment Structure \(p. 755\)](#)
- [PartitionError Structure \(p. 755\)](#)
- [BatchUpdatePartitionFailureEntry Structure \(p. 755\)](#)
- [BatchUpdatePartitionRequestEntry Structure \(p. 755\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 732\)](#) 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 \(p. 980\)](#).

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.

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the Data Catalog in which the partition resides.

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 \(p. 732\)](#) 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 \(p. 980\)](#).

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 \(p. 732\)](#) object.
The shared physical storage information.
- `Partitions` – An array of [Partition \(p. 752\)](#) objects.
A list of the partitions that share this physical location.

PartitionListComposingSpec Structure

Lists the related partitions.

Fields

- `Partitions` – An array of [Partition \(p. 752\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).
The root path of the proxy for addressing the partitions.
- `PartitionSpecWithSharedSD` – A [PartitionSpecWithSharedStorageDescriptor \(p. 753\)](#) object.
A specification of partitions that share the same physical storage location.
- `PartitionListComposingSpec` – A [PartitionListComposingSpec \(p. 754\)](#) 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 run 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 \(p. 975\)](#) object.

The details about the partition error.

BatchUpdatePartitionFailureEntry Structure

Contains information about a batch update partition error.

Fields

- `PartitionValueList` – An array of UTF-8 strings, not more than 100 strings.

A list of values defining the partitions.

- `ErrorDetail` – An [ErrorDetail \(p. 975\)](#) object.

The details about the batch update partition error.

BatchUpdatePartitionRequestEntry Structure

A structure that contains the values and structure used to update a partition.

Fields

- `PartitionValueList` – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of values defining the partitions.

- `PartitionInput` – *Required*: A [PartitionInput \(p. 753\)](#) object.

The structure used to update a partition.

Operations

- [CreatePartition Action \(Python: create_partition\) \(p. 756\)](#)
- [BatchCreatePartition Action \(Python: batch_create_partition\) \(p. 757\)](#)
- [UpdatePartition Action \(Python: update_partition\) \(p. 757\)](#)
- [DeletePartition Action \(Python: delete_partition\) \(p. 758\)](#)
- [BatchDeletePartition Action \(Python: batch_delete_partition\) \(p. 759\)](#)
- [GetPartition Action \(Python: get_partition\) \(p. 759\)](#)
- [GetPartitions Action \(Python: get_partitions\) \(p. 760\)](#)
- [BatchGetPartition Action \(Python: batch_get_partition\) \(p. 763\)](#)
- [BatchUpdatePartition Action \(Python: batch_update_partition\) \(p. 764\)](#)
- [GetColumnStatisticsForPartition Action \(Python: get_column_statistics_for_partition\) \(p. 765\)](#)
- [UpdateColumnStatisticsForPartition Action \(Python: update_column_statistics_for_partition\) \(p. 766\)](#)
- [DeleteColumnStatisticsForPartition Action \(Python: delete_column_statistics_for_partition\) \(p. 767\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the metadata table in which the partition is to be created.

- **PartitionInput** – *Required:* A [PartitionInput \(p. 753\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the metadata table in which the partition is to be created.

- **PartitionInputList** – *Required*: An array of [PartitionInput \(p. 753\)](#) objects, not more than 100 structures.

A list of `PartitionInput` structures that define the partitions to be created.

Response

- **Errors** – An array of [PartitionError \(p. 755\)](#) objects.

The errors encountered when trying to create the requested partitions.

Errors

- `InvalidInputException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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.

List of partition key values that define the partition to update.

- **PartitionInput** – *Required*: A [PartitionInput \(p. 753\)](#) object.

The new partition object to update the partition to.

The **Values** property can't be changed. If you want to change the partition key values for a partition, delete and recreate the partition.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the table that contains the partitions to be deleted.

- `PartitionsToDelete` – *Required*: An array of [PartitionValueList \(p. 754\)](#) objects, not more than 25 structures.

A list of `PartitionInput` structures that define the partitions to be deleted.

Response

- `Errors` – An array of [PartitionError \(p. 755\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 752\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- **Expression** – Predicate string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

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 [JSQParser](#) 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 type is encountered that is not valid, 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 \(p. 755\)](#) 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.

- **ExcludeColumnSchema** – Boolean.

When true, specifies not returning the partition column schema. Useful when you are interested only in other partition attributes such as partition values or location. This approach avoids the problem of a large response by not returning duplicate data.

- **TransactionId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #11 \(p. 980\)](#).

The transaction ID at which to read the partition contents.

- **QueryAsOfTime** – Timestamp.

The time as of when to read the partition contents. If not set, the most recent transaction commit time will be used. Cannot be specified along with **TransactionId**.

Response

- **Partitions** – An array of [Partition \(p. 752\)](#) 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`
- `InvalidStateException`
- `ResourceNotReadyException`

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.
- **PartitionsToGet** – *Required*: An array of [PartitionValueList \(p. 754\)](#) objects, not more than 1000 structures.

A list of partition values identifying the partitions to retrieve.

Response

- **Partitions** – An array of [Partition \(p. 752\)](#) objects.

A list of the requested partitions.
- **UnprocessedKeys** – An array of [PartitionValueList \(p. 754\)](#) 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`
- `InvalidStateException`

BatchUpdatePartition Action (Python: batch_update_partition)

Updates 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 \(p. 980\)](#).

The ID of the catalog in which the partition is to be updated. 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 \(p. 980\)](#).

The name of the metadata database in which the partition is to be updated.
- **TableName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the metadata table in which the partition is to be updated.
- **Entries** – *Required*: An array of [BatchUpdatePartitionRequestEntry \(p. 755\)](#) objects, not less than 1 or more than 100 structures.

A list of up to 100 `BatchUpdatePartitionRequestEntry` objects to update.

Response

- **Errors** – An array of [BatchUpdatePartitionFailureEntry \(p. 755\)](#) objects.
The errors encountered when trying to update the requested partitions. A list of [BatchUpdatePartitionFailureEntry](#) objects.

Errors

- [InvalidArgumentException](#)
- [EntityNotFoundException](#)
- [OperationTimeoutException](#)
- [InternalServiceException](#)
- [GlueEncryptionException](#)

[GetColumnStatisticsForPartition Action \(Python: get_column_statistics_for_partition\)](#)

Retrieves partition statistics of columns.

The Identity and Access Management (IAM) permission required for this operation is [GetPartition](#).

Request

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- **PartitionValues** – *Required:* An array of UTF-8 strings.

A list of partition values identifying the partition.

- **ColumnNames** – *Required:* An array of UTF-8 strings, not more than 100 strings.

A list of the column names.

Response

- **ColumnStatisticsList** – An array of [ColumnStatistics \(p. 976\)](#) objects.

List of ColumnStatistics that failed to be retrieved.

- **Errors** – An array of [ColumnError \(p. 977\)](#) objects.

Error occurred during retrieving column statistics data.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForPartition Action (Python: `update_column_statistics_for_partition`)

Creates or updates partition statistics of columns.

The Identity and Access Management (IAM) permission required for this operation is `UpdatePartition`.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- `PartitionValues` – *Required:* An array of UTF-8 strings.

A list of partition values identifying the partition.

- `ColumnStatisticsList` – *Required:* An array of [ColumnStatistics \(p. 976\)](#) objects, not more than 25 structures.

A list of the column statistics.

Response

- `Errors` – An array of [ColumnStatisticsError \(p. 976\)](#) objects.

Error occurred during updating column statistics data.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteColumnStatisticsForPartition Action (Python: delete_column_statistics_for_partition)

Delete the partition column statistics of a column.

The Identity and Access Management (IAM) permission required for this operation is `DeletePartition`.

Request

- `CatalogId` – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the partitions' table.

- `PartitionValues` – *Required*: An array of UTF-8 strings.

A list of partition values identifying the partition.

- `ColumnName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Name of the column.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`
- `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 \(p. 768\)](#)
- [ConnectionInput Structure \(p. 770\)](#)
- [PhysicalConnectionRequirements Structure \(p. 771\)](#)

- [GetConnectionsFilter Structure \(p. 771\)](#)

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 \(p. 980\)](#).

The name of the connection definition.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The description of the connection.

- **ConnectionType** – UTF-8 string (valid values: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM).

The type of the connection. Currently, 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 | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | KAFKA_SASL_MECHANISM | AUTHENTICATION_SECRET_ARN | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME).

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 connecting to a JDBC data source.
- **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`.
- **CONNECTION_URL** - The URL for connecting to a general (non-JDBC) data source.
- **SECRET_ID** - The secret ID used for the secret manager of credentials.
- **CONNECTOR_URL** - The connector URL for a MARKETPLACE or CUSTOM connection.
- **CONNECTOR_TYPE** - The connector type for a MARKETPLACE or CUSTOM connection.
- **CONNECTOR_CLASS_NAME** - The connector class name for a MARKETPLACE or CUSTOM connection.
- **KAFKA_BOOTSTRAP_SERVERS** - A comma-separated list of host and port pairs that are the addresses of the Apache Kafka brokers in a Kafka cluster to which a Kafka client will connect to and bootstrap itself.
- **KAFKA_SSL_ENABLED** - Whether to enable or disable SSL on an Apache Kafka connection. Default value is "true".
- **KAFKA_CUSTOM_CERT** - The Amazon S3 URL for the private CA cert file (.pem format). The default is an empty string.
- **KAFKA_SKIP_CUSTOM_CERT_VALIDATION** - Whether to skip the validation of the CA cert file or not. AWS Glue validates for three algorithms: SHA256withRSA, SHA384withRSA and SHA512withRSA. Default value is "false".
- **KAFKA_CLIENT_KEYSTORE** - The Amazon S3 location of the client keystore file for Kafka client side authentication (Optional).
- **KAFKA_CLIENT_KEYSTORE_PASSWORD** - The password to access the provided keystore (Optional).
- **KAFKA_CLIENT_KEY_PASSWORD** - A keystore can consist of multiple keys, so this is the password to access the client key to be used with the Kafka server side key (Optional).
- **ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD** - The encrypted version of the Kafka client keystore password (if the user has the AWS Glue encrypt passwords setting selected).
- **ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD** - The encrypted version of the Kafka client key password (if the user has the AWS Glue encrypt passwords setting selected).
- **KAFKA_SASL_MECHANISM** - "SCRAM-SHA-512" or "GSSAPI". These are the two supported [SASL Mechanisms](#).
- **KAFKA_SASL_SCRAM_USERNAME** - A plaintext username used to authenticate with the "SCRAM-SHA-512" mechanism.
- **KAFKA_SASL_SCRAM_PASSWORD** - A plaintext password used to authenticate with the "SCRAM-SHA-512" mechanism.

- **ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD** - The encrypted version of the Kafka SASL SCRAM password (if the user has the AWS Glue encrypt passwords setting selected).
- **KAFKA_SASL_GSSAPI_KEYTAB** - The S3 location of a Kerberos keytab file. A keytab stores long-term keys for one or more principals. For more information, see [MIT Kerberos Documentation: Keytab](#).
- **KAFKA_SASL_GSSAPI_KRB5_CONF** - The S3 location of a Kerberos krb5.conf file. A krb5.conf stores Kerberos configuration information, such as the location of the KDC server. For more information, see [MIT Kerberos Documentation: krb5.conf](#).
- **KAFKA_SASL_GSSAPI_SERVICE** - The Kerberos service name, as set with `sasl.kerberos.service.name` in your [Kafka Configuration](#).
- **KAFKA_SASL_GSSAPI_PRINCIPAL** - The name of the Kerberos principal used by AWS Glue. For more information, see [Kafka Documentation: Configuring Kafka Brokers](#).
- **PhysicalConnectionRequirements** – A [PhysicalConnectionRequirements \(p. 771\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the connection.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The description of the connection.

- **ConnectionType** – *Required*: UTF-8 string (valid values: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM).

The type of the connection. Currently, these types are supported:

- **JDBC** - Designates a connection to a database through Java Database Connectivity (JDBC).
- **KAFKA** - Designates a connection to an Apache Kafka streaming platform.
- **MONGODB** - Designates a connection to a MongoDB document database.
- **NETWORK** - Designates a network connection to a data source within an Amazon Virtual Private Cloud environment (Amazon VPC).
- **MARKETPLACE** - Uses configuration settings contained in a connector purchased from AWS Marketplace to read from and write to data stores that are not natively supported by AWS Glue.

- **CUSTOM** – Uses configuration settings contained in a custom connector to read from and write to data stores that are not natively supported by AWS Glue.

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 | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | KAFKA_SASL_MECHANISM | AUTHENTICATION_SECRET_ARN | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME).

Each value is a Value string, not more than 1024 bytes long.

These key-value pairs define parameters for the connection.

- **PhysicalConnectionRequirements** – A [PhysicalConnectionRequirements \(p. 771\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM).
The type of connections to return. Currently, SFTP is not supported.

Operations

- [CreateConnection Action \(Python: create_connection\) \(p. 772\)](#)
- [DeleteConnection Action \(Python: delete_connection\) \(p. 773\)](#)
- [GetConnection Action \(Python: get_connection\) \(p. 773\)](#)
- [GetConnections Action \(Python: get_connections\) \(p. 774\)](#)
- [UpdateConnection Action \(Python: update_connection\) \(p. 775\)](#)
- [BatchDeleteConnection Action \(Python: batch_delete_connection\) \(p. 775\)](#)

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 \(p. 980\)](#).

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 \(p. 770\)](#) object.

A ConnectionInput object defining the connection to create.

- **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 you assign to the connection.

Response

- *No Response parameters.*

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`

- `ResourceNumberLimitExceeded`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 768\)](#) 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 \(p. 980\)](#).

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 \(p. 771\)](#) 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 \(p. 768\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the connection definition to update.

- **ConnectionInput** – *Required*: A [ConnectionInput \(p. 770\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

Each value is a An [ErrorDetail \(p. 975\)](#) 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 \(p. 776\)](#)
- [UserDefinedFunctionInput Structure \(p. 777\)](#)

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 \(p. 980\)](#).
The name of the function.
- **DatabaseName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of the catalog database that contains the function.
- **ClassName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 976\)](#) objects, not more than 1000 structures.
The resource URIs for the function.

- **CatalogId** – Catalog id string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the Data Catalog in which the function resides.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The owner of the function.

- **OwnerType** – UTF-8 string (valid values: `USER` | `ROLE` | `GROUP`).

The owner type.

- **ResourceUris** – An array of [ResourceUri \(p. 976\)](#) objects, not more than 1000 structures.

The resource URIs for the function.

Operations

- [CreateUserDefinedFunction Action \(Python: create_user_defined_function\) \(p. 777\)](#)
- [UpdateUserDefinedFunction Action \(Python: update_user_defined_function\) \(p. 778\)](#)
- [DeleteUserDefinedFunction Action \(Python: delete_user_defined_function\) \(p. 779\)](#)
- [GetUserDefinedFunction Action \(Python: get_user_defined_function\) \(p. 779\)](#)
- [GetUserDefinedFunctions Action \(Python: get_user_defined_functions\) \(p. 780\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the catalog database in which to create the function.

- **FunctionInput** – *Required:* An [UserDefinedFunctionInput \(p. 777\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the function.

- **FunctionInput** – *Required:* An [UserDefinedFunctionInput \(p. 777\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the function definition to be deleted.

Response

- *No Response parameters.*

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the function.

Response

- **UserDefinedFunction** – An [UserDefinedFunction \(p. 776\)](#) 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 \(p. 980\)](#).

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** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the catalog database where the functions are located. If none is provided, functions from all the databases across the catalog will be returned.

- **Pattern** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 100.

The maximum number of functions to return in one response.

Response

- **UserDefinedFunctions** – An array of [UserDefinedFunction \(p. 776\)](#) 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`
- `InvalidInputException`
- `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 \(p. 781\)](#)

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 \(p. 980\)](#).
The name of the person who initiated the migration.

Operations

- [ImportCatalogToGlue Action \(Python: import_catalog_to_glue\) \(p. 781\)](#)
- [GetCatalogImportStatus Action \(Python: get_catalog_import_status\) \(p. 782\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The ID of the catalog to migrate. Currently, this should be the AWS account ID.

Response

- ImportStatus – A [CatalogImportStatus \(p. 781\)](#) 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 \(p. 782\)](#)
- [Crawler API \(p. 792\)](#)
- [Crawler Scheduler API \(p. 806\)](#)

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 \(p. 783\)](#)

- [GrokClassifier Structure \(p. 783\)](#)
- [XMLClassifier Structure \(p. 784\)](#)
- [JsonClassifier Structure \(p. 784\)](#)
- [CsvClassifier Structure \(p. 785\)](#)
- [CreateGrokClassifierRequest Structure \(p. 786\)](#)
- [UpdateGrokClassifierRequest Structure \(p. 786\)](#)
- [CreateXMLClassifierRequest Structure \(p. 786\)](#)
- [UpdateXMLClassifierRequest Structure \(p. 787\)](#)
- [CreateJsonClassifierRequest Structure \(p. 787\)](#)
- [UpdateJsonClassifierRequest Structure \(p. 787\)](#)
- [CreateCsvClassifierRequest Structure \(p. 788\)](#)
- [UpdateCsvClassifierRequest Structure \(p. 788\)](#)

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 \(p. 783\)](#) object.
A classifier that uses `grok`.
- **XMLClassifier** – A [XMLClassifier \(p. 784\)](#) object.
A classifier for XML content.
- **JsonClassifier** – A [JsonClassifier \(p. 784\)](#) object.
A classifier for JSON content.
- **CsvClassifier** – A [CsvClassifier \(p. 785\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(Python: create_classifier\) \(p. 789\)](#)
- [DeleteClassifier Action \(Python: delete_classifier\) \(p. 790\)](#)
- [GetClassifier Action \(Python: get_classifier\) \(p. 790\)](#)
- [GetClassifiers Action \(Python: get_classifiers\) \(p. 790\)](#)
- [UpdateClassifier Action \(Python: update_classifier\) \(p. 791\)](#)

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 \(p. 786\)](#) object.
A `GrokClassifier` object specifying the classifier to create.
- **XMLClassifier** – A [CreateXMLClassifierRequest \(p. 786\)](#) object.
An `XMLClassifier` object specifying the classifier to create.
- **JsonClassifier** – A [CreateJsonClassifierRequest \(p. 787\)](#) object.
A `JsonClassifier` object specifying the classifier to create.
- **CsvClassifier** – A [CreateCsvClassifierRequest \(p. 788\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

Name of the classifier to retrieve.

Response

- `Classifier` – A [Classifier \(p. 783\)](#) 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 \(p. 783\)](#) 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 \(p. 786\)](#) object.
A `GrokClassifier` object with updated fields.
- **XMLClassifier** – An [UpdateXMLClassifierRequest \(p. 787\)](#) object.
An `XMLClassifier` object with updated fields.
- **JsonClassifier** – An [UpdateJsonClassifierRequest \(p. 787\)](#) object.
A `JsonClassifier` object with updated fields.
- **CsvClassifier** – An [UpdateCsvClassifierRequest \(p. 788\)](#) object.
A `CsvClassifier` object with updated fields.

Response

- *No Response parameters.*

Errors

- [InvalidInputException](#)
- [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 \(p. 792\)](#)
- [Schedule Structure \(p. 793\)](#)
- [CrawlerTargets Structure \(p. 794\)](#)
- [S3Target Structure \(p. 794\)](#)
- [JdbcTarget Structure \(p. 795\)](#)
- [MongoDBTarget Structure \(p. 795\)](#)
- [DynamoDBTarget Structure \(p. 795\)](#)
- [DeltaTarget Structure \(p. 796\)](#)
- [CatalogTarget Structure \(p. 796\)](#)
- [CrawlerMetrics Structure \(p. 796\)](#)
- [SchemaChangePolicy Structure \(p. 797\)](#)
- [LastCrawlInfo Structure \(p. 797\)](#)
- [RecrawlPolicy Structure \(p. 798\)](#)
- [LineageConfiguration Structure \(p. 798\)](#)
- [LakeFormationConfiguration Structure \(p. 798\)](#)

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 \(p. 980\)](#).
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 \(p. 794\)](#) 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 \(p. 980\)](#).
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.

- **RecrawlPolicy** – A [RecrawlPolicy \(p. 798\)](#) object.
A policy that specifies whether to crawl the entire dataset again, or to crawl only folders that were added since the last crawler run.
- **SchemaChangePolicy** – A [SchemaChangePolicy \(p. 797\)](#) object.
The policy that specifies update and delete behaviors for the crawler.
- **LineageConfiguration** – A [LineageConfiguration \(p. 798\)](#) object.
A configuration that specifies whether data lineage is enabled 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 \(p. 806\)](#) 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 \(p. 797\)](#) 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 [Include and Exclude Patterns](#).
- **CrawlerSecurityConfiguration** – UTF-8 string, not more than 128 bytes long.
The name of the SecurityConfiguration structure to be used by this crawler.
- **LakeFormationConfiguration** – A [LakeFormationConfiguration \(p. 798\)](#) object.
Specifies whether the crawler should use AWS Lake Formation credentials for the crawler instead of the IAM role credentials.

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 \(p. 794\)](#) objects.
Specifies Amazon Simple Storage Service (Amazon S3) targets.
- **JdbcTargets** – An array of [JdbcTarget \(p. 795\)](#) objects.
Specifies JDBC targets.
- **MongoDBTargets** – An array of [MongoDBTarget \(p. 795\)](#) objects.
Specifies Amazon DocumentDB or MongoDB targets.
- **DynamoDBTargets** – An array of [DynamoDBTarget \(p. 795\)](#) objects.
Specifies Amazon DynamoDB targets.
- **CatalogTargets** – An array of [CatalogTarget \(p. 796\)](#) objects.
Specifies AWS Glue Data Catalog targets.
- **DeltaTargets** – An array of [DeltaTarget \(p. 796\)](#) objects.
Specifies Delta data store 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](#).
- **ConnectionName** – UTF-8 string.
The name of a connection which allows a job or crawler to access data in Amazon S3 within an Amazon Virtual Private Cloud environment (Amazon VPC).
- **SampleSize** – Number (integer).
Sets the number of files in each leaf folder to be crawled when crawling sample files in a dataset. If not set, all the files are crawled. A valid value is an integer between 1 and 249.
- **EventQueueArn** – UTF-8 string.
A valid Amazon SQS ARN. For example, `arn:aws:sqs:region:account:sqs`.
- **DlqEventQueueArn** – UTF-8 string.

A valid Amazon dead-letter SQS ARN. For example,
`arn:aws:sqs:region:account:deadLetterQueue`.

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](#).

MongoDBTarget Structure

Specifies an Amazon DocumentDB or MongoDB data store to crawl.

Fields

- **ConnectionName** – UTF-8 string.
The name of the connection to use to connect to the Amazon DocumentDB or MongoDB target.
- **Path** – UTF-8 string.
The path of the Amazon DocumentDB or MongoDB target (database/collection).
- **ScanAll** – Boolean.
Indicates whether to scan all the records, or to sample rows from the table. Scanning all the records can take a long time when the table is not a high throughput table.
A value of `true` means to scan all records, while a value of `false` means to sample the records. If no value is specified, the value defaults to `true`.

DynamoDBTarget Structure

Specifies an Amazon DynamoDB table to crawl.

Fields

- **Path** – UTF-8 string.
The name of the DynamoDB table to crawl.
- **scanAll** – Boolean.
Indicates whether to scan all the records, or to sample rows from the table. Scanning all the records can take a long time when the table is not a high throughput table.

A value of `true` means to scan all records, while a value of `false` means to sample the records. If no value is specified, the value defaults to `true`.

- `scanRate` – Number (double).

The percentage of the configured read capacity units to use by the AWS Glue crawler. Read capacity units is a term defined by DynamoDB, and is a numeric value that acts as rate limiter for the number of reads that can be performed on that table per second.

The valid values are null or a value between 0.1 to 1.5. A null value is used when user does not provide a value, and defaults to 0.5 of the configured Read Capacity Unit (for provisioned tables), or 0.25 of the max configured Read Capacity Unit (for tables using on-demand mode).

DeltaTarget Structure

Specifies a Delta data store to crawl one or more Delta tables.

Fields

- `DeltaTables` – An array of UTF-8 strings.
A list of the Amazon S3 paths to the Delta tables.
- `ConnectionName` – UTF-8 string.
The name of the connection to use to connect to the Delta table target.
- `WriteManifest` – Boolean.
Specifies whether to write the manifest files to the Delta table path.

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 \(p. 980\)](#).
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.
- `ConnectionName` – UTF-8 string.
The name of the connection for an Amazon S3-backed Data Catalog table to be a target of the crawl when using a Catalog connection type paired with a NETWORK Connection type.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The prefix for a message about this crawl.

- `StartTime` – Timestamp.

The time at which the crawl started.

RecrawlPolicy Structure

When crawling an Amazon S3 data source after the first crawl is complete, specifies whether to crawl the entire dataset again or to crawl only folders that were added since the last crawler run. For more information, see [Incremental Crawls in AWS Glue](#) in the developer guide.

Fields

- `RecrawlBehavior` – UTF-8 string (valid values: `CRAWL_EVERYTHING` | `CRAWL_NEW_FOLDERS_ONLY` | `CRAWL_EVENT_MODE`).

Specifies whether to crawl the entire dataset again or to crawl only folders that were added since the last crawler run.

A value of `CRAWL_EVERYTHING` specifies crawling the entire dataset again.

A value of `CRAWL_NEW_FOLDERS_ONLY` specifies crawling only folders that were added since the last crawler run.

A value of `CRAWL_EVENT_MODE` specifies crawling only the changes identified by Amazon S3 events.

LineageConfiguration Structure

Specifies data lineage configuration settings for the crawler.

Fields

- `CrawlerLineageSettings` – UTF-8 string (valid values: `ENABLE` | `DISABLE`).

Specifies whether data lineage is enabled for the crawler. Valid values are:

- `ENABLE`: enables data lineage for the crawler
- `DISABLE`: disables data lineage for the crawler

LakeFormationConfiguration Structure

Specifies AWS Lake Formation configuration settings for the crawler.

Fields

- `UseLakeFormationCredentials` – Boolean.

Specifies whether to use AWS Lake Formation credentials for the crawler instead of the IAM role credentials.

- **AccountId** – UTF-8 string, not more than 12 bytes long.

Required for cross account crawls. For same account crawls as the target data, this can be left as null.

Operations

- [CreateCrawler Action \(Python: create_crawler\) \(p. 799\)](#)
- [DeleteCrawler Action \(Python: delete_crawler\) \(p. 800\)](#)
- [GetCrawler Action \(Python: get_crawler\) \(p. 801\)](#)
- [GetCrawlers Action \(Python: get_crawlers\) \(p. 801\)](#)
- [GetCrawlerMetrics Action \(Python: get_crawler_metrics\) \(p. 802\)](#)
- [UpdateCrawler Action \(Python: update_crawler\) \(p. 802\)](#)
- [StartCrawler Action \(Python: start_crawler\) \(p. 804\)](#)
- [StopCrawler Action \(Python: stop_crawler\) \(p. 804\)](#)
- [BatchGetCrawlers Action \(Python: batch_get_crawlers\) \(p. 804\)](#)
- [ListCrawlers Action \(Python: list_crawlers\) \(p. 805\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

A description of the new crawler.

- **Targets** – *Required*: A [CrawlerTargets \(p. 794\)](#) 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 \(p. 797\)](#) object.

The policy for the crawler's update and deletion behavior.

- **RecrawlPolicy** – A [RecrawlPolicy \(p. 798\)](#) object.

A policy that specifies whether to crawl the entire dataset again, or to crawl only folders that were added since the last crawler run.

- **LineageConfiguration** – A [LineageConfiguration \(p. 798\)](#) object.

Specifies data lineage configuration settings for the crawler.

- **LakeFormationConfiguration** – A [LakeFormationConfiguration \(p. 798\)](#) object.

Specifies AWS Lake Formation configuration settings for 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.

- **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

- **InvalidArgumentException**
- **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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the crawler to retrieve metadata for.

Response

- `Crawler` – A [Crawler \(p. 792\)](#) 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 \(p. 792\)](#) 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 \(p. 796\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).

A description of the new crawler.

- **Targets** – A [CrawlerTargets \(p. 794\)](#) 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 \(p. 797\)](#) object.

The policy for the crawler's update and deletion behavior.

- **RecrawlPolicy** – A [RecrawlPolicy \(p. 798\)](#) object.

A policy that specifies whether to crawl the entire dataset again, or to crawl only folders that were added since the last crawler run.

- **LineageConfiguration** – A [LineageConfiguration \(p. 798\)](#) object.

Specifies data lineage configuration settings for the crawler.

- **LakeFormationConfiguration** – A [LakeFormationConfiguration \(p. 798\)](#) object.

Specifies AWS Lake Formation configuration settings for 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.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 792\)](#) 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 \(p. 806\)](#)

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 \(Python: update_crawler_schedule\) \(p. 806\)](#)
- [StartCrawlerSchedule Action \(Python: start_crawler_schedule\) \(p. 807\)](#)
- [StopCrawlerSchedule Action \(Python: stop_crawler_schedule\) \(p. 807\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 808\)](#)
- [CodeGenNodeArg Structure \(p. 808\)](#)
- [CodeGenEdge Structure \(p. 809\)](#)
- [Location Structure \(p. 809\)](#)
- [CatalogEntry Structure \(p. 809\)](#)
- [MappingEntry Structure \(p. 810\)](#)

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 \(p. 980\)](#).
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 \(p. 808\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 808\)](#) objects, not more than 50 structures.

A JDBC location.

- **S3** – An array of [CodeGenNodeArg \(p. 808\)](#) objects, not more than 50 structures.

An Amazon Simple Storage Service (Amazon S3) location.

- **DynamoDB** – An array of [CodeGenNodeArg \(p. 808\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(Python: create_script\) \(p. 810\)](#)
- [GetDataflowGraph Action \(Python: get_dataflow_graph\) \(p. 811\)](#)
- [GetMapping Action \(Python: get_mapping\) \(p. 811\)](#)
- [GetPlan Action \(Python: get_plan\) \(p. 812\)](#)

CreateScript Action (Python: create_script)

Transforms a directed acyclic graph (DAG) into code.

Request

- **DagNodes** – An array of [CodeGenNode \(p. 808\)](#) objects.
A list of the nodes in the DAG.
- **DagEdges** – An array of [CodeGenEdge \(p. 809\)](#) 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 \(p. 808\)](#) objects.

A list of the nodes in the resulting DAG.

- **DagEdges** – An array of [CodeGenEdge \(p. 809\)](#) 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 \(p. 809\)](#) object.

Specifies the source table.

- **Sinks** – An array of [CatalogEntry \(p. 809\)](#) objects.

A list of target tables.

- **Location** – A [Location \(p. 809\)](#) object.

Parameters for the mapping.

Response

- **Mapping – Required:** An array of [MappingEntry \(p. 810\)](#) 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 \(p. 810\)](#) objects.
The list of mappings from a source table to target tables.
- **Source – Required:** A [CatalogEntry \(p. 809\)](#) object.
The source table.
- **Sinks –** An array of [CatalogEntry \(p. 809\)](#) objects.
The target tables.
- **Location –** A [Location \(p. 809\)](#) object.
The parameters for the mapping.
- **Language –** UTF-8 string (valid values: PYTHON | SCALA).
The programming language of the code to perform the mapping.
- **AdditionalPlanOptionsMap –** A map array of key-value pairs.
Each key is a UTF-8 string.
Each value is a UTF-8 string.
A map to hold additional optional key-value parameters.
Currently, these key-value pairs are supported:
 - **inferSchema** — Specifies whether to set `inferSchema` to true or false for the default script generated by an AWS Glue job. For example, to set `inferSchema` to true, pass the following key value pair:
`--additional-plan-options-map '{"inferSchema": "true"}'`

Response

- **PythonScript –** UTF-8 string.
A Python script to perform the mapping.

- `ScalaCode` – UTF-8 string.

The Scala code to perform the mapping.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

Visual Job API (Preview)

The Visual Job API is in preview release for AWS Glue and is subject to change.

The Visual Job API allows you to create data integration jobs by using the AWS Glue API from a JSON object that represents a DAG. If you create a DAG from an application outside of AWS Glue Studio, you can create jobs through the AWS Glue Visual Job API. Customers can then use the visual editor in AWS Glue Studio to work with these jobs.

A list of `CodeGenConfigurationNodes` are provided to a create or update job API to register a DAG in AWS Glue Studio for the created job and generate the associated code.

Data Types

- [CodeGenConfigurationNode Structure \(p. 814\)](#)
- [JDBCConnectorOptions Structure \(p. 817\)](#)
- [StreamingDataPreviewOptions Structure \(p. 818\)](#)
- [AthenaConnectorSource Structure \(p. 818\)](#)
- [JDBCConnectorSource Structure \(p. 819\)](#)
- [SparkConnectorSource Structure \(p. 819\)](#)
- [CatalogSource Structure \(p. 820\)](#)
- [CatalogKinesisSource Structure \(p. 820\)](#)
- [DirectKinesisSource Structure \(p. 821\)](#)
- [KinesisStreamingSourceOptions Structure \(p. 821\)](#)
- [CatalogKafkaSource Structure \(p. 823\)](#)
- [DirectKafkaSource Structure \(p. 823\)](#)
- [KafkaStreamingSourceOptions Structure \(p. 824\)](#)
- [RedshiftSource Structure \(p. 825\)](#)
- [S3CatalogSource Structure \(p. 826\)](#)
- [S3SourceAdditionalOptions Structure \(p. 826\)](#)
- [S3CSVSource Structure \(p. 826\)](#)
- [S3JsonSource Structure \(p. 828\)](#)
- [S3ParquetSource Structure \(p. 829\)](#)
- [DynamoDBELTConnectorSource Structure \(p. 830\)](#)

- [DDBELTConnectionOptions Structure \(p. 830\)](#)
- [JDBCConnectorTarget Structure \(p. 831\)](#)
- [SparkConnectorTarget Structure \(p. 831\)](#)
- [BasicCatalogTarget Structure \(p. 832\)](#)
- [RedshiftTarget Structure \(p. 833\)](#)
- [S3CatalogTarget Structure \(p. 833\)](#)
- [S3GlueParquetTarget Structure \(p. 834\)](#)
- [CatalogSchemaChangePolicy Structure \(p. 834\)](#)
- [S3DirectTarget Structure \(p. 834\)](#)
- [DirectSchemaChangePolicy Structure \(p. 835\)](#)
- [ApplyMapping Structure \(p. 835\)](#)
- [Mapping Structure \(p. 836\)](#)
- [SelectFields Structure \(p. 837\)](#)
- [DropFields Structure \(p. 837\)](#)
- [RenameField Structure \(p. 837\)](#)
- [Spigot Structure \(p. 838\)](#)
- [Join Structure \(p. 838\)](#)
- [JoinColumn Structure \(p. 838\)](#)
- [SplitFields Structure \(p. 839\)](#)
- [SelectFromCollection Structure \(p. 839\)](#)
- [FillMissingValues Structure \(p. 839\)](#)
- [Filter Structure \(p. 840\)](#)
- [FilterExpression Structure \(p. 840\)](#)
- [FilterValue Structure \(p. 841\)](#)
- [CustomCode Structure \(p. 841\)](#)
- [SparkSQL Structure \(p. 841\)](#)
- [SqlAlias Structure \(p. 842\)](#)
- [DropNullFields Structure \(p. 842\)](#)
- [NullCheckBoxList Structure \(p. 843\)](#)
- [NullValueField Structure \(p. 843\)](#)
- [Datatype Structure \(p. 843\)](#)
- [Merge Structure \(p. 843\)](#)
- [Union Structure \(p. 844\)](#)

CodeGenConfigurationNode Structure

CodeGenConfigurationNode enumerates all valid Node types. One and only one of its member variables can be populated.

Fields

- [AthenaConnectorSource – An \[AthenaConnectorSource \\(p. 818\\)\]\(#\) object.](#)

Specifies a connector to an Amazon Athena data source.

- [JDBCConnectorSource](#) – A [JDBCConnectorSource \(p. 819\)](#) object.
Specifies a connector to a JDBC data source.
- [SparkConnectorSource](#) – A [SparkConnectorSource \(p. 819\)](#) object.
Specifies a connector to an Apache Spark data source.
- [CatalogSource](#) – A [CatalogSource \(p. 820\)](#) object.
Specifies a data store in the AWS Glue Data Catalog.
- [RedshiftSource](#) – A [RedshiftSource \(p. 825\)](#) object.
Specifies an Amazon Redshift data store.
- [S3CatalogSource](#) – A [S3CatalogSource \(p. 826\)](#) object.
Specifies an Amazon S3 data store in the AWS Glue Data Catalog.
- [S3CSVSource](#) – A [S3CSVSource \(p. 826\)](#) object.
Specifies a command-separated value (CSV) data store stored in Amazon S3.
- [S3JsonSource](#) – A [S3JsonSource \(p. 828\)](#) object.
Specifies a JSON data store stored in Amazon S3.
- [S3ParquetSource](#) – A [S3ParquetSource \(p. 829\)](#) object.
Specifies an Apache Parquet data store stored in Amazon S3.
- [JDBCConnectorTarget](#) – A [JDBCConnectorTarget \(p. 831\)](#) object.
Specifies a data target that writes to Amazon S3 in Apache Parquet columnar storage.
- [SparkConnectorTarget](#) – A [SparkConnectorTarget \(p. 831\)](#) object.
Specifies a target that uses an Apache Spark connector.
- [CatalogTarget](#) – A [BasicCatalogTarget \(p. 832\)](#) object.
Specifies a target that uses a AWS Glue Data Catalog table.
- [RedshiftTarget](#) – A [RedshiftTarget \(p. 833\)](#) object.
Specifies a target that uses Amazon Redshift.
- [S3CatalogTarget](#) – A [S3CatalogTarget \(p. 833\)](#) object.
Specifies a data target that writes to Amazon S3 using the AWS Glue Data Catalog.
- [S3GlueParquetTarget](#) – A [S3GlueParquetTarget \(p. 834\)](#) object.
Specifies a data target that writes to Amazon S3 in Apache Parquet columnar storage.
- [S3DirectTarget](#) – A [S3DirectTarget \(p. 834\)](#) object.
Specifies a data target that writes to Amazon S3.
- [ApplyMapping](#) – An [ApplyMapping \(p. 835\)](#) object.
Specifies a transform that maps data property keys in the data source to data property keys in the data target. You can rename keys, modify the data types for keys, and choose which keys to drop from the dataset.
- [SelectFields](#) – A [SelectFields \(p. 837\)](#) object.
Specifies a transform that chooses the data property keys that you want to keep.
- [DropFields](#) – A [DropFields \(p. 837\)](#) object.

Specifies a transform that chooses the data property keys that you want to drop.

- [RenameField](#) – A [RenameField \(p. 837\)](#) object.

Specifies a transform that renames a single data property key.

- [Spigot](#) – A [Spigot \(p. 838\)](#) object.

Specifies a transform that writes samples of the data to an Amazon S3 bucket.

- [Join](#) – A [Join \(p. 838\)](#) object.

Specifies a transform that joins two datasets into one dataset using a comparison phrase on the specified data property keys. You can use inner, outer, left, right, left semi, and left anti joins.

- [SplitFields](#) – A [SplitFields \(p. 839\)](#) object.

Specifies a transform that splits data property keys into two `DynamicFrames`. The output is a collection of `DynamicFrames`: one with selected data property keys, and one with the remaining data property keys.

- [SelectFromCollection](#) – A [SelectFromCollection \(p. 839\)](#) object.

Specifies a transform that chooses one `DynamicFrame` from a collection of `DynamicFrames`. The output is the selected `DynamicFrame`.

- [FillMissingValues](#) – A [FillMissingValues \(p. 839\)](#) object.

Specifies a transform that locates records in the dataset that have missing values and adds a new field with a value determined by imputation. The input data set is used to train the machine learning model that determines what the missing value should be.

- [Filter](#) – A [Filter \(p. 840\)](#) object.

Specifies a transform that splits a dataset into two, based on a filter condition.

- [CustomCode](#) – A [CustomCode \(p. 841\)](#) object.

Specifies a transform that uses custom code you provide to perform the data transformation. The output is a collection of `DynamicFrames`.

- [SparkSQL](#) – A [SparkSQL \(p. 841\)](#) object.

Specifies a transform where you enter a SQL query using Spark SQL syntax to transform the data. The output is a single `DynamicFrame`.

- [DirectKinesisSource](#) – A [DirectKinesisSource \(p. 821\)](#) object.

Specifies a direct Amazon Kinesis data source.

- [DirectKafkaSource](#) – A [DirectKafkaSource \(p. 823\)](#) object.

Specifies an Apache Kafka data store.

- [CatalogKinesisSource](#) – A [CatalogKinesisSource \(p. 820\)](#) object.

Specifies a Kinesis data source in the AWS Glue Data Catalog.

- [CatalogKafkaSource](#) – A [CatalogKafkaSource \(p. 823\)](#) object.

Specifies an Apache Kafka data store in the Data Catalog.

- [DropNullFields](#) – A [DropNullFields \(p. 842\)](#) object.

Specifies a transform that removes columns from the dataset if all values in the column are 'null'. By default, AWS Glue Studio will recognize null objects, but some values such as empty strings, strings that are "null", -1 integers or other placeholders such as zeros, are not automatically recognized as nulls.

- [Merge](#) – A [Merge \(p. 843\)](#) object.

Specifies a transform that merges a `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.

- `Union` – An [Union \(p. 844\)](#) object.

Specifies a transform that combines the rows from two or more datasets into a single result.

- `DynamoDBELTConnectorSource` – A [DynamoDBELTConnectorSource \(p. 830\)](#) object.

Specifies a connector to an Amazon DynamoDB ELT source.

JDBCConnectorOptions Structure

Additional connection options for the connector.

Fields

- `FilterPredicate` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Extra condition clause to filter data from source. For example:

```
BillingCity='Mountain View'
```

When using a query instead of a table name, you should validate that the query works with the specified `filterPredicate`.

- `PartitionColumn` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of an integer column that is used for partitioning. This option works only when it's included with `lowerBound`, `upperBound`, and `numPartitions`. This option works the same way as in the Spark SQL JDBC reader.

- `LowerBound` – Number (long), not more than None.

The minimum value of `partitionColumn` that is used to decide partition stride.

- `UpperBound` – Number (long), not more than None.

The maximum value of `partitionColumn` that is used to decide partition stride.

- `NumPartitions` – Number (long), not more than None.

The number of partitions. This value, along with `lowerBound` (inclusive) and `upperBound` (exclusive), form partition strides for generated WHERE clause expressions that are used to split the `partitionColumn`.

- `JobBookmarkKeys` – An array of UTF-8 strings, not more than 100 strings.

The name of the job bookmark keys on which to sort.

- `jobBookmarkKeysSortOrder` – UTF-8 string (valid values: `asc="ASC"` | `desc="DESC"`).

Specifies an ascending or descending sort order.

- `dataTypeMapping` – A map array of key-value pairs.

Each key is a UTF-8 string (valid values: `ARRAY` | `BIGINT` | `BINARY` | `BIT` | `BLOB` | `BOOLEAN` | `CHAR` | `CLOB` | `DATALINK` | `DATE` | `DECIMAL` | `DISTINCT` | `DOUBLE` | `FLOAT` | `INTEGER` | `JAVA_OBJECT` | `LONGNVARCHAR` | `LONGVARBINARY` | `LONGVARCHAR` | `NCHAR` | `NCLOB` | `NULL` | `NUMERIC` | `NVARCHAR` | `OTHER` | `REAL` | `REF` | `REF_CURSOR` | `ROWID` | `SMALLINT` | `SQLXML` | `STRUCT` | `TIME` |

TIME_WITH_TIMEZONE | TIMESTAMP | TIMESTAMP_WITH_TIMEZONE | TINYINT | VARBINARY | VARCHAR).

Each value is a UTF-8 string (valid values: ARRAY | BIGINT | BINARY | BIT | BLOB | BOOLEAN | CHAR | CLOB | DATALINK | DATE | DECIMAL | DISTINCT | DOUBLE | FLOAT | INTEGER | JAVA_OBJECT | LONGNVARCHAR | LONGVARBINARY | LONGVARCHAR | NCHAR | NCLOB | NULL | NUMERIC | NVARCHAR | OTHER | REAL | REF | REF_CURSOR | ROWID | SMALLINT | SQLXML | STRUCT | TIME | TIME_WITH_TIMEZONE | TIMESTAMP | TIMESTAMP_WITH_TIMEZONE | TINYINT | VARBINARY | VARCHAR).

Custom data type mapping that builds a mapping from a JDBC data type to an AWS Glue data type. For example, the option "dataTypeMapping": {"FLOAT": "STRING"} maps data fields of JDBC type FLOAT into the Java String type by calling the ResultSet.getString() method of the driver, and uses it to build the AWS Glue record. The ResultSet object is implemented by each driver, so the behavior is specific to the driver you use. Refer to the documentation for your JDBC driver to understand how the driver performs the conversions.

StreamingDataPreviewOptions Structure

Specifies options related to data preview for viewing a sample of your data.

Fields

- **PollingTime** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The polling time in milliseconds.

- **RecordPollingLimit** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The limit to the number of records polled.

AthenaConnectorSource Structure

Specifies a connector to an Amazon Athena data source.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data source.

- **ConnectionName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the connection that is associated with the connector.

- **ConnectorName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of a connector that assists with accessing the data store in AWS Glue Studio.

- **ConnectionType** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The type of connection, such as marketplace.athena or custom.athena, designating a connection to an Amazon Athena data store.

- **ConnectionTable** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the table in the data source.

- **SchemaName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the Cloudwatch log group to read from. For example, /aws-glue/jobs/output.

JDBCConnectorSource Structure

Specifies a connector to a JDBC data source.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data source.

- **ConnectionName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the connection that is associated with the connector.

- **ConnectorName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of a connector that assists with accessing the data store in AWS Glue Studio.

- **ConnectionType** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The type of connection, such as marketplace.jdbc or custom.jdbc, designating a connection to a JDBC data store.

- **AdditionalOptions** – A [JDBCConnectorOptions \(p. 817\)](#) object.

Additional connection options for the connector.

- **ConnectionTable** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the table in the data source.

- **query** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The table or SQL query to get the data from. You can specify either ConnectionTable or query, but not both.

SparkConnectorSource Structure

Specifies a connector to an Apache Spark data source.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data source.

- **ConnectionName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the connection that is associated with the connector.
- **ConnectorName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of a connector that assists with accessing the data store in AWS Glue Studio.
- **ConnectionType** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The type of connection, such as marketplace.spark or custom.spark, designating a connection to an Apache Spark data store.
- **AdditionalOptions** – A map array of key-value pairs, not more than 10 pairs.
 - Each key is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - Each value is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
- Additional connection options for the connector.

CatalogSource Structure

Specifies a data store in the AWS Glue Data Catalog.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the data store.
- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the database to read from.
- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the table in the database to read from.

CatalogKinesisSource Structure

Specifies a Kinesis data source in the AWS Glue Data Catalog.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the data source.
- **WindowSize** – Number (integer), at least 1.
The amount of time to spend processing each micro batch.

- **DetectSchema** – Boolean.
Whether to automatically determine the schema from the incoming data.
- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the table in the database to read from.
- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the database to read from.
- **StreamingOptions** – A [KinesisStreamingSourceOptions \(p. 821\)](#) object.
Additional options for the Kinesis streaming data source.
- **DataPreviewOptions** – A [StreamingDataPreviewOptions \(p. 818\)](#) object.
Additional options for data preview.

DirectKinesisSource Structure

Specifies a direct Amazon Kinesis data source.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the data source.
- **WindowSize** – Number (integer), at least 1.
The amount of time to spend processing each micro batch.
- **DetectSchema** – Boolean.
Whether to automatically determine the schema from the incoming data.
- **StreamingOptions** – A [KinesisStreamingSourceOptions \(p. 821\)](#) object.
Additional options for the Kinesis streaming data source.
- **DataPreviewOptions** – A [StreamingDataPreviewOptions \(p. 818\)](#) object.
Additional options for data preview.

KinesisStreamingSourceOptions Structure

Additional options for the Amazon Kinesis streaming data source.

Fields

- **EndpointUrl** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The URL of the Kinesis endpoint.
- **StreamName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the Kinesis data stream.

- **Classification** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

An optional classification.

- **Delimiter** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Specifies the delimiter character.

- **StartingPosition** – UTF-8 string (valid values: `latest="LATEST"` | `trim_horizon="TRIM_HORIZON"` | `earliest="EARLIEST"`).

The starting position in the Kinesis data stream to read data from. The possible values are "latest", "trim_horizon", or "earliest". The default value is "latest".

- **MaxFetchTimeInMs** – Number (long), not more than None.

The maximum time spent in the job executor to fetch a record from the Kinesis data stream per shard, specified in milliseconds (ms). The default value is 1000.

- **MaxFetchRecordsPerShard** – Number (long), not more than None.

The maximum number of records to fetch per shard in the Kinesis data stream. The default value is 100000.

- **MaxRecordPerRead** – Number (long), not more than None.

The maximum number of records to fetch from the Kinesis data stream in each getRecords operation. The default value is 10000.

- **AddIdleTimeBetweenReads** – Boolean.

Adds a time delay between two consecutive getRecords operations. The default value is "False". This option is only configurable for Glue version 2.0 and above.

- **IdleTimeBetweenReadsInMs** – Number (long), not more than None.

The minimum time delay between two consecutive getRecords operations, specified in ms. The default value is 1000. This option is only configurable for Glue version 2.0 and above.

- **DescribeShardInterval** – Number (long), not more than None.

The minimum time interval between two ListShards API calls for your script to consider resharding. The default value is 1s.

- **NumRetries** – Number (integer), not more than None.

The maximum number of retries for Kinesis Data Streams API requests. The default value is 3.

- **RetryIntervalMs** – Number (long), not more than None.

The cool-off time period (specified in ms) before retrying the Kinesis Data Streams API call. The default value is 1000.

- **MaxRetryIntervalMs** – Number (long), not more than None.

The maximum cool-off time period (specified in ms) between two retries of a Kinesis Data Streams API call. The default value is 10000.

- **AvoidEmptyBatches** – Boolean.

Avoids creating an empty microbatch job by checking for unread data in the Kinesis data stream before the batch is started. The default value is "False".

- **StreamARN** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the Kinesis data stream.

- `AwsSTSRoleARN` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the role to assume using AWS Security Token Service (AWS STS). This role must have permissions for describe or read record operations for the Kinesis data stream. You must use this parameter when accessing a data stream in a different account. Used in conjunction with "awsSTSSessionName".

- `AwsSTSSessionName` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

An identifier for the session assuming the role using AWS STS. You must use this parameter when accessing a data stream in a different account. Used in conjunction with "awsSTSRoleARN".

CatalogKafkaSource Structure

Specifies an Apache Kafka data store in the Data Catalog.

Fields

- `Name` – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- `WindowSize` – Number (integer), at least 1.

The amount of time to spend processing each micro batch.

- `DetectSchema` – Boolean.

Whether to automatically determine the schema from the incoming data.

- `Table` – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the table in the database to read from.

- `Database` – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the database to read from.

- `StreamingOptions` – A [KafkaStreamingSourceOptions \(p. 824\)](#) object.

Specifies the streaming options.

- `DataPreviewOptions` – A [StreamingDataPreviewOptions \(p. 818\)](#) object.

Specifies options related to data preview for viewing a sample of your data.

DirectKafkaSource Structure

Specifies an Apache Kafka data store.

Fields

- `Name` – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- **StreamingOptions** – A [KafkaStreamingSourceOptions \(p. 824\)](#) object.
 - Specifies the streaming options.
- **WindowSize** – Number (integer), at least 1.
 - The amount of time to spend processing each micro batch.
- **DetectSchema** – Boolean.
 - Whether to automatically determine the schema from the incoming data.
- **DataPreviewOptions** – A [StreamingDataPreviewOptions \(p. 818\)](#) object.
 - Specifies options related to data preview for viewing a sample of your data.

KafkaStreamingSourceOptions Structure

Additional options for streaming.

Fields

- **BootstrapServers** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - A list of bootstrap server URLs, for example, as b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094. This option must be specified in the API call or defined in the table metadata in the Data Catalog.
- **SecurityProtocol** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - The protocol used to communicate with brokers. The possible values are "SSL" or "PLAINTEXT".
- **ConnectionName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - The name of the connection.
- **TopicName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - The topic name as specified in Apache Kafka. You must specify at least one of "topicName", "assign" or "subscribePattern".
- **Assign** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - The specific TopicPartitions to consume. You must specify at least one of "topicName", "assign" or "subscribePattern".
- **SubscribePattern** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - A Java regex string that identifies the topic list to subscribe to. You must specify at least one of "topicName", "assign" or "subscribePattern".
- **Classification** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
 - An optional classification.
- **Delimiter** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Specifies the delimiter character.

- **StartingOffsets** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The starting position in the Kafka topic to read data from. The possible values are "earliest" or "latest". The default value is "latest".

- **EndingOffsets** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The end point when a batch query is ended. Possible values are either "latest" or a JSON string that specifies an ending offset for each `TopicPartition`.

- **PollTimeoutMs** – Number (long), not more than None.

The timeout in milliseconds to poll data from Kafka in Spark job executors. The default value is 512.

- **NumRetries** – Number (integer), not more than None.

The number of times to retry before failing to fetch Kafka offsets. The default value is 3.

- **RetryIntervalMs** – Number (long), not more than None.

The time in milliseconds to wait before retrying to fetch Kafka offsets. The default value is 10.

- **MaxOffsetsPerTrigger** – Number (long), not more than None.

The rate limit on the maximum number of offsets that are processed per trigger interval. The specified total number of offsets is proportionally split across `topicPartitions` of different volumes. The default value is null, which means that the consumer reads all offsets until the known latest offset.

- **MinPartitions** – Number (integer), not more than None.

The desired minimum number of partitions to read from Kafka. The default value is null, which means that the number of spark partitions is equal to the number of Kafka partitions.

RedshiftSource Structure

Specifies an Amazon Redshift data store.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the Amazon Redshift data store.

- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The database to read from.

- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The database table to read from.

- **RedshiftTmpDir** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The Amazon S3 path where temporary data can be staged when copying out of the database.

- **tmpDirIAMRole** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The IAM role with permissions.

S3CatalogSource Structure

Specifies an Amazon S3 data store in the AWS Glue Data Catalog.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The database to read from.

- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The database table to read from.

- **PartitionPredicate** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Partitions satisfying this predicate are deleted. Files within the retention period in these partitions are not deleted. Set to "" – empty by default.

- **additionalOptions** – A [S3SourceAdditionalOptions \(p. 826\)](#) object.

Specifies additional connection options.

S3SourceAdditionalOptions Structure

Specifies additional connection options for the Amazon S3 data store.

Fields

- **BoundedSize** – Number (long).

Sets the upper limit for the target size of the dataset in bytes that will be processed.

- **BoundedFiles** – Number (long).

Sets the upper limit for the target number of files that will be processed.

S3CSVSource Structure

Specifies a command-separated value (CSV) data store stored in Amazon S3.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- **Paths** – *Required*: An array of UTF-8 strings, not more than 100 strings.
A list of the Amazon S3 paths to read from.
- **CompressionType** – UTF-8 string (valid values: `gzip="GZIP"` | `bzip2="BZIP2"`).
Specifies how the data is compressed. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip").
- **Exclusions** – An array of UTF-8 strings, not more than 100 strings.
A string containing a JSON list of Unix-style glob patterns to exclude. For example, "[\\"**.pdf\\"]" excludes all PDF files.
- **GroupSize** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
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.
- **GroupFiles** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
Grouping files is turned on by default when the input contains more than 50,000 files. To turn on 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".
- **Recurse** – Boolean.
If set to true, recursively reads files in all subdirectories under the specified paths.
- **MaxBand** – Number (integer), not more than None.
This option controls the duration in milliseconds after which the s3 listing is likely to be consistent. Files with modification timestamps falling within the last maxBand milliseconds are tracked specially when using JobBookmarks to account for Amazon S3 eventual consistency. Most users don't need to set this option. The default is 900000 milliseconds, or 15 minutes.
- **MaxFilesInBand** – Number (integer), not more than None.
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.
- **AdditionalOptions** – A [S3SourceAdditionalOptions \(p. 826\)](#) object.
Specifies additional connection options.
- **Separator** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
Specifies the delimiter character. The default is a comma: ",", but any other character can be specified.
- **Escaper** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
Specifies a character to use for escaping. This option is used only when reading CSV files. 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 `\0`).
- **QuoteChar** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
Specifies the character to use for quoting. The default is a double quote: ' "''. Set this to `-1` to turn off quoting entirely.
- **Multiline** – Boolean.

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` – Boolean.

A Boolean value that specifies whether to treat the first line as a header. The default value is `False`.

- `WriteHeader` – Boolean.

A Boolean value that specifies whether to write the header to output. The default value is `True`.

- `SkipFirst` – Boolean.

A Boolean value that specifies whether to skip the first data line. The default value is `False`.

- `optimizePerformance` – Boolean.

A Boolean value that specifies whether to use the advanced SIMD CSV reader along with Apache Arrow based columnar memory formats. Only available in AWS Glue 3.0.

S3JsonSource Structure

Specifies a JSON data store stored in Amazon S3.

Fields

- `Name` – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- `Paths` – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the Amazon S3 paths to read from.

- `CompressionType` – UTF-8 string (valid values: `gzip="GZIP"` | `bzip2="BZIP2"`).

Specifies how the data is compressed. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip".

- `Exclusions` – An array of UTF-8 strings, not more than 100 strings.

A string containing a JSON list of Unix-style glob patterns to exclude. For example, "[\\"**.pdf\\"]" excludes all PDF files.

- `GroupSize` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

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.

- `GroupFiles` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Grouping files is turned on by default when the input contains more than 50,000 files. To turn on 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".

- `Recurse` – Boolean.

If set to true, recursively reads files in all subdirectories under the specified paths.

- **MaxBand** – Number (integer), not more than None.

This option controls the duration in milliseconds after which the s3 listing is likely to be consistent. Files with modification timestamps falling within the last maxBand milliseconds are tracked specially when using JobBookmarks to account for Amazon S3 eventual consistency. Most users don't need to set this option. The default is 900000 milliseconds, or 15 minutes.

- **MaxFilesInBand** – Number (integer), not more than None.

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.

- **AdditionalOptions** – A [S3SourceAdditionalOptions \(p. 826\)](#) object.

Specifies additional connection options.

- **JsonPath** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A JsonPath string defining the JSON data.

- **Multiline** – Boolean.

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.

S3ParquetSource Structure

Specifies an Apache Parquet data store stored in Amazon S3.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- **Paths** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A list of the Amazon S3 paths to read from.

- **CompressionType** – UTF-8 string (valid values: snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED").

Specifies how the data is compressed. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip".

- **Exclusions** – An array of UTF-8 strings, not more than 100 strings.

A string containing a JSON list of Unix-style glob patterns to exclude. For example, "[\ **.pdf\]" excludes all PDF files.

- **GroupSize** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

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.

- **GroupFiles** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Grouping files is turned on by default when the input contains more than 50,000 files. To turn on 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".

- **Recurse** – Boolean.

If set to true, recursively reads files in all subdirectories under the specified paths.

- **MaxBand** – Number (integer), not more than None.

This option controls the duration in milliseconds after which the s3 listing is likely to be consistent. Files with modification timestamps falling within the last maxBand milliseconds are tracked specially when using JobBookmarks to account for Amazon S3 eventual consistency. Most users don't need to set this option. The default is 900000 milliseconds, or 15 minutes.

- **MaxFilesInBand** – Number (integer), not more than None.

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.

- **AdditionalOptions** – A [S3SourceAdditionalOptions \(p. 826\)](#) object.

Specifies additional connection options.

DynamoDBELTConnectorSource Structure

Specifies a connector to an Amazon DynamoDB ELT source.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data store.

- **ConnectionOptions** – *Required*: A [DDBELTConnectionOptions \(p. 830\)](#) object.

The options for an Amazon Dynamo DB ELT connection.

DDBELTConnectionOptions Structure

Options for an Amazon Dynamo DB ELT connection.

Fields

- **dynamodbUseExportConnector** – *Required*: Boolean.

Specifies exporting your Amazon DynamoDB table data to Amazon S3.

- **dynamodbTableArn** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the DynamoDB table to read from.

- **dynamodbS3Bucket** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the Amazon S3 bucket where your DynamoDB table is stored.

- **dynamodbS3Prefix** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The Amazon S3 bucket prefix.

- `dynamodbS3BucketOwner` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The AWS account used to create the bucket.

- `dynamodbStsRoleArn` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The IAM role ARN to be assumed for cross-account access. This parameter is available in AWS Glue 1.0 or later.

JDBCConnectorTarget Structure

Specifies a data target that writes to Amazon S3 in Apache Parquet columnar storage.

Fields

- `Name` – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data target.

- `Inputs` – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The nodes that are inputs to the data target.

- `ConnectionName` – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the connection that is associated with the connector.

- `ConnectionTable` – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the table in the data target.

- `ConnectorName` – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of a connector that will be used.

- `ConnectionType` – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The type of connection, such as `marketplace.jdbc` or `custom.jdbc`, designating a connection to a JDBC data target.

- `AdditionalOptions` – A map array of key-value pairs, not more than 10 pairs.

Each key is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Additional connection options for the connector.

SparkConnectorTarget Structure

Specifies a target that uses an Apache Spark connector.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data target.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The nodes that are inputs to the data target.

- **ConnectionName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of a connection for an Apache Spark connector.

- **ConnectorName** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of an Apache Spark connector.

- **ConnectionType** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The type of connection, such as marketplace.spark or custom.spark, designating a connection to an Apache Spark data store.

- **AdditionalOptions** – A map array of key-value pairs, not more than 10 pairs.

Each key is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Additional connection options for the connector.

BasicCatalogTarget Structure

Specifies a target that uses a AWS Glue Data Catalog table.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of your data target.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The nodes that are inputs to the data target.

- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The database that contains the table you want to use as the target. This database must already exist in the Data Catalog.

- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The table that defines the schema of your output data. This table must already exist in the Data Catalog.

RedshiftTarget Structure

Specifies a target that uses Amazon Redshift.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the data target.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.
The nodes that are inputs to the data target.
- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the database to write to.
- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the table in the database to write to.
- **RedshiftTmpDir** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The Amazon S3 path where temporary data can be staged when copying out of the database.
- **TmpDirIAMRole** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The IAM role with permissions.

S3CatalogTarget Structure

Specifies a data target that writes to Amazon S3 using the AWS Glue Data Catalog.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the data target.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.
The nodes that are inputs to the data target.
- **PartitionKeys** – An array of UTF-8 strings, not more than 100 strings.
Specifies native partitioning using a sequence of keys.
- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the table in the database to write to.
- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the database to write to.
- **SchemaChangePolicy** – A [CatalogSchemaChangePolicy \(p. 834\)](#) object.

A policy that specifies update behavior for the crawler.

S3GlueParquetTarget Structure

Specifies a data target that writes to Amazon S3 in Apache Parquet columnar storage.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data target.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The nodes that are inputs to the data target.

- **PartitionKeys** – An array of UTF-8 strings, not more than 100 strings.

Specifies native partitioning using a sequence of keys.

- **Path** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A single Amazon S3 path to write to.

- **Compression** – UTF-8 string (valid values: `snappy="SNAPPY"` | `lzo="LZO"` | `gzip="GZIP"` | `uncompressed="UNCOMPRESSED"`).

Specifies how the data is compressed. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip".

- **SchemaChangePolicy** – A [DirectSchemaChangePolicy \(p. 835\)](#) object.

A policy that specifies update behavior for the crawler.

CatalogSchemaChangePolicy Structure

A policy that specifies update behavior for the crawler.

Fields

- **EnableUpdateCatalog** – Boolean.

Whether to use the specified update behavior when the crawler finds a changed schema.

- **UpdateBehavior** – UTF-8 string (valid values: `UPDATE_IN_DATABASE` | `LOG`).

The update behavior when the crawler finds a changed schema.

S3DirectTarget Structure

Specifies a data target that writes to Amazon S3.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the data target.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The nodes that are inputs to the data target.

- **PartitionKeys** – An array of UTF-8 strings, not more than 100 strings.

Specifies native partitioning using a sequence of keys.

- **Path** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A single Amazon S3 path to write to.

- **Compression** – UTF-8 string (valid values: `gzip="GZIP"` | `bzip2="BZIP2"`).

Specifies how the data is compressed. This is generally not necessary if the data has a standard file extension. Possible values are "gzip" and "bzip".

- **Format** – *Required*: UTF-8 string (valid values: `json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"`).

Specifies the data output format for the target.

- **SchemaChangePolicy** – A [DirectSchemaChangePolicy \(p. 835\)](#) object.

A policy that specifies update behavior for the crawler.

DirectSchemaChangePolicy Structure

A policy that specifies update behavior for the crawler.

Fields

- **EnableUpdateCatalog** – *Required*: Boolean.

Whether to use the specified update behavior when the crawler finds a changed schema.

- **UpdateBehavior** – *Required*: UTF-8 string (valid values: `UPDATE_IN_DATABASE` | `LOG`).

The update behavior when the crawler finds a changed schema.

- **Table** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Specifies the table in the database that the schema change policy applies to.

- **Database** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

Specifies the database that the schema change policy applies to.

ApplyMapping Structure

Specifies a transform that maps data property keys in the data source to data property keys in the data target. You can rename keys, modify the data types for keys, and choose which keys to drop from the dataset.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs – Required:** An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Mapping – Required:** An array of [Mapping \(p. 836\)](#) objects, not more than 250 structures.

Specifies the mapping of data property keys in the data source to data property keys in the data target.

Mapping Structure

Specifies the mapping of data property keys.

Fields

- **ToKey – Required:** UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

After the apply mapping, what the name of the column should be. Can be the same as `FromPath`.

- **FromPath – Required:** An array of UTF-8 strings, not more than 100 strings.

The table or column to be modified.

- **FromType – Required:** UTF-8 string (valid values: `bigint="BIGINT"` | `binary="BINARY"` | `boolean="BOOLEAN"` | `char="CHAR"` | `date="DATE"` | `decimal="DECIMAL"` | `double="DOUBLE"` | `float="FLOAT"` | `int="INT"` | `interval="INTERVAL"` | `long="LONG"` | `smallint="SMALLINT"` | `string="STRING"` | `timestamp="TIMESTAMP"` | `tinyint="TINYINT"` | `varchar="VARCHAR"`).

The type of the data to be modified.

- **ToType – Required:** UTF-8 string (valid values: `bigint="BIGINT"` | `binary="BINARY"` | `boolean="BOOLEAN"` | `char="CHAR"` | `date="DATE"` | `decimal="DECIMAL"` | `double="DOUBLE"` | `float="FLOAT"` | `int="INT"` | `interval="INTERVAL"` | `long="LONG"` | `smallint="SMALLINT"` | `string="STRING"` | `timestamp="TIMESTAMP"` | `tinyint="TINYINT"` | `varchar="VARCHAR"`).

The data type that the data is to be modified to.

- **Dropped – Boolean:**

If true, then the column is removed.

- **Children – An array of [Mapping \(p. 836\)](#) objects, not more than 250 structures.**

Only applicable to nested data structures. If you want to change the parent structure, but also one of its children, you can fill out this data strucutre. It is also `Mapping`, but its `FromPath` will be the parent's `FromPath` plus the `FromPath` from this structure.

For the children part, suppose you have the structure:

```
{ "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType": "Struct", "Dropped": false, "Children": [ { "FromPath": "inner", "ToKey": "inner", "ToType": "Double", "Dropped": false, } ] }
```

You can specify a `Mapping` that looks like:

```
{ "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType": "Struct", "Dropped": false, "Children": [ { "FromPath": "inner", "ToKey": "inner", "ToType": "Double", "Dropped": false, } ] }
```

SelectFields Structure

Specifies a transform that chooses the data property keys that you want to keep.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Paths** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A JSON path to a variable in the data structure.

DropFields Structure

Specifies a transform that chooses the data property keys that you want to drop.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Paths** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A JSON path to a variable in the data structure.

RenameField Structure

Specifies a transform that renames a single data property key.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **SourcePath** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A JSON path to a variable in the data structure for the source data.

- **TargetPath** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A JSON path to a variable in the data structure for the target data.

Spigot Structure

Specifies a transform that writes samples of the data to an Amazon S3 bucket.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Path** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A path in Amazon S3 where the transform will write a subset of records from the dataset to a JSON file in an Amazon S3 bucket.

- **Topk** – Number (integer), not more than 100.

Specifies a number of records to write starting from the beginning of the dataset.

- **Prob** – Number (double), not more than 1.

The probability (a decimal value with a maximum value of 1) of picking any given record. A value of 1 indicates that each row read from the dataset should be included in the sample output.

Join Structure

Specifies a transform that joins two datasets into one dataset using a comparison phrase on the specified data property keys. You can use inner, outer, left, right, left semi, and left anti joins.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 2 or more than 2 strings.

The data inputs identified by their node names.

- **JoinType** – *Required*: UTF-8 string (valid values: `equijoin="EQUIJOIN" | left="LEFT" | right="RIGHT" | outer="OUTER" | leftsemi="LEFT_SEMI" | leftanti="LEFT_ANTI"`).

Specifies the type of join to be performed on the datasets.

- **Columns** – *Required*: An array of [JoinColumn \(p. 838\)](#) objects, not less than 2 or more than 2 structures.

A list of the two columns to be joined.

JoinColumn Structure

Specifies a column to be joined.

Fields

- **from** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The column to be joined.

- **keys** – *Required*: An array of UTF-8 strings, not more than 100 strings.

The key of the column to be joined.

SplitFields Structure

Specifies a transform that splits data property keys into two `DynamicFrames`. The output is a collection of `DynamicFrames`: one with selected data property keys, and one with the remaining data property keys.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Paths** – *Required*: An array of UTF-8 strings, not more than 100 strings.

A JSON path to a variable in the data structure.

SelectFromCollection Structure

Specifies a transform that chooses one `DynamicFrame` from a collection of `DynamicFrames`. The output is the selected `DynamicFrame`.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Index** – *Required*: Number (integer), not more than None.

The index for the `DynamicFrame` to be selected.

FillMissingValues Structure

Specifies a transform that locates records in the dataset that have missing values and adds a new field with a value determined by imputation. The input data set is used to train the machine learning model that determines what the missing value should be.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **ImputedPath** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A JSON path to a variable in the data structure for the dataset that is imputed.

- **FilledPath** – UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A JSON path to a variable in the data structure for the dataset that is filled.

Filter Structure

Specifies a transform that splits a dataset into two, based on a filter condition.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **LogicalOperator** – *Required*: UTF-8 string (valid values: AND | OR).

The operator used to filter rows by comparing the key value to a specified value.

- **Filters** – *Required*: An array of [FilterExpression \(p. 840\)](#) objects, not more than 20 structures.

Specifies a filter expression.

FilterExpression Structure

Specifies a filter expression.

Fields

- **Operation** – *Required*: UTF-8 string (valid values: EQ | LT | GT | LTE | GTE | REGEX | ISNULL).

The type of operation to perform in the expression.

- **Negated** – Boolean.

Whether the expression is to be negated.

- **Values** – *Required*: An array of [FilterValue \(p. 841\)](#) objects, not more than 20 structures.

A list of filter values.

FilterValue Structure

Represents a single entry in the list of values for a `FilterExpression`.

Fields

- **Type** – *Required*: UTF-8 string (valid values: COLUMNEXTRACTED | CONSTANT).
The type of filter value.
- **Value** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The value to be associated.

CustomCode Structure

Specifies a transform that uses custom code you provide to perform the data transformation. The output is a collection of `DynamicFrames`.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the transform node.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 50 strings.
The data inputs identified by their node names.
- **Code** – *Required*: UTF-8 string, not more than 51200 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The custom code that is used to perform the data transformation.
- **ClassName** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name defined for the custom code node class.

SparkSQL Structure

Specifies a transform where you enter a SQL query using Spark SQL syntax to transform the data. The output is a single `DynamicFrame`.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the transform node.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 1 or more than 50 strings.
The data inputs identified by their node names. You can associate a table name with each input node to use in the SQL query. The name you choose must meet the Spark SQL naming restrictions.
- **SqlQuery** – *Required*: UTF-8 string, not more than 51200 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A SQL query that must use Spark SQL syntax and return a single data set.

- **SqlAliases** – *Required:* An array of [SqlAlias \(p. 842\)](#) objects, not more than 256 structures.

A list of aliases. An alias allows you to specify what name to use in the SQL for a given input. For example, you have a datasource named "MyDataSource". If you specify `From` as MyDataSource, and `Alias` as `SqlName`, then in your SQL you can do:

```
select * from SqlName
```

and that gets data from MyDataSource.

SqlAlias Structure

Represents a single entry in the list of values for `SqlAliases`.

Fields

- **From** – *Required:* UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A table, or a column in a table.

- **Alias** – *Required:* UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

A temporary name given to a table, or a column in a table.

DropNullFields Structure

Specifies a transform that removes columns from the dataset if all values in the column are 'null'. By default, AWS Glue Studio will recognize null objects, but some values such as empty strings, strings that are "null", -1 integers or other placeholders such as zeros, are not automatically recognized as nulls.

Fields

- **Name** – *Required:* UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name of the transform node.

- **Inputs** – *Required:* An array of UTF-8 strings, not less than 1 or more than 1 strings.

The data inputs identified by their node names.

- **Paths** – An array of UTF-8 strings, not more than 100 strings.

A list of paths in Amazon S3 for the output.

- **NullCheckBoxList** – A [NullCheckBoxList \(p. 843\)](#) object.

A structure that represents whether certain values are recognized as null values for removal.

- **NullTextList** – An array of [NullValueField \(p. 843\)](#) objects, not more than 50 structures.

A structure that specifies a list of NullValueField structures that represent a custom null value such as zero or other value being used as a null placeholder unique to the dataset.

The `DropNullFields` transform removes custom null values only if both the value of the null placeholder and the datatype match the data.

NullCheckBoxList Structure

Represents whether certain values are recognized as null values for removal.

Fields

- **IsEmpty** – Boolean.
Specifies that an empty string is considered as a null value.
- **IsNullString** – Boolean.
Specifies that a value spelling out the word 'null' is considered as a null value.
- **IsNegOne** – Boolean.
Specifies that an integer value of -1 is considered as a null value.

NullValueField Structure

Represents a custom null value such as a zeros or other value being used as a null placeholder unique to the dataset.

Fields

- **Value** – *Required:* UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The value of the null placeholder.
- **Datatype** – *Required:* A [Datatype \(p. 843\)](#) object.
The datatype of the value.

Datatype Structure

A structure representing the datatype of the value.

Fields

- **Id** – *Required:* UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The datatype of the value.
- **Label** – *Required:* UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
A label assigned to the datatype.

Merge Structure

Specifies a transform that merges a `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.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the transform node.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 2 or more than 2 strings.
The data inputs identified by their node names.
- **Source** – *Required*: UTF-8 string, not more than 256 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The source `DynamicFrame` that will be merged with a staging `DynamicFrame`.
- **PrimaryKeys** – *Required*: An array of UTF-8 strings, not more than 100 strings.
The list of primary key fields to match records from the source and staging dynamic frames.

Union Structure

Specifies a transform that combines the rows from two or more datasets into a single result.

Fields

- **Name** – *Required*: UTF-8 string, not more than 100 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
The name of the transform node.
- **Inputs** – *Required*: An array of UTF-8 strings, not less than 2 or more than 2 strings.
The node ID inputs to the transform.
- **Sources** – An array of UTF-8 strings, not less than 2 or more than 2 strings.
A list of two or more data sources that have the same schema.
- **UnionType** – *Required*: UTF-8 string (valid values: `ALL` | `DISTINCT`).
Indicates the type of Union transform.
 - Specify `ALL` to join all rows from data sources to the resulting `DynamicFrame`. The resulting union does not remove duplicate rows.
 - Specify `DISTINCT` to remove duplicate rows in the resulting `DynamicFrame`.

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 \(p. 845\)](#)
- [Job Runs \(p. 856\)](#)
- [Triggers \(p. 866\)](#)

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 \(p. 845\)](#)
- [ExecutionProperty Structure \(p. 847\)](#)
- [NotificationProperty Structure \(p. 847\)](#)
- [JobCommand Structure \(p. 847\)](#)
- [ConnectionsList Structure \(p. 848\)](#)
- [JobUpdate Structure \(p. 848\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 847\)](#) object.

An **ExecutionProperty** specifying the maximum number of concurrent runs allowed for this job.

- **Command** – A [JobCommand \(p. 847\)](#) object.

The **JobCommand** that runs 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.

- **NonOverridableArguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

Non-overridable arguments for this job, specified as name-value pairs.

- **Connections** – A [ConnectionsList \(p. 848\)](#) 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 a minimum of 2 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).

For Glue version 1.0 or earlier jobs, using the standard worker type, 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, an Apache Spark ETL job, or an Apache Spark streaming 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"`) or Apache Spark streaming ETL job (`JobCommand.Name="gluestreaming"`), you can allocate a minimum of 2 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.

For Glue version 2.0 jobs, you cannot instead specify a **Maximum capacity**. Instead, you should specify a **Worker type** and the **Number of workers**.

- **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.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the `SecurityConfiguration` structure to be used with this job.

- `NotificationProperty` – A [NotificationProperty \(p. 847\)](#) 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 #15 \(p. 980\)](#).

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 that runs 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`. For an Apache Spark streaming ETL job, this must be `gluestreaming`.

- **ScriptLocation** – UTF-8 string, not more than 400000 bytes long.

Specifies the Amazon Simple Storage Service (Amazon S3) path to a script that runs a job.

- **PythonVersion** – UTF-8 string, matching the [Custom string pattern #16 \(p. 980\)](#).

The Python version being used to run 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 \(p. 980\)](#).

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 \(p. 847\)](#) object.

An `ExecutionProperty` specifying the maximum number of concurrent runs allowed for this job.

- **Command** – A [JobCommand \(p. 847\)](#) object.

The `JobCommand` that runs 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.

- **NonOverridableArguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

Non-overridable arguments for this job, specified as name-value pairs.

- **Connections** – A [ConnectionsList \(p. 848\)](#) 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 a minimum of 2 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).

For Glue version 1.0 or earlier jobs, using the standard worker type, 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"`) or Apache Spark streaming ETL job (`JobCommand.Name="gluestreaming"`), you can allocate a minimum of 2 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.

For Glue version 2.0 jobs, you cannot instead specify a **Maximum capacity**. Instead, you should specify a **Worker type** and the **Number of workers**.

- **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.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the `SecurityConfiguration` structure to be used with this job.

- `NotificationProperty` – A [NotificationProperty \(p. 847\)](#) 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 #15 \(p. 980\)](#).

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 \(Python: create_job\) \(p. 850\)](#)
- [UpdateJob Action \(Python: update_job\) \(p. 853\)](#)
- [GetJob Action \(Python: get_job\) \(p. 853\)](#)
- [GetJobs Action \(Python: get_jobs\) \(p. 854\)](#)
- [DeleteJob Action \(Python: delete_job\) \(p. 854\)](#)
- [ListJobs Action \(Python: list_jobs\) \(p. 855\)](#)
- [BatchGetJobs Action \(Python: batch_get_jobs\) \(p. 856\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 847\)](#) object.

An [ExecutionProperty](#) specifying the maximum number of concurrent runs allowed for this job.

- **Command** – *Required:* A [JobCommand \(p. 847\)](#) object.

The [JobCommand](#) that runs 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.

- **NonOverridableArguments** – A map array of key-value pairs.

Each key is a UTF-8 string.

Each value is a UTF-8 string.

Non-overridable arguments for this job, specified as name-value pairs.

- **Connections** – A [ConnectionsList \(p. 848\)](#) 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 a minimum of 2 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).

For Glue version 1.0 or earlier jobs, using the standard worker type, 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!"`) or Apache Spark streaming ETL job (`JobCommand.Name="gluestreaming"`), you can allocate a minimum of 2 DPUs. The default is 10 DPUs. This job type cannot have a fractional DPU allocation.

For Glue version 2.0 jobs, you cannot instead specify a `MaximumCapacity`. Instead, you should specify a `WorkerType` and the `NumberOfWorkers`.

- `SecurityConfiguration` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 847\)](#) 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 #15 \(p. 980\)](#).

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.

- `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 DPUs (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 \(p. 980\)](#).

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 \(p. 980\)](#).
- The name of the job definition to update.

- `JobUpdate` – *Required*: A [JobUpdate \(p. 848\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the job definition to retrieve.

Response

- Job – A [Job \(p. 845\)](#) 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 \(p. 845\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 use 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 \(p. 845\)](#) 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 \(p. 856\)](#)
- [Predecessor Structure \(p. 859\)](#)
- [JobBookmarkEntry Structure \(p. 859\)](#)
- [BatchStopJobRunSuccessfulSubmission Structure \(p. 860\)](#)
- [BatchStopJobRunError Structure \(p. 860\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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. For more information about the statuses of jobs that have terminated abnormally, see [AWS Glue Job Run Statuses](#).

- **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 \(p. 859\)](#) 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 a minimum of 2 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.

- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 847\)](#) 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 #15 \(p. 980\)](#).

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.

- **DPUSeconds** – Number (double).

This field populates only when an Auto Scaling job run completes, and represents the total time each executor ran during the lifecycle of a job run in seconds, multiplied by a DPU factor (1 for G.1X and 2 for G.2X workers).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The **JobRunId** of the job run in question.

- **ErrorDetail** – An [ErrorDetail \(p. 975\)](#) object.

Specifies details about the error that was encountered.

Operations

- [StartJobRun Action \(Python: start_job_run\) \(p. 860\)](#)
- [BatchStopJobRun Action \(Python: batch_stop_job_run\) \(p. 862\)](#)
- [GetJobRun Action \(Python: get_job_run\) \(p. 863\)](#)
- [GetJobRuns Action \(Python: get_job_runs\) \(p. 863\)](#)
- [GetJobBookmark Action \(Python: get_job_bookmark\) \(p. 864\)](#)
- [GetJobBookmarks Action \(Python: get_job_bookmarks\) \(p. 865\)](#)
- [ResetJobBookmark Action \(Python: reset_job_bookmark\) \(p. 865\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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. You can allocate a minimum of 2 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 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 a minimum of 2 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 \(p. 980\)](#).

The name of the `SecurityConfiguration` structure to be used with this job run.

- `NotificationProperty` – A [NotificationProperty \(p. 847\)](#) 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.

Response

- `JobRunId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 860\)](#) objects.

A list of the JobRuns that were successfully submitted for stopping.

- **Errors** – An array of [BatchStopJobRunError \(p. 860\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).
The ID of the job run.
- **PredecessorsIncluded** – Boolean.
True if a list of predecessor runs should be returned.

Response

- **JobRun** – A [JobRun \(p. 856\)](#) 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 \(p. 980\)](#).

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 \(p. 856\)](#) 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

- `InvalidArgumentException`
- `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 \(p. 859\)](#) object.

A structure that defines a point that a job can resume processing.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 859\)](#) 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 \(p. 859\)](#) object.
The reset bookmark entry.

Errors

- [EntityNotFoundException](#)
- [InvalidInputException](#)
- [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 \(p. 866\)](#)
- [TriggerUpdate Structure \(p. 867\)](#)
- [Predicate Structure \(p. 867\)](#)
- [Condition Structure \(p. 868\)](#)
- [Action Structure \(p. 868\)](#)
- [EventBatchingCondition Structure \(p. 869\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

Reserved for future use.

- Type – UTF-8 string (valid values: SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT).

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 \(p. 980\)](#).

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 \(p. 868\)](#) objects.

The actions initiated by this trigger.

- **Predicate** – A [Predicate \(p. 867\)](#) object.

The predicate of this trigger, which defines when it will fire.

- **EventBatchingCondition** – An [EventBatchingCondition \(p. 869\)](#) object.

Batch condition that must be met (specified number of events received or batch time window expired) before EventBridge event trigger fires.

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 \(p. 980\)](#).

Reserved for future use.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

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 \(p. 868\)](#) objects.

The actions initiated by this trigger.

- **Predicate** – A [Predicate \(p. 867\)](#) object.

The predicate of this trigger, which defines when it will fire.

- **EventBatchingCondition** – An [EventBatchingCondition \(p. 869\)](#) object.

Batch condition that must be met (specified number of events received or batch time window expired) before EventBridge event trigger fires.

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 \(p. 868\)](#) 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 \(p. 980\)](#).
The name of the job whose `JobRuns` this condition applies to, and on which this trigger waits.

The condition state. Currently, the only job states that a trigger can listen for are `SUCCEEDED`, `STOPPED`, `FAILED`, and `TIMEOUT`. The only crawler states that a trigger can listen for are `SUCCEEDED`, `FAILED`, and `CANCELLED`.

- **CrawlerName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of the crawler to which this condition applies.

- **CrawlState** – UTF-8 string (valid values: `RUNNING` | `CANCELLING` | `CANCELLED` | `SUCCEEDED` | `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 \(p. 980\)](#).
The name of a job to be 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 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 \(p. 980\)](#).

The name of the SecurityConfiguration structure to be used with this action.

- **NotificationProperty** – A [NotificationProperty \(p. 847\)](#) 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 \(p. 980\)](#).

The name of the crawler to be used with this action.

EventBatchingCondition Structure

Batch condition that must be met (specified number of events received or batch time window expired) before EventBridge event trigger fires.

Fields

- **BatchSize** – *Required:* Number (integer), not less than 1 or more than 100.

Number of events that must be received from Amazon EventBridge before EventBridge event trigger fires.

- **BatchWindow** – Number (integer), not less than 1 or more than 900.

Window of time in seconds after which EventBridge event trigger fires. Window starts when first event is received.

Operations

- [CreateTrigger Action \(Python: create_trigger\) \(p. 869\)](#)
- [StartTrigger Action \(Python: start_trigger\) \(p. 871\)](#)
- [GetTrigger Action \(Python: get_trigger\) \(p. 871\)](#)
- [GetTriggers Action \(Python: get_triggers\) \(p. 872\)](#)
- [UpdateTrigger Action \(Python: update_trigger\) \(p. 872\)](#)
- [StopTrigger Action \(Python: stop_trigger\) \(p. 873\)](#)
- [DeleteTrigger Action \(Python: delete_trigger\) \(p. 874\)](#)
- [ListTriggers Action \(Python: list_triggers\) \(p. 874\)](#)
- [BatchGetTriggers Action \(Python: batch_get_triggers\) \(p. 875\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the workflow associated with the trigger.

- **Type** – *Required*: UTF-8 string (valid values: SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT).

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 \(p. 867\)](#) 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 \(p. 868\)](#) 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 \(p. 980\)](#).

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.

- **EventBatchingCondition** – An [EventBatchingCondition \(p. 869\)](#) object.

Batch condition that must be met (specified number of events received or batch time window expired) before EventBridge event trigger fires.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The name of the trigger that was started.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentRunsExceededException`

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 \(p. 980\)](#).

The name of the trigger to retrieve.

Response

- **Trigger** – A [Trigger \(p. 866\)](#) 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 \(p. 980\)](#).

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 \(p. 866\)](#) 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 \(p. 980\)](#).

The name of the trigger to update.

- **TriggerUpdate** – *Required*: A [TriggerUpdate \(p. 867\)](#) object.

The new values with which to update the trigger.

Response

- **Trigger** – A [Trigger \(p. 866\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 866\)](#) 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`

Interactive sessions API

The interactive sessions API describes the AWS Glue API related to using AWS Glue interactive sessions to build and test extract, transform, and load (ETL) scripts for data integration.

Data Types

- [Session Structure \(p. 876\)](#)

- [SessionCommand Structure \(p. 877\)](#)
- [Statement Structure \(p. 877\)](#)
- [StatementOutput Structure \(p. 878\)](#)
- [StatementOutputData Structure \(p. 878\)](#)

Session Structure

The period in which a remote Spark runtime environment is running.

Fields

- **Id** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the session.

- **CreatedOn** – Timestamp.

The time and date when the session was created.

- **Status** – UTF-8 string (valid values: PROVISIONING | READY | FAILED | TIMEOUT | STOPPING | STOPPED).

The session status.

- **ErrorMessage** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The error message displayed during the session.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The description of the session.

- **Role** – UTF-8 string, not less than 20 or more than 2048 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The name or Amazon Resource Name (ARN) of the IAM role associated with the Session.

- **Command** – A [SessionCommand \(p. 877\)](#) object.

The command object.See [SessionCommand](#).

- **DefaultArguments** – A map array of key-value pairs, not more than 75 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 4096 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A map array of key-value pairs. Max is 75 pairs.

- **Connections** – A [ConnectionsList \(p. 848\)](#) object.

The number of connections used for the session.

- **Progress** – Number (double).

The code execution progress of the session.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when the job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB memory.

- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the SecurityConfiguration structure to be used with the session.

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #15 \(p. 980\)](#).

The AWS Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The GlueVersion must be greater than 2.0.

- **DataAccessId** – UTF-8 string, not less than 1 or more than 36 bytes long.

The data access ID of the session.

- **PartitionId** – UTF-8 string, not less than 1 or more than 36 bytes long.

The partition ID of the sesion.

SessionCommand Structure

The SessionCommand that runs the job.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Specifies the name of the SessionCommand. Can be 'glueetl' or 'gluestreaming'.

- **PythonVersion** – UTF-8 string, matching the [Custom string pattern #16 \(p. 980\)](#).

Specifies the Python version. The Python version indicates the version supported for jobs of type Spark.

Statement Structure

The statement or request for a particular action to occur in a session.

Fields

- **Id** – Number (integer).

The ID of the statement.

- **Code** – UTF-8 string.

The execution code of the statement.

- **State** – UTF-8 string (valid values: WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR).

The state while request is actioned.

- **Output** – A [StatementOutput \(p. 878\)](#) object.

The output in JSON.

- **Progress** – Number (double).
The code execution progress.
- **StartedOn** – Number (long).
The unix time and date that the job definition was started.
- **CompletedOn** – Number (long).
The unix time and date that the job definition was completed.

StatementOutput Structure

The code execution output in JSON format.

Fields

- **Data** – A [StatementOutputData \(p. 878\)](#) object.
The code execution output.
- **ExecutionCount** – Number (integer).
The execution count of the output.
- **Status** – UTF-8 string (valid values: WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR).
The status of the code execution output.
- **ErrorName** – UTF-8 string.
The name of the error in the output.
- **ErrorMessage** – UTF-8 string.
The error value of the output.
- **Traceback** – An array of UTF-8 strings.
The traceback of the output.

StatementOutputData Structure

The code execution output in JSON format.

Fields

- **TextPlain** – UTF-8 string.
The code execution output in text format.

Operations

- [CreateSession Action \(Python: create_session\) \(p. 879\)](#)
- [StopSession Action \(Python: stop_session\) \(p. 881\)](#)
- [DeleteSession Action \(Python: delete_session\) \(p. 881\)](#)
- [GetSession Action \(Python: get_session\) \(p. 882\)](#)

- [ListSessions Action \(Python: list_sessions\) \(p. 882\)](#)
- [RunStatement Action \(Python: run_statement\) \(p. 883\)](#)
- [CancelStatement Action \(Python: cancel_statement\) \(p. 884\)](#)
- [GetStatement Action \(Python: get_statement\) \(p. 884\)](#)
- [ListStatements Action \(Python: list_statements\) \(p. 885\)](#)

CreateSession Action (Python: create_session)

Creates a new session.

Request

Request to create a new session.

- **Id** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the session request.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The description of the session.

- **Role** – *Required*: UTF-8 string, not less than 20 or more than 2048 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The IAM Role ARN

- **Command** – *Required*: A [SessionCommand \(p. 877\)](#) object.

The SessionCommand that runs the job.

- **Timeout** – Number (integer), at least 1.

The number of seconds before request times out.

- **IdleTimeout** – Number (integer), at least 1.

The number of seconds when idle before request times out.

- **DefaultArguments** – A map array of key-value pairs, not more than 75 pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

Each value is a UTF-8 string, not more than 4096 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A map array of key-value pairs. Max is 75 pairs.

- **Connections** – A [ConnectionsList \(p. 848\)](#) object.

The number of connections to use for the session.

- **MaxCapacity** – Number (double).

The number of AWS Glue data processing units (DPUs) that can be allocated when the job runs. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB memory.

- **NumberOfWorkers** – Number (integer).

The number of workers of a defined WorkerType to use for the session.

- **WorkerType** – UTF-8 string (valid values: Standard="" | G.1X="" | G.2X="").

The type of predefined worker that is allocated to use for the session. 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.
- **SecurityConfiguration** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the SecurityConfiguration structure to be used with the session

- **GlueVersion** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #15 \(p. 980\)](#).

The AWS Glue version determines the versions of Apache Spark and Python that AWS Glue supports. The GlueVersion must be greater than 2.0.

- **DataAccessId** – UTF-8 string, not less than 1 or more than 36 bytes long.

The data access ID of the session.

- **PartitionId** – UTF-8 string, not less than 1 or more than 36 bytes long.

The partition ID of the session.

- **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 map of key value pairs (tags) belonging to the session.

- **RequestOrigin** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The origin of the request.

Response

- **Session** – A [Session \(p. 876\)](#) object.

Returns the session object in the response.

Errors

- **AccessDeniedException**
- **IdempotentParameterMismatchException**
- **InternalServiceException**
- **OperationTimeoutException**
- **InvalidArgumentException**
- **ValidationException**
- **AlreadyExistsException**

- `ResourceNumberLimitExceeded`

StopSession Action (Python: stop_session)

Stops the session.

Request

- `Id` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the session to be stopped.
- `RequestOrigin` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The origin of the request.

Response

- `Id` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
Returns the Id of the stopped session.

Errors

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`
- `ConcurrentModificationException`

DeleteSession Action (Python: delete_session)

Deletes the session.

Request

- `Id` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the session to be deleted.
- `RequestOrigin` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The name of the origin of the delete session request.

Response

- `Id` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Returns the ID of the deleted session.

Errors

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`
- `ConcurrentModificationException`

GetSession Action (Python: get_session)

Retrieves the session.

Request

- `Id` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the session.

- `RequestOrigin` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The origin of the request.

Response

- `Session` – A [Session \(p. 876\)](#) object.

The session object is returned in the response.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

ListSessions Action (Python: list_sessions)

Retrieve a list of sessions.

Request

- `NextToken` – UTF-8 string, not more than 400000 bytes long.

The token for the next set of results, or null if there are no more result.

- `MaxResults` – Number (integer), not less than 1 or more than 1000.

The maximum number of results.

- **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.

Tags belonging to the session.

- **RequestOrigin** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The origin of the request.

Response

- **Ids** – An array of UTF-8 strings.

Returns the ID of the session.

- **Sessions** – An array of [Session \(p. 876\)](#) objects.

Returns the session object.

- **NextToken** – UTF-8 string, not more than 400000 bytes long.

The token for the next set of results, or null if there are no more result.

Errors

- `AccessDeniedException`
- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

RunStatement Action (Python: run_statement)

Executes the statement.

Request

- **SessionId** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The Session Id of the statement to be run.

- **Code** – *Required:* UTF-8 string, not more than 68000 bytes long.

The statement code to be run.

- **RequestOrigin** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The origin of the request.

Response

- **Id** – Number (integer).

Returns the Id of the statement that was run.

Errors

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ValidationException`
- `ResourceNumberLimitExceededException`
- `IllegalSessionStateException`

CancelStatement Action (Python: cancel_statement)

Cancels the statement.

Request

- `SessionId` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The Session ID of the statement to be cancelled.

- `Id` – *Required*: Number (integer).

The ID of the statement to be cancelled.

- `RequestOrigin` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The origin of the request to cancel the statement.

Response

- *No Response parameters.*

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

GetStatement Action (Python: get_statement)

Retrieves the statement.

Request

- **SessionId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The Session ID of the statement.
- **Id** – *Required*: Number (integer).
The Id of the statement.
- **RequestOrigin** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The origin of the request.

Response

- **Statement** – A [Statement \(p. 877\)](#) object.
Returns the statement.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidArgumentException`
- `IllegalSessionStateException`

ListStatements Action (Python: list_statements)

Lists statements for the session.

Request

- **SessionId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The Session ID of the statements.
- **RequestOrigin** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The origin of the request to list statements.
- **NextToken** – UTF-8 string, not more than 400000 bytes long.
A continuation token, if this is a continuation call.

Response

- **Statements** – An array of [Statement \(p. 877\)](#) objects.
Returns the list of statements.
- **NextToken** – UTF-8 string, not more than 400000 bytes long.

A continuation token, if not all statements have yet been returned.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

Development Endpoints API

The Development Endpoints API describes the AWS Glue API related to testing using a custom DevEndpoint.

Data Types

- [DevEndpoint Structure \(p. 886\)](#)
- [DevEndpointCustomLibraries Structure \(p. 888\)](#)

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 \(p. 980\)](#).
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 #15 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(Python: create_dev_endpoint\) \(p. 889\)](#)
- [UpdateDevEndpoint Action \(Python: update_dev_endpoint\) \(p. 893\)](#)
- [DeleteDevEndpoint Action \(Python: delete_dev_endpoint\) \(p. 894\)](#)
- [GetDevEndpoint Action \(Python: get_dev_endpoint\) \(p. 894\)](#)
- [GetDevEndpoints Action \(Python: get_dev_endpoints\) \(p. 895\)](#)
- [BatchGetDevEndpoints Action \(Python: batch_get_dev_endpoints\) \(p. 895\)](#)
- [ListDevEndpoints Action \(Python: list_dev_endpoints\) \(p. 896\)](#)

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 \(p. 980\)](#).

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 #15 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 #15 \(p. 980\)](#).

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 \(p. 980\)](#).

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`
- `InvalidArgumentException`
- `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 \(p. 888\)](#) 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. 886) object.

A DevEndpoint definition.

Errors

- `EntityNotFoundException`
- `InternalServiceException`

- `OperationTimeoutException`
- `InvalidInputException`

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 \(p. 886\)](#) 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`
- `InvalidInputException`

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 \(p. 886\)](#) 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`

Schema Registry

The Schema Registry API describes the data types and API related to working with schemas in AWS Glue.

Data Types

- [RegistryId Structure \(p. 897\)](#)
- [RegistryListItem Structure \(p. 897\)](#)
- [MetadataInfo Structure \(p. 898\)](#)
- [OtherMetadataValueListItem Structure \(p. 898\)](#)
- [SchemaListItem Structure \(p. 899\)](#)
- [SchemaVersionListItem Structure \(p. 899\)](#)
- [MetadataKeyValuePair Structure \(p. 900\)](#)
- [SchemaVersionErrorItem Structure \(p. 900\)](#)
- [ErrorDetails Structure \(p. 900\)](#)
- [SchemaVersionNumber Structure \(p. 900\)](#)
- [SchemaId Structure \(p. 901\)](#)

RegistryId Structure

A wrapper structure that may contain the registry name and Amazon Resource Name (ARN).

Fields

- `RegistryName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).
Name of the registry. Used only for lookup. One of `RegistryArn` or `RegistryName` has to be provided.
- `RegistryArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).
`Arn` of the registry to be updated. One of `RegistryArn` or `RegistryName` has to be provided.

RegistryListItem Structure

A structure containing the details for a registry.

Fields

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the registry.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of the registry.

- **Status** – UTF-8 string (valid values: `AVAILABLE` | `DELETING`).

The status of the registry.

- **CreatedTime** – UTF-8 string.

The date the registry was created.

- **UpdatedTime** – UTF-8 string.

The date the registry was updated.

MetadataInfo Structure

A structure containing metadata information for a schema version.

Fields

- **MetadataValue** – UTF-8 string, not less than 1 or more than 256 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

The metadata key's corresponding value.

- **CreatedTime** – UTF-8 string.

The time at which the entry was created.

- **OtherMetadataValueList** – An array of [OtherMetadataValueListItem \(p. 898\)](#) objects.

Other metadata belonging to the same metadata key.

OtherMetadataValueListItem Structure

A structure containing other metadata for a schema version belonging to the same metadata key.

Fields

- **MetadataValue** – UTF-8 string, not less than 1 or more than 256 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

The metadata key's corresponding value for the other metadata belonging to the same metadata key.

- **CreatedTime** – UTF-8 string.

The time at which the entry was created.

SchemaListItem Structure

An object that contains minimal details for a schema.

Fields

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).
the name of the registry where the schema resides.
- **SchemaName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).
The name of the schema.
- **SchemaArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).
The Amazon Resource Name (ARN) for the schema.
- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).
A description for the schema.
- **SchemaStatus** – UTF-8 string (valid values: AVAILABLE | PENDING | DELETING).
The status of the schema.
- **CreatedTime** – UTF-8 string.
The date and time that a schema was created.
- **UpdatedTime** – UTF-8 string.
The date and time that a schema was updated.

SchemaVersionListItem Structure

An object containing the details about a schema version.

Fields

- **SchemaArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).
The Amazon Resource Name (ARN) of the schema.
- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).
The unique identifier of the schema version.
- **VersionNumber** – Number (long), not less than 1 or more than 100000.
The version number of the schema.
- **Status** – UTF-8 string (valid values: AVAILABLE | PENDING | FAILURE | DELETING).
The status of the schema version.
- **CreatedTime** – UTF-8 string.

The date and time the schema version was created.

MetadataKeyValuePair Structure

A structure containing a key value pair for metadata.

Fields

- `MetadataKey` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).
A metadata key.
- `MetadataValue` – UTF-8 string, not less than 1 or more than 256 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).
A metadata key's corresponding value.

SchemaVersionErrorItem Structure

An object that contains the error details for an operation on a schema version.

Fields

- `VersionNumber` – Number (long), not less than 1 or more than 100000.
The version number of the schema.
- `ErrorDetails` – An [ErrorDetails \(p. 900\)](#) object.
The details of the error for the schema version.

ErrorDetails Structure

An object containing error details.

Fields

- `ErrorCode` – UTF-8 string.
The error code for an error.
- `ErrorMessage` – UTF-8 string.
The error message for an error.

SchemaVersionNumber Structure

A structure containing the schema version information.

Fields

- `LatestVersion` – Boolean.

The latest version available for the schema.

- `VersionNumber` – Number (long), not less than 1 or more than 100000.

The version number of the schema.

Schemald Structure

The unique ID of the schema in the AWS Glue schema registry.

Fields

- `SchemaArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema. One of `SchemaArn` or `SchemaName` has to be provided.

- `SchemaName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema. One of `SchemaArn` or `SchemaName` has to be provided.

- `RegistryName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema registry that contains the schema.

Operations

- [CreateRegistry Action \(Python: create_registry\) \(p. 902\)](#)
- [CreateSchema Action \(Python: create_schema\) \(p. 903\)](#)
- [GetSchema Action \(Python: get_schema\) \(p. 905\)](#)
- [ListSchemaVersions Action \(Python: list_schema_versions\) \(p. 907\)](#)
- [GetSchemaVersion Action \(Python: get_schema_version\) \(p. 907\)](#)
- [GetSchemaVersionsDiff Action \(Python: get_schema_versions_diff\) \(p. 908\)](#)
- [ListRegistries Action \(Python: list_registries\) \(p. 909\)](#)
- [ListSchemas Action \(Python: list_schemas\) \(p. 910\)](#)
- [RegisterSchemaVersion Action \(Python: register_schema_version\) \(p. 911\)](#)
- [UpdateSchema Action \(Python: update_schema\) \(p. 912\)](#)
- [CheckSchemaVersionValidity Action \(Python: check_schema_version_validity\) \(p. 913\)](#)
- [UpdateRegistry Action \(Python: update_registry\) \(p. 913\)](#)
- [GetSchemaByDefinition Action \(Python: get_schema_by_definition\) \(p. 914\)](#)
- [GetRegistry Action \(Python: get_registry\) \(p. 915\)](#)
- [PutSchemaVersionMetadata Action \(Python: put_schema_version_metadata\) \(p. 916\)](#)
- [QuerySchemaVersionMetadata Action \(Python: query_schema_version_metadata\) \(p. 917\)](#)
- [RemoveSchemaVersionMetadata Action \(Python: remove_schema_version_metadata\) \(p. 918\)](#)
- [DeleteRegistry Action \(Python: delete_registry\) \(p. 919\)](#)
- [DeleteSchema Action \(Python: delete_schema\) \(p. 920\)](#)

- [DeleteSchemaVersions Action \(Python: delete_schema_versions\) \(p. 920\)](#)

CreateRegistry Action (Python: create_registry)

Creates a new registry which may be used to hold a collection of schemas.

Request

- **RegistryName** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

Name of the registry to be created of max length of 255, and may only contain letters, numbers, hyphen, underscore, dollar sign, or hash mark. No whitespace.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of the registry. If description is not provided, there will not be any default value for this.

- **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.

AWS tags that contain a key value pair and may be searched by console, command line, or API.

Response

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the newly created registry.

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of the registry.

- **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 for the registry.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

- InternalServiceException

CreateSchema Action (Python: create_schema)

Creates a new schema set and registers the schema definition. Returns an error if the schema set already exists without actually registering the version.

When the schema set is created, a version checkpoint will be set to the first version. Compatibility mode "DISABLED" restricts any additional schema versions from being added after the first schema version. For all other compatibility modes, validation of compatibility settings will be applied only from the second version onwards when the RegisterSchemaVersion API is used.

When this API is called without a RegistryId, this will create an entry for a "default-registry" in the registry database tables, if it is not already present.

Request

- RegistryId – A [RegistryId \(p. 897\)](#) object.

This is a wrapper shape to contain the registry identity fields. If this is not provided, the default registry will be used. The ARN format for the same will be: arn:aws:glue:us-east-2:<customer id>:registry/default-registry:random-5-letter-id.

- SchemaName – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

Name of the schema to be created of max length of 255, and may only contain letters, numbers, hyphen, underscore, dollar sign, or hash mark. No whitespace.

- DataFormat – *Required:* UTF-8 string (valid values: AVRO | JSON | PROTOBUF).

The data format of the schema definition. Currently AVRO and JSON are supported.

- Compatibility – UTF-8 string (valid values: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

The compatibility mode of the schema. The possible values are:

- *NONE:* No compatibility mode applies. You can use this choice in development scenarios or if you do not know the compatibility mode that you want to apply to schemas. Any new version added will be accepted without undergoing a compatibility check.
- *DISABLED:* This compatibility choice prevents versioning for a particular schema. You can use this choice to prevent future versioning of a schema.
- *BACKWARD:* This compatibility choice is recommended as it allows data receivers to read both the current and one previous schema version. This means that for instance, a new schema version cannot drop data fields or change the type of these fields, so they can't be read by readers using the previous version.
- *BACKWARD_ALL:* This compatibility choice allows data receivers to read both the current and all previous schema versions. You can use this choice when you need to delete fields or add optional fields, and check compatibility against all previous schema versions.
- *FORWARD:* This compatibility choice allows data receivers to read both the current and one next schema version, but not necessarily later versions. You can use this choice when you need to add fields or delete optional fields, but only check compatibility against the last schema version.
- *FORWARD_ALL:* This compatibility choice allows data receivers to read written by producers of any new registered schema. You can use this choice when you need to add fields or delete optional fields, and check compatibility against all previous schema versions.
- *FULL:* This compatibility choice allows data receivers to read data written by producers using the previous or next version of the schema, but not necessarily earlier or later versions. You can use this

choice when you need to add or remove optional fields, but only check compatibility against the last schema version.

- **FULL_ALL**: This compatibility choice allows data receivers to read data written by producers using all previous schema versions. You can use this choice when you need to add or remove optional fields, and check compatibility against all previous schema versions.
- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

An optional description of the schema. If description is not provided, there will not be any automatic default value for this.

- **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.

AWS tags that contain a key value pair and may be searched by console, command line, or API. If specified, follows the AWS tags-on-create pattern.

- **SchemaDefinition** – UTF-8 string, not less than 1 or more than 170000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).

The schema definition using the `DataFormat` setting for `SchemaName`.

Response

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the registry.

- **SchemaName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema.

- **SchemaArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of the schema if specified when created.

- **DataFormat** – UTF-8 string (valid values: `AVRO` | `JSON` | `PROTOBUF`).

The data format of the schema definition. Currently `AVRO` and `JSON` are supported.

- **Compatibility** – UTF-8 string (valid values: `NONE` | `DISABLED` | `BACKWARD` | `BACKWARD_ALL` | `FORWARD` | `FORWARD_ALL` | `FULL` | `FULL_ALL`).

The schema compatibility mode.

- **SchemaCheckpoint** – Number (long), not less than 1 or more than 100000.

The version number of the checkpoint (the last time the compatibility mode was changed).

- **LatestSchemaVersion** – Number (long), not less than 1 or more than 100000.
The latest version of the schema associated with the returned schema definition.
- **NextSchemaVersion** – Number (long), not less than 1 or more than 100000.
The next version of the schema associated with the returned schema definition.
- **SchemaStatus** – UTF-8 string (valid values: AVAILABLE | PENDING | DELETING).
The status of the schema.
- **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 for the schema.
- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).
The unique identifier of the first schema version.
- **SchemaVersionStatus** – UTF-8 string (valid values: AVAILABLE | PENDING | FAILURE | DELETING).
The status of the first schema version created.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`
- `InternalServiceException`

GetSchema Action (Python: get_schema)

Describes the specified schema in detail.

Request

- **SchemaId** – *Required:* A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- **SchemaId\$SchemaArn**: The Amazon Resource Name (ARN) of the schema. Either `SchemaArn` or `SchemaName` and `RegistryName` has to be provided.
- **SchemaId\$SchemaName**: The name of the schema. Either `SchemaArn` or `SchemaName` and `RegistryName` has to be provided.

Response

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- `RegistryArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the registry.

- `SchemaName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema.

- `SchemaArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- `Description` – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of schema if specified when created

- `DataFormat` – UTF-8 string (valid values: AVRO | JSON | PROTOBUF).

The data format of the schema definition. Currently AVRO and JSON are supported.

- `Compatibility` – UTF-8 string (valid values: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

The compatibility mode of the schema.

- `SchemaCheckpoint` – Number (long), not less than 1 or more than 100000.

The version number of the checkpoint (the last time the compatibility mode was changed).

- `LatestSchemaVersion` – Number (long), not less than 1 or more than 100000.

The latest version of the schema associated with the returned schema definition.

- `NextSchemaVersion` – Number (long), not less than 1 or more than 100000.

The next version of the schema associated with the returned schema definition.

- `SchemaStatus` – UTF-8 string (valid values: AVAILABLE | PENDING | DELETING).

The status of the schema.

- `CreatedTime` – UTF-8 string.

The date and time the schema was created.

- `UpdatedTime` – UTF-8 string.

The date and time the schema was updated.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

ListSchemaVersions Action (Python: list_schema_versions)

Returns a list of schema versions that you have created, with minimal information. Schema versions in Deleted status will not be included in the results. Empty results will be returned if there are no schema versions available.

Request

- **SchemaId** – *Required:* A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- **SchemaId\$SchemaArn:** The Amazon Resource Name (ARN) of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- **SchemaId\$SchemaName:** The name of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- **MaxResults** – Number (integer), not less than 1 or more than 100.

Maximum number of results required per page. If the value is not supplied, this will be defaulted to 25 per page.

- **NextToken** – UTF-8 string.

A continuation token, if this is a continuation call.

Response

- **Schemas** – An array of [SchemaVersionListItem \(p. 899\)](#) objects.

An array of SchemaVersionList objects containing details of each schema version.

- **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

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

GetSchemaVersion Action (Python: get_schema_version)

Get the specified schema by its unique ID assigned when a version of the schema is created or registered. Schema versions in Deleted status will not be included in the results.

Request

- **SchemaId** – A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- SchemaId\$SchemaArn: The Amazon Resource Name (ARN) of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- SchemaId\$SchemaName: The name of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- SchemaVersionId – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The SchemaVersionId of the schema version. This field is required for fetching by schema ID. Either this or the SchemaId wrapper has to be provided.

- SchemaVersionNumber – A [SchemaVersionNumber \(p. 900\)](#) object.

The version number of the schema.

Response

- SchemaVersionId – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The SchemaVersionId of the schema version.

- SchemaDefinition – UTF-8 string, not less than 1 or more than 170000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).

The schema definition for the schema ID.

- DataFormat – UTF-8 string (valid values: AVRO | JSON | PROTOBUF).

The data format of the schema definition. Currently AVRO and JSON are supported.

- SchemaArn – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- VersionNumber – Number (long), not less than 1 or more than 100000.

The version number of the schema.

- Status – UTF-8 string (valid values: AVAILABLE | PENDING | FAILURE | DELETING).

The status of the schema version.

- CreatedTime – UTF-8 string.

The date and time the schema version was created.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

GetSchemaVersionsDiff Action (Python: `get_schema_versions_diff`)

Fetches the schema version difference in the specified difference type between two stored schema versions in the Schema Registry.

This API allows you to compare two schema versions between two schema definitions under the same schema.

Request

- **SchemaId – Required:** A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- SchemaId\$SchemaArn: The Amazon Resource Name (ARN) of the schema. One of SchemaArn or SchemaName has to be provided.
- SchemaId\$SchemaName: The name of the schema. One of SchemaArn or SchemaName has to be provided.
- **FirstSchemaVersionNumber – Required:** A [SchemaVersionNumber \(p. 900\)](#) object.

The first of the two schema versions to be compared.

- **SecondSchemaVersionNumber – Required:** A [SchemaVersionNumber \(p. 900\)](#) object.

The second of the two schema versions to be compared.

- **SchemaDiffType – Required:** UTF-8 string (valid values: SYNTAX_DIFF).

Refers to SYNTAX_DIFF, which is the currently supported diff type.

Response

- **Diff –** UTF-8 string, not less than 1 or more than 340000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).

The difference between schemas as a string in JsonPatch format.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`

ListRegistries Action (Python: list_registries)

Returns a list of registries that you have created, with minimal registry information. Registries in the Deleting status will not be included in the results. Empty results will be returned if there are no registries available.

Request

- **MaxResults –** Number (integer), not less than 1 or more than 100.

Maximum number of results required per page. If the value is not supplied, this will be defaulted to 25 per page.

- **NextToken –** UTF-8 string.

A continuation token, if this is a continuation call.

Response

- **Registries** – An array of [RegistryListItem \(p. 897\)](#) objects.
An array of `RegistryDetailedListItem` objects containing minimal details of each registry.
- **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

- `InvalidArgumentException`
- `AccessDeniedException`
- `InternalServiceException`

ListSchemas Action (Python: list_schemas)

Returns a list of schemas with minimal details. Schemas in Deleting status will not be included in the results. Empty results will be returned if there are no schemas available.

When the `RegistryId` is not provided, all the schemas across registries will be part of the API response.

Request

- **RegistryId** – A [RegistryId \(p. 897\)](#) object.
A wrapper structure that may contain the registry name and Amazon Resource Name (ARN).
- **MaxResults** – Number (integer), not less than 1 or more than 100.
Maximum number of results required per page. If the value is not supplied, this will be defaulted to 25 per page.
- **NextToken** – UTF-8 string.
A continuation token, if this is a continuation call.

Response

- **Schemas** – An array of [SchemaListItem \(p. 899\)](#) objects.
An array of `SchemaListItem` objects containing details of each schema.
- **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

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

RegisterSchemaVersion Action (Python: register_schema_version)

Adds a new version to the existing schema. Returns an error if new version of schema does not meet the compatibility requirements of the schema set. This API will not create a new schema set and will return a 404 error if the schema set is not already present in the Schema Registry.

If this is the first schema definition to be registered in the Schema Registry, this API will store the schema version and return immediately. Otherwise, this call has the potential to run longer than other operations due to compatibility modes. You can call the GetSchemaVersion API with the SchemaVersionId to check compatibility modes.

If the same schema definition is already stored in Schema Registry as a version, the schema ID of the existing schema is returned to the caller.

Request

- SchemaId – *Required*: A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- SchemaId\$SchemaArn: The Amazon Resource Name (ARN) of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- SchemaId\$SchemaName: The name of the schema. Either SchemaArn or SchemaName and RegistryName has to be provided.
- SchemaDefinition – *Required*: UTF-8 string, not less than 1 or more than 170000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).

The schema definition using the DataFormat setting for the SchemaName.

Response

- SchemaVersionId – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The unique ID that represents the version of this schema.

- VersionNumber – Number (long), not less than 1 or more than 100000.

The version of this schema (for sync flow only, in case this is the first version).

- Status – UTF-8 string (valid values: AVAILABLE | PENDING | FAILURE | DELETING).

The status of the schema version.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`
- `InternalServiceException`

UpdateSchema Action (Python: update_schema)

Updates the description, compatibility setting, or version checkpoint for a schema set.

For updating the compatibility setting, the call will not validate compatibility for the entire set of schema versions with the new compatibility setting. If the value for `Compatibility` is provided, the `VersionNumber` (a checkpoint) is also required. The API will validate the checkpoint version number for consistency.

If the value for the `VersionNumber` (checkpoint) is provided, `Compatibility` is optional and this can be used to set/reset a checkpoint for the schema.

This update will happen only if the schema is in the AVAILABLE state.

Request

- `SchemaId` – *Required*: A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- `SchemaId$SchemaArn`: The Amazon Resource Name (ARN) of the schema. One of `SchemaArn` or `SchemaName` has to be provided.
- `SchemaId$SchemaName`: The name of the schema. One of `SchemaArn` or `SchemaName` has to be provided.
- `SchemaVersionNumber` – A [SchemaVersionNumber \(p. 900\)](#) object.

Version number required for check pointing. One of `VersionNumber` or `Compatibility` has to be provided.

- `Compatibility` – UTF-8 string (valid values: `NONE` | `DISABLED` | `BACKWARD` | `BACKWARD_ALL` | `FORWARD` | `FORWARD_ALL` | `FULL` | `FULL_ALL`).

The new compatibility setting for the schema.

- `Description` – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

The new description for the schema.

Response

- `SchemaArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- `SchemaName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema.

- `RegistryName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry that contains the schema.

Errors

- `InvalidArgumentException`

- `AccessDeniedException`
- `EntityNotFoundException`
- `ConcurrentModificationException`
- `InternalServiceException`

CheckSchemaVersionValidity Action (Python: check_schema_version_validity)

Validates the supplied schema. This call has no side effects, it simply validates using the supplied schema using `DataFormat` as the format. Since it does not take a schema set name, no compatibility checks are performed.

Request

- `DataFormat` – *Required*: UTF-8 string (valid values: `AVRO` | `JSON` | `PROTOBUF`).
The data format of the schema definition. Currently `AVRO` and `JSON` are supported.
- `SchemaDefinition` – *Required*: UTF-8 string, not less than 1 or more than 170000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).
The definition of the schema that has to be validated.

Response

- `Valid` – Boolean.
Return true, if the schema is valid and false otherwise.
- `Error` – UTF-8 string, not less than 1 or more than 5000 bytes long.
A validation failure error message.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `InternalServiceException`

UpdateRegistry Action (Python: update_registry)

Updates an existing registry which is used to hold a collection of schemas. The updated properties relate to the registry, and do not modify any of the schemas within the registry.

Request

- `RegistryId` – *Required*: A [RegistryId \(p. 897\)](#) object.
This is a wrapper structure that may contain the registry name and Amazon Resource Name (ARN).
- `Description` – *Required*: Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).
A description of the registry. If description is not provided, this field will not be updated.

Response

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the updated registry.

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource name (ARN) of the updated registry.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `ConcurrentModificationException`
- `InternalServiceException`

GetSchemaByDefinition Action (Python: `get_schema_by_definition`)

Retrieves a schema by the `SchemaDefinition`. The schema definition is sent to the Schema Registry, canonicalized, and hashed. If the hash is matched within the scope of the `SchemaName` or `ARN` (or the default registry, if none is supplied), that schema's metadata is returned. Otherwise, a `404` or `NotFound` error is returned. Schema versions in `Deleted` statuses will not be included in the results.

Request

- **SchemaId** – *Required:* A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure to contain schema identity fields. The structure contains:

- `SchemaId$SchemaArn`: The Amazon Resource Name (ARN) of the schema. One of `SchemaArn` or `SchemaName` has to be provided.
- `SchemaId$SchemaName`: The name of the schema. One of `SchemaArn` or `SchemaName` has to be provided.
- **SchemaDefinition** – *Required:* UTF-8 string, not less than 1 or more than 170000 bytes long, matching the [Custom string pattern #23 \(p. 980\)](#).

The definition of the schema for which schema details are required.

Response

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The schema ID of the schema version.

- **SchemaArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- **DataFormat** – UTF-8 string (valid values: `AVRO` | `JSON` | `PROTOBUF`).

The data format of the schema definition. Currently only AVRO and JSON are supported.

- **Status** – UTF-8 string (valid values: AVAILABLE | PENDING | FAILURE | DELETING).

The status of the schema version.

- **CreatedTime** – UTF-8 string.

The date and time the schema was created.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

GetRegistry Action (Python: get_registry)

Describes the specified registry in detail.

Request

- **RegistryId** – *Required:* A [RegistryId \(p. 897\)](#) object.

This is a wrapper structure that may contain the registry name and Amazon Resource Name (ARN).

Response

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the registry.

- **Description** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

A description of the registry.

- **Status** – UTF-8 string (valid values: AVAILABLE | DELETING).

The status of the registry.

- **CreatedTime** – UTF-8 string.

The date and time the registry was created.

- **UpdatedTime** – UTF-8 string.

The date and time the registry was updated.

Errors

- `InvalidArgumentException`

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

PutSchemaVersionMetadata Action (Python: put_schema_version_metadata)

Puts the metadata key value pair for a specified schema version ID. A maximum of 10 key value pairs will be allowed per schema version. They can be added over one or more calls.

Request

- `SchemaId` – A [SchemaId](#) (p. 901) object.
The unique ID for the schema.
- `SchemaVersionNumber` – A [SchemaVersionNumber](#) (p. 900) object.
The version number of the schema.
- `SchemaVersionId` – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12](#) (p. 980).
The unique version ID of the schema version.
- `MetadataKeyValue` – *Required*: A [MetadataKeyValuePair](#) (p. 900) object.
The metadata key's corresponding value.

Response

- `SchemaArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17](#) (p. 980).
The Amazon Resource Name (ARN) for the schema.
- `SchemaName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13](#) (p. 980).
The name for the schema.
- `RegistryName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13](#) (p. 980).
The name for the registry.
- `LatestVersion` – Boolean.
The latest version of the schema.
- `VersionNumber` – Number (long), not less than 1 or more than 100000.
The version number of the schema.
- `SchemaVersionId` – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12](#) (p. 980).
The unique version ID of the schema version.
- `MetadataKey` – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #24](#) (p. 980).

The metadata key.

- **MetadataValue** – UTF-8 string, not less than 1 or more than 256 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

The value of the metadata key.

Errors

- [InvalidInputException](#)
- [AccessDeniedException](#)
- [AlreadyExistsException](#)
- [EntityNotFoundException](#)
- [ResourceNumberLimitExceededException](#)

QuerySchemaVersionMetadata Action (Python: query_schema_version_metadata)

Queries for the schema version metadata information.

Request

- **SchemaId** – A [SchemaId \(p. 901\)](#) object.

A wrapper structure that may contain the schema name and Amazon Resource Name (ARN).

- **SchemaVersionNumber** – A [SchemaVersionNumber \(p. 900\)](#) object.

The version number of the schema.

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The unique version ID of the schema version.

- **MetadataList** – An array of [MetadataKeyValuePair \(p. 900\)](#) objects.

Search key-value pairs for metadata, if they are not provided all the metadata information will be fetched.

- **MaxResults** – Number (integer), not less than 1 or more than 50.

Maximum number of results required per page. If the value is not supplied, this will be defaulted to 25 per page.

- **NextToken** – UTF-8 string.

A continuation token, if this is a continuation call.

Response

- **MetadataInfoMap** – A map array of key-value pairs.

Each key is a UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

Each value is a A [MetadataInfo \(p. 898\)](#) object.

A map of a metadata key and associated values.

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The unique version ID of the schema version.

- **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](#)
- [AccessDeniedException](#)
- [EntityNotFoundException](#)

RemoveSchemaVersionMetadata Action (Python: remove_schema_version_metadata)

Removes a key value pair from the schema version metadata for the specified schema version ID.

Request

- **SchemaId** – A [SchemaId \(p. 901\)](#) object.

A wrapper structure that may contain the schema name and Amazon Resource Name (ARN).

- **SchemaVersionNumber** – A [SchemaVersionNumber \(p. 900\)](#) object.

The version number of the schema.

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The unique version ID of the schema version.

- **MetadataKeyValue** – *Required:* A [MetadataKeyValuePair \(p. 900\)](#) object.

The value of the metadata key.

Response

- **SchemaArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema.

- **SchemaName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema.

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry.

- **LatestVersion** – Boolean.

The latest version of the schema.

- **VersionNumber** – Number (long), not less than 1 or more than 100000.

The version number of the schema.

- **SchemaVersionId** – UTF-8 string, not less than 36 or more than 36 bytes long, matching the [Custom string pattern #12 \(p. 980\)](#).

The version ID for the schema version.

- **MetadataKey** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

The metadata key.

- **MetadataValue** – UTF-8 string, not less than 1 or more than 256 bytes long, matching the [Custom string pattern #24 \(p. 980\)](#).

The value of the metadata key.

Errors

- `InvalidArgumentException`
- `AccessDeniedException`
- `EntityNotFoundException`

DeleteRegistry Action (Python: delete_registry)

Delete the entire registry including schema and all of its versions. To get the status of the delete operation, you can call the `GetRegistry` API after the asynchronous call. Deleting a registry will deactivate all online operations for the registry such as the `UpdateRegistry`, `CreateSchema`, `UpdateSchema`, and `RegisterSchemaVersion` APIs.

Request

- **RegistryId** – *Required:* A [RegistryId \(p. 897\)](#) object.

This is a wrapper structure that may contain the registry name and Amazon Resource Name (ARN).

Response

- **RegistryName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the registry being deleted.

- **RegistryArn** – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the registry being deleted.

- **Status** – UTF-8 string (valid values: `AVAILABLE` | `DELETING`).

The status of the registry. A successful operation will return the `Deleting` status.

Errors

- `InvalidArgumentException`

- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

DeleteSchema Action (Python: `delete_schema`)

Deletes the entire schema set, including the schema set and all of its versions. To get the status of the delete operation, you can call `GetSchema` API after the asynchronous call. Deleting a registry will deactivate all online operations for the schema, such as the `GetSchemaByDefinition`, and `RegisterSchemaVersion` APIs.

Request

- `SchemaId` – *Required:* A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure that may contain the schema name and Amazon Resource Name (ARN).

Response

- `SchemaArn` – UTF-8 string, not less than 1 or more than 10240 bytes long, matching the [Custom string pattern #17 \(p. 980\)](#).

The Amazon Resource Name (ARN) of the schema being deleted.

- `SchemaName` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Custom string pattern #13 \(p. 980\)](#).

The name of the schema being deleted.

- `Status` – UTF-8 string (valid values: `AVAILABLE` | `PENDING` | `DELETING`).

The status of the schema.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

DeleteSchemaVersions Action (Python: `delete_schema_versions`)

Remove versions from the specified schema. A version number or range may be supplied. If the compatibility mode forbids deleting of a version that is necessary, such as `BACKWARDS_FULL`, an error is returned. Calling the `GetSchemaVersions` API after this call will list the status of the deleted versions.

When the range of version numbers contain check pointed version, the API will return a 409 conflict and will not proceed with the deletion. You have to remove the checkpoint first using the `DeleteSchemaCheckpoint` API before using this API.

You cannot use the `DeleteSchemaVersions` API to delete the first schema version in the schema set. The first schema version can only be deleted by the `DeleteSchema` API. This operation will also delete

the attached SchemaVersionMetadata under the schema versions. Hard deletes will be enforced on the database.

If the compatibility mode forbids deleting of a version that is necessary, such as BACKWARDS_FULL, an error is returned.

Request

- **SchemaId** – *Required:* A [SchemaId \(p. 901\)](#) object.

This is a wrapper structure that may contain the schema name and Amazon Resource Name (ARN).

- **Versions** – *Required:* UTF-8 string, not less than 1 or more than 100000 bytes long, matching the [Custom string pattern #25 \(p. 980\)](#).

A version range may be supplied which may be of the format:

- a single version number, 5
- a range, 5-8 : deletes versions 5, 6, 7, 8

Response

- **SchemaVersionErrors** – An array of [SchemaVersionErrorItem \(p. 900\)](#) objects.

A list of SchemaVersionErrorItem objects, each containing an error and schema version.

Errors

- [InvalidArgumentException](#)
- [EntityNotFoundException](#)
- [AccessDeniedException](#)
- [ConcurrentModificationException](#)

Workflows

The Workflows API describes the data types and API related to creating, updating, or viewing workflows in AWS Glue.

Data Types

- [JobNodeDetails Structure \(p. 922\)](#)
- [CrawlerNodeDetails Structure \(p. 922\)](#)
- [TriggerNodeDetails Structure \(p. 922\)](#)
- [Crawl Structure \(p. 922\)](#)
- [Node Structure \(p. 923\)](#)
- [Edge Structure \(p. 923\)](#)
- [Workflow Structure \(p. 924\)](#)
- [WorkflowGraph Structure \(p. 924\)](#)
- [WorkflowRun Structure \(p. 925\)](#)
- [WorkflowRunStatistics Structure \(p. 926\)](#)
- [StartingEventBatchCondition Structure \(p. 926\)](#)

- [Blueprint Structure \(p. 926\)](#)
- [BlueprintDetails Structure \(p. 927\)](#)
- [LastActiveDefinition Structure \(p. 927\)](#)
- [BlueprintRun Structure \(p. 928\)](#)

JobNodeDetails Structure

The details of a Job node present in the workflow.

Fields

- **JobRuns** – An array of [JobRun \(p. 856\)](#) 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 \(p. 922\)](#) 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 \(p. 866\)](#) 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 | CANCELLING | CANCELLED | SUCCEEDED | 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The log stream associated with the crawl.

Node Structure

A node represents an AWS Glue component (trigger, crawler, or job) on a workflow graph.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

The unique Id assigned to the node within the workflow.

- **TriggerDetails** – A [TriggerNodeDetails \(p. 922\)](#) object.

Details of the Trigger when the node represents a Trigger.

- **JobDetails** – A [JobNodeDetails \(p. 922\)](#) object.

Details of the Job when the node represents a Job.

- **CrawlerDetails** – A [CrawlerNodeDetails \(p. 922\)](#) object.

Details of the crawler when the node represents a crawler.

Edge Structure

An edge represents a directed connection between two AWS Glue components that 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 \(p. 980\)](#).

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 \(p. 980\)](#).

The unique of the node within the workflow where the edge ends.

Workflow Structure

A workflow is a collection of multiple dependent AWS Glue jobs and crawlers that are run to complete a complex ETL task. A workflow manages the execution and monitoring of all its jobs and crawlers.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the workflow.

- **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 \(p. 980\)](#).

Each value is a UTF-8 string.

A collection of properties to be used as part of each execution of the workflow. The run properties are made available to each job in the workflow. A job can modify the properties for the next jobs in the flow.

- **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 \(p. 925\)](#) object.

The information about the last execution of the workflow.

- **Graph** – A [WorkflowGraph \(p. 924\)](#) 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.

- **MaxConcurrentRuns** – Number (integer).

You can use this parameter to prevent unwanted multiple updates to data, to control costs, or in some cases, to prevent exceeding the maximum number of concurrent runs of any of the component jobs. If you leave this parameter blank, there is no limit to the number of concurrent workflow runs.

- **BlueprintDetails** – A [BlueprintDetails \(p. 927\)](#) object.

This structure indicates the details of the blueprint that this particular workflow is created from.

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 \(p. 923\)](#) objects.

A list of the the AWS Glue components belong to the workflow represented as nodes.

- **Edges** – An array of [Edge \(p. 923\)](#) 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 \(p. 980\)](#).

Name of the workflow that was run.

- **WorkflowRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of this workflow run.

- **PreviousRunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID of the previous 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 \(p. 980\)](#).

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 | STOPPING | STOPPED | ERROR).

The status of the workflow run.

- **ErrorMessage** – UTF-8 string.

This error message describes any error that may have occurred in starting the workflow run. Currently the only error message is "Concurrent runs exceeded for workflow: foo."

- **Statistics** – A [WorkflowRunStatistics \(p. 926\)](#) object.

The statistics of the run.

- **Graph** – A [WorkflowGraph \(p. 924\)](#) object.

The graph representing all the AWS Glue components that belong to the workflow as nodes and directed connections between them as edges.

- **StartingEventBatchCondition** – A [StartingEventBatchCondition \(p. 926\)](#) object.

The batch condition that started the workflow run.

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 that timed out.
- **FailedActions** – Number (integer).
Total number of Actions that have failed.
- **StoppedActions** – Number (integer).
Total number of Actions that have stopped.
- **SucceededActions** – Number (integer).
Total number of Actions that have succeeded.
- **RunningActions** – Number (integer).
Total number Actions in running state.

StartingEventBatchCondition Structure

The batch condition that started the workflow run. Either the number of events in the batch size arrived, in which case the `BatchSize` member is non-zero, or the batch window expired, in which case the `BatchWindow` member is non-zero.

Fields

- **BatchSize** – Number (integer).
Number of events in the batch.
- **BatchWindow** – Number (integer).
Duration of the batch window in seconds.

Blueprint Structure

The details of a blueprint.

Fields

- **Name** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The name of the blueprint.

- **Description** – UTF-8 string, not less than 1 or more than 512 bytes long.
The description of the blueprint.
- **CreatedOn** – Timestamp.
The date and time the blueprint was registered.
- **LastModifiedOn** – Timestamp.
The date and time the blueprint was last modified.
- **ParameterSpec** – UTF-8 string, not less than 1 or more than 131072 bytes long.
A JSON string that indicates the list of parameter specifications for the blueprint.
- **BlueprintLocation** – UTF-8 string.
Specifies the path in Amazon S3 where the blueprint is published.
- **BlueprintServiceLocation** – UTF-8 string.
Specifies a path in Amazon S3 where the blueprint is copied when you call `CreateBlueprint`/`UpdateBlueprint` to register the blueprint in AWS Glue.
- **Status** – UTF-8 string (valid values: `CREATING` | `ACTIVE` | `UPDATING` | `FAILED`).
The status of the blueprint registration.
 - `Creating` — The blueprint registration is in progress.
 - `Active` — The blueprint has been successfully registered.
 - `Updating` — An update to the blueprint registration is in progress.
 - `Failed` — The blueprint registration failed.
- **ErrorMessage** – UTF-8 string.
An error message.
- **LastActiveDefinition** – A [LastActiveDefinition \(p. 927\)](#) object.

When there are multiple versions of a blueprint and the latest version has some errors, this attribute indicates the last successful blueprint definition that is available with the service.

BlueprintDetails Structure

The details of a blueprint.

Fields

- **BlueprintName** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The name of the blueprint.
- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The run ID for this blueprint.

LastActiveDefinition Structure

When there are multiple versions of a blueprint and the latest version has some errors, this attribute indicates the last successful blueprint definition that is available with the service.

Fields

- **Description** – UTF-8 string, not less than 1 or more than 512 bytes long.
The description of the blueprint.
- **LastModifiedOn** – Timestamp.
The date and time the blueprint was last modified.
- **ParameterSpec** – UTF-8 string, not less than 1 or more than 131072 bytes long.
A JSON string specifying the parameters for the blueprint.
- **BlueprintLocation** – UTF-8 string.
Specifies a path in Amazon S3 where the blueprint is published by the AWS Glue developer.
- **BlueprintServiceLocation** – UTF-8 string.
Specifies a path in Amazon S3 where the blueprint is copied when you create or update the blueprint.

BlueprintRun Structure

The details of a blueprint run.

Fields

- **BlueprintName** – UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The name of the blueprint.
- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The run ID for this blueprint run.
- **WorkflowName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of a workflow that is created as a result of a successful blueprint run. If a blueprint run has an error, there will not be a workflow created.
- **State** – UTF-8 string (valid values: RUNNING | SUCCEEDED | FAILED | ROLLING_BACK).
The state of the blueprint run. Possible values are:
 - Running — The blueprint run is in progress.
 - Succeeded — The blueprint run completed successfully.
 - Failed — The blueprint run failed and rollback is complete.
 - Rolling Back — The blueprint run failed and rollback is in progress.
- **StartedOn** – Timestamp.
The date and time that the blueprint run started.
- **CompletedOn** – Timestamp.
The date and time that the blueprint run completed.
- **ErrorMessage** – UTF-8 string.
Indicates any errors that are seen while running the blueprint.

- **RollbackErrorMessage** – UTF-8 string.

If there are any errors while creating the entities of a workflow, we try to roll back the created entities until that point and delete them. This attribute indicates the errors seen while trying to delete the entities that are created.

- **Parameters** – UTF-8 string, not less than 1 or more than 131072 bytes long.

The blueprint parameters as a string. You will have to provide a value for each key that is required from the parameter spec that is defined in the [Blueprint\\$ParameterSpec](#).

- **RoleArn** – UTF-8 string, not less than 1 or more than 1024 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).

The role ARN. This role will be assumed by the AWS Glue service and will be used to create the workflow and other entities of a workflow.

Operations

- [CreateWorkflow Action \(Python: create_workflow\) \(p. 929\)](#)
- [UpdateWorkflow Action \(Python: update_workflow\) \(p. 930\)](#)
- [DeleteWorkflow Action \(Python: delete_workflow\) \(p. 931\)](#)
- [GetWorkflow Action \(Python: get_workflow\) \(p. 932\)](#)
- [ListWorkflows Action \(Python: list_workflows\) \(p. 932\)](#)
- [BatchGetWorkflows Action \(Python: batch_get_workflows\) \(p. 933\)](#)
- [GetWorkflowRun Action \(Python: get_workflow_run\) \(p. 933\)](#)
- [GetWorkflowRuns Action \(Python: get_workflow_runs\) \(p. 934\)](#)
- [GetWorkflowRunProperties Action \(Python: get_workflow_run_properties\) \(p. 935\)](#)
- [PutWorkflowRunProperties Action \(Python: put_workflow_run_properties\) \(p. 935\)](#)
- [CreateBlueprint Action \(Python: create_blueprint\) \(p. 936\)](#)
- [UpdateBlueprint Action \(Python: update_blueprint\) \(p. 937\)](#)
- [DeleteBlueprint Action \(Python: delete_blueprint\) \(p. 937\)](#)
- [ListBlueprints Action \(Python: list_blueprints\) \(p. 938\)](#)
- [BatchGetBlueprints Action \(Python: batch_get_blueprints\) \(p. 938\)](#)
- [StartBlueprintRun Action \(Python: start_blueprint_run\) \(p. 939\)](#)
- [GetBlueprintRun Action \(Python: get_blueprint_run\) \(p. 940\)](#)
- [GetBlueprintRuns Action \(Python: get_blueprint_runs\) \(p. 940\)](#)
- [StartWorkflowRun Action \(Python: start_workflow_run\) \(p. 941\)](#)
- [StopWorkflowRun Action \(Python: stop_workflow_run\) \(p. 942\)](#)
- [ResumeWorkflowRun Action \(Python: resume_workflow_run\) \(p. 942\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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.

- **MaxConcurrentRuns** – Number (integer).

You can use this parameter to prevent unwanted multiple updates to data, to control costs, or in some cases, to prevent exceeding the maximum number of concurrent runs of any of the component jobs. If you leave this parameter blank, there is no limit to the number of concurrent workflow runs.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

Each value is a UTF-8 string.

A collection of properties to be used as part of each execution of the workflow.

- **MaxConcurrentRuns** – Number (integer).

You can use this parameter to prevent unwanted multiple updates to data, to control costs, or in some cases, to prevent exceeding the maximum number of concurrent runs of any of the component jobs. If you leave this parameter blank, there is no limit to the number of concurrent workflow runs.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

Name of the workflow specified in input.

Errors

- `InvalidArgumentException`
- `InternalServiceException`

- `OperationTimeoutException`
- `ConcurrentModificationException`

GetWorkflow Action (Python: get_workflow)

Retrieves resource metadata for a workflow.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the workflow to retrieve.

- `IncludeGraph` – Boolean.

Specifies whether to include a graph when returning the workflow resource metadata.

Response

- `Workflow` – A [Workflow \(p. 924\)](#) object.

The resource metadata for the workflow.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

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

- `InvalidArgumentException`
- `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 \(p. 924\)](#) 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`
- `InvalidArgumentException`

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 \(p. 980\)](#).
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 \(p. 980\)](#).
The ID of the workflow run.
- **IncludeGraph** – Boolean.

Specifies whether to include the workflow graph in response or not.

Response

- Run – A [WorkflowRun \(p. 925\)](#) 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 \(p. 980\)](#).

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 \(p. 925\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).

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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).

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`

CreateBlueprint Action (Python: create_blueprint)

Registers a blueprint with AWS Glue.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The name of the blueprint.

- **Description** – UTF-8 string, not less than 1 or more than 512 bytes long.

A description of the blueprint.

- **BlueprintLocation** – *Required*: UTF-8 string, not less than 1 or more than 8192 bytes long, matching the [Custom string pattern #22 \(p. 980\)](#).

Specifies a path in Amazon S3 where the blueprint is published.

- **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 applied to this blueprint.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Returns the name of the blueprint that was registered.

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`

UpdateBlueprint Action (Python: update_blueprint)

Updates a registered blueprint.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The name of the blueprint.
- **Description** – UTF-8 string, not less than 1 or more than 512 bytes long.
A description of the blueprint.
- **BlueprintLocation** – *Required*: UTF-8 string, not less than 1 or more than 8192 bytes long, matching the [Custom string pattern #22 \(p. 980\)](#).
Specifies a path in Amazon S3 where the blueprint is published.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Returns the name of the blueprint that was updated.

Errors

- `EntityNotFoundException`
- `ConcurrentModificationException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `IllegalBlueprintStateException`

DeleteBlueprint Action (Python: delete_blueprint)

Deletes an existing blueprint.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the blueprint to delete.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Returns the name of the blueprint that was deleted.

Errors

- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`

ListBlueprints Action (Python: list_blueprints)

Lists all the blueprint names in an 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.
- **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.
Filters the list by an AWS resource tag.

Response

- **Blueprints** – An array of UTF-8 strings.
List of names of blueprints in the account.
- **NextToken** – UTF-8 string.
A continuation token, if not all blueprint names have been returned.

Errors

- `InvalidArgumentException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetBlueprints Action (Python: batch_get_blueprints)

Retrieves information about a list of blueprints.

Request

- **Names** – *Required*: An array of UTF-8 strings, not less than 1 or more than 25 strings.
A list of blueprint names.
- **IncludeBlueprint** – Boolean.
Specifies whether or not to include the blueprint in the response.
- **IncludeParameterSpec** – Boolean.
Specifies whether or not to include the parameters, as a JSON string, for the blueprint in the response.

Response

- **Blueprints** – An array of [Blueprint \(p. 926\)](#) objects.
Returns a list of blueprint as a `Blueprints` object.
- **MissingBlueprints** – An array of UTF-8 strings.
Returns a list of `BlueprintNames` that were not found.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

StartBlueprintRun Action (Python: start_blueprint_run)

Starts a new run of the specified blueprint.

Request

- **BlueprintName** – *Required*: UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).
The name of the blueprint.
- **Parameters** – UTF-8 string, not less than 1 or more than 131072 bytes long.
Specifies the parameters as a `BlueprintParameters` object.
- **RoleArn** – *Required*: UTF-8 string, not less than 1 or more than 1024 bytes long, matching the [Custom string pattern #20 \(p. 980\)](#).
Specifies the IAM role used to create the workflow.

Response

- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The run ID for this blueprint run.

Errors

- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`
- `EntityNotFoundException`
- `IllegalBlueprintStateException`

GetBlueprintRun Action (Python: `get_blueprint_run`)

Retrieves the details of a blueprint run.

Request

- `BlueprintName` – *Required*: UTF-8 string, not less than 1 or more than 128 bytes long, matching the [Custom string pattern #21 \(p. 980\)](#).

The name of the blueprint.

- `RunId` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The run ID for the blueprint run you want to retrieve.

Response

- `BlueprintRun` – A [BlueprintRun \(p. 928\)](#) object.

Returns a `BlueprintRun` object.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetBlueprintRuns Action (Python: `get_blueprint_runs`)

Retrieves the details of blueprint runs for a specified blueprint.

Request

- `BlueprintName` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the blueprint.

- `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

- **BlueprintRuns** – An array of [BlueprintRun \(p. 928\)](#) objects.
Returns a list of BlueprintRun objects.
- **NextToken** – UTF-8 string.
A continuation token, if not all blueprint runs have been returned.

Errors

- [EntityNotFoundException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)
- [InvalidArgumentException](#)

StartWorkflowRun Action (Python: start_workflow_run)

Starts a new run of the specified workflow.

Request

- **Name** – *Required:* UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of the workflow to start.
- **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 \(p. 980\)](#).
Each value is a UTF-8 string.
The workflow run properties for the new workflow run.

Response

- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
An Id for the new run.

Errors

- [InvalidArgumentException](#)
- [EntityNotFoundException](#)
- [InternalServiceException](#)
- [OperationTimeoutException](#)

- ResourceNumberLimitExceeded**Exception**
- ConcurrentRunsExceeded**Exception**

StopWorkflowRun Action (Python: stop_workflow_run)

Stops the execution of the specified workflow run.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of the workflow to stop.
- **RunId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the workflow run to stop.

Response

- *No Response parameters.*

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `IllegalWorkflowStateException`

ResumeWorkflowRun Action (Python: resume_workflow_run)

Restarts selected nodes of a previous partially completed workflow run and resumes the workflow run. The selected nodes and all nodes that are downstream from the selected nodes are run.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The name of the workflow to resume.
- **RunId** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
The ID of the workflow run to resume.
- **NodeIds** – *Required*: An array of UTF-8 strings.
A list of the node IDs for the nodes you want to restart. The nodes that are to be restarted must have a run attempt in the original run.

Response

- **RunId** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The new ID assigned to the resumed workflow run. Each resume of a workflow run will have a new run ID.

- **NodeIds** – An array of UTF-8 strings.

A list of the node IDs for the nodes that were actually restarted.

Errors

- `InvalidArgumentException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentRunsExceededException`
- `IllegalWorkflowStateException`

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 \(p. 944\)](#)
- [EvaluationMetrics Structure \(p. 944\)](#)
- [MLTransform Structure \(p. 944\)](#)
- [FindMatchesParameters Structure \(p. 946\)](#)
- [FindMatchesMetrics Structure \(p. 947\)](#)
- [ConfusionMatrix Structure \(p. 948\)](#)
- [GlueTable Structure \(p. 948\)](#)
- [TaskRun Structure \(p. 949\)](#)
- [TransformFilterCriteria Structure \(p. 949\)](#)
- [TransformSortCriteria Structure \(p. 950\)](#)
- [TaskRunFilterCriteria Structure \(p. 950\)](#)
- [TaskRunSortCriteria Structure \(p. 951\)](#)
- [TaskRunProperties Structure \(p. 951\)](#)
- [FindMatchesTaskRunProperties Structure \(p. 951\)](#)
- [ImportLabelsTaskRunProperties Structure \(p. 952\)](#)
- [ExportLabelsTaskRunProperties Structure \(p. 952\)](#)
- [LabelingSetGenerationTaskRunProperties Structure \(p. 952\)](#)
- [SchemaColumn Structure \(p. 952\)](#)
- [TransformEncryption Structure \(p. 953\)](#)
- [MLUserDataEncryption Structure \(p. 953\)](#)
- [ColumnImportance Structure \(p. 953\)](#)

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 \(p. 946\)](#) 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 \(p. 947\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 948\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- `Parameters` – A [TransformParameters \(p. 944\)](#) 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 \(p. 944\)](#) 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 \(p. 952\)](#) 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 #15 \(p. 980\)](#).

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.

- **TransformEncryption** – A [TransformEncryption \(p. 953\)](#) object.

The encryption-at-rest settings of the transform that apply to accessing user data. Machine learning transforms can access user data encrypted in Amazon S3 using KMS.

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 \(p. 980\)](#).

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 \(p. 948\)](#) 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.

- `ColumnImportances` – An array of [ColumnImportance \(p. 953\)](#) objects, not more than 100 structures.

A list of `ColumnImportance` structures containing column importance metrics, sorted in order of descending importance.

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 951\)](#) object.

Specifies configuration properties associated with this task run.

- **ErrorResponse** – 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 \(p. 980\)](#).

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 #15 \(p. 980\)](#).
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 \(p. 952\)](#) 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 \(p. 952\)](#) object.

The configuration properties for an importing labels task run.

- **ExportLabelsTaskRunProperties** – An [ExportLabelsTaskRunProperties \(p. 952\)](#) object.

The configuration properties for an exporting labels task run.

- **LabelingSetGenerationTaskRunProperties** – A [LabelingSetGenerationTaskRunProperties \(p. 952\)](#) object.

The configuration properties for a labeling set generation task run.

- **FindMatchesTaskRunProperties** – A [FindMatchesTaskRunProperties \(p. 951\)](#) 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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).
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 \(p. 980\)](#).

The name of the column.

- DataType – UTF-8 string, not more than 131072 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The type of data in the column.

TransformEncryption Structure

The encryption-at-rest settings of the transform that apply to accessing user data. Machine learning transforms can access user data encrypted in Amazon S3 using KMS.

Additionally, imported labels and trained transforms can now be encrypted using a customer provided KMS key.

Fields

- MlUserDataEncryption – A [MLUserDataEncryption \(p. 953\)](#) object.

An MLUserDataEncryption object containing the encryption mode and customer-provided KMS key ID.

- TaskRunSecurityConfigurationName – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the security configuration.

MLUserDataEncryption Structure

The encryption-at-rest settings of the transform that apply to accessing user data.

Fields

- MlUserDataEncryptionMode – *Required*: UTF-8 string (valid values: DISABLED | SSE-KMS="SSEKMS").

The encryption mode applied to user data. Valid values are:

- DISABLED: encryption is disabled
- SSEKMS: use of server-side encryption with AWS Key Management Service (SSE-KMS) for user data stored in Amazon S3.
- KmsKeyId – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The ID for the customer-provided KMS key.

ColumnImportance Structure

A structure containing the column name and column importance score for a column.

Column importance helps you understand how columns contribute to your model, by identifying which columns in your records are more important than others.

Fields

- **ColumnName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of a column.

- **Importance** – Number (double), not more than 1.0.

The column importance score for the column, as a decimal.

Operations

- [CreateMLTransform Action \(Python: create_ml_transform\) \(p. 954\)](#)
- [UpdateMLTransform Action \(Python: update_ml_transform\) \(p. 957\)](#)
- [DeleteMLTransform Action \(Python: delete_ml_transform\) \(p. 958\)](#)
- [GetMLTransform Action \(Python: get_ml_transform\) \(p. 959\)](#)
- [GetMLTransforms Action \(Python: get_ml_transforms\) \(p. 961\)](#)
- [ListMLTransforms Action \(Python: list_ml_transforms\) \(p. 961\)](#)
- [StartMLEvaluationTaskRun Action \(Python: start_ml_evaluation_task_run\) \(p. 962\)](#)
- [StartMLLabelingSetGenerationTaskRun Action \(Python: start_ml_labeling_set_generation_task_run\) \(p. 963\)](#)
- [GetMLTaskRun Action \(Python: get_ml_task_run\) \(p. 964\)](#)
- [GetMLTaskRuns Action \(Python: get_ml_task_runs\) \(p. 965\)](#)
- [CancelMLTaskRun Action \(Python: cancel_ml_task_run\) \(p. 966\)](#)
- [StartExportLabelsTaskRun Action \(Python: start_export_labels_task_run\) \(p. 966\)](#)
- [StartImportLabelsTaskRun Action \(Python: start_import_labels_task_run\) \(p. 967\)](#)

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 \(p. 980\)](#).

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 \(p. 980\)](#).

A description of the machine learning transform that is being defined. The default is an empty string.

- **InputRecordTables** – *Required*: An array of [GlueTable \(p. 948\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- **Parameters** – *Required*: A [TransformParameters \(p. 944\)](#) 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 #15 \(p. 980\)](#).

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.

- `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 machine learning transform. You may use tags to limit access to the machine learning transform. For more information about tags in AWS Glue, see [AWS Tags in AWS Glue](#) in the developer guide.

- `TransformEncryption` – A [TransformEncryption \(p. 953\)](#) object.

The encryption-at-rest settings of the transform that apply to accessing user data. Machine learning transforms can access user data encrypted in Amazon S3 using KMS.

Response

- `TransformId` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

A description of the transform. The default is an empty string.

- **Parameters** – A [TransformParameters \(p. 944\)](#) 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 #15 \(p. 980\)](#).

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 \(p. 980\)](#).

The unique identifier for the transform that was updated.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

The unique identifier of the transform that was deleted.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 948\)](#) objects, not more than 10 structures.

A list of AWS Glue table definitions used by the transform.

- `Parameters` – A [TransformParameters \(p. 944\)](#) object.

The configuration parameters that are specific to the algorithm used.

- `EvaluationMetrics` – An [EvaluationMetrics \(p. 944\)](#) object.

The latest evaluation metrics.

- **LabelCount** – Number (integer).

The number of labels available for this transform.

- **Schema** – An array of [SchemaColumn \(p. 952\)](#) 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 #15 \(p. 980\)](#).

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.

- **TransformEncryption** – A [TransformEncryption \(p. 953\)](#) object.

The encryption-at-rest settings of the transform that apply to accessing user data. Machine learning transforms can access user data encrypted in Amazon S3 using KMS.

Errors

- [EntityNotFoundException](#)
- [InvalidInputException](#)

- `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 \(p. 949\)](#) object.
The filter transformation criteria.
- `Sort` – A [TransformSortCriteria \(p. 950\)](#) object.
The sorting criteria.

Response

- `Transforms` – *Required:* An array of [MLTransform \(p. 944\)](#) objects.
A list of machine learning transforms.
- `NextToken` – UTF-8 string.
A pagination token, if more results are available.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListMLTransforms Action (Python: list_ml_transforms)

Retrieves a sortable, filterable list of existing AWS Glue machine learning transforms in this AWS account, or the resources with the specified tag. This operation takes the optional `Tags` field, which you can use as a filter of the responses so that tagged resources can be retrieved as a group. If you choose to use tag filtering, only resources with the tags 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.

- **Filter** – A [TransformFilterCriteria \(p. 949\)](#) object.

A `TransformFilterCriteria` used to filter the machine learning transforms.

- **Sort** – A [TransformSortCriteria \(p. 950\)](#) object.

A `TransformSortCriteria` used to sort the machine learning transforms.

- **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

- **TransformIds** – *Required:* An array of UTF-8 strings.

The identifiers of all the machine learning transforms in the account, or the machine learning transforms 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`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

The unique identifier associated with this run.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 951\)](#) 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`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 950\)](#) object.

The filter criteria, in the `TaskRunFilterCriteria` structure, for the task run.

- `Sort` – A [TaskRunSortCriteria \(p. 951\)](#) object.

The sorting criteria, in the `TaskRunSortCriteria` structure, for the task run.

Response

- `TaskRuns` – An array of [TaskRun \(p. 949\)](#) 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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).

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`
- `InvalidArgumentException`
- `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 \(p. 980\)](#).

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 \(p. 980\)](#).

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 \(p. 980\)](#).
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 \(p. 980\)](#).

The unique identifier for the task run.

Errors

- `EntityNotFoundException`
- `InvalidArgumentException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`

Sensitive Data Detection API

The Sensitive Data Detection API describes the APIs used to detect sensitive data across the columns and rows of your structured data.

Data Types

- [CustomEntityType Structure \(p. 968\)](#)

CustomEntityType Structure

An object representing a custom pattern for detecting sensitive data across the columns and rows of your structured data.

Fields

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).
A name for the custom pattern that allows it to be retrieved or deleted later. This name must be unique per AWS account.
- **RegexString** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

A regular expression string that is used for detecting sensitive data in a custom pattern.

- **ContextWords** – An array of UTF-8 strings, not less than 1 or more than 20 strings.

A list of context words. If none of these context words are found within the vicinity of the regular expression the data will not be detected as sensitive data.

If no context words are passed only a regular expression is checked.

Operations

- [CreateCustomEntityType Action \(Python: create_custom_entity_type\) \(p. 969\)](#)
- [DeleteCustomEntityType Action \(Python: delete_custom_entity_type\) \(p. 970\)](#)
- [GetCustomEntityType Action \(Python: get_custom_entity_type\) \(p. 970\)](#)
- [BatchGetCustomEntityTypes Action \(Python: batch_get_custom_entity_types\) \(p. 971\)](#)
- [ListCustomEntityTypes Action \(Python: list_custom_entity_types\) \(p. 972\)](#)

CreateCustomEntityType Action (Python: create_custom_entity_type)

Creates a custom pattern that is used to detect sensitive data across the columns and rows of your structured data.

Each custom pattern you create specifies a regular expression and an optional list of context words. If no context words are passed only a regular expression is checked.

Request

- **Name** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

A name for the custom pattern that allows it to be retrieved or deleted later. This name must be unique per AWS account.

- **RegexString** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

A regular expression string that is used for detecting sensitive data in a custom pattern.

- **ContextWords** – An array of UTF-8 strings, not less than 1 or more than 20 strings.

A list of context words. If none of these context words are found within the vicinity of the regular expression the data will not be detected as sensitive data.

If no context words are passed only a regular expression is checked.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the custom pattern you created.

Errors

- `AccessDeniedException`
- `AlreadyExistsException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCustomEntityType Action (Python: `delete_custom_entity_type`)

Deletes a custom pattern by specifying its name.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the custom pattern that you want to delete.

Response

- `Name` – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the custom pattern you deleted.

Errors

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

GetCustomEntityType Action (Python: `get_custom_entity_type`)

Retrieves the details of a custom pattern by specifying its name.

Request

- `Name` – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the custom pattern that you want to retrieve.

Response

- **Name** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the custom pattern that you retrieved.

- **RegexString** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

A regular expression string that is used for detecting sensitive data in a custom pattern.

- **ContextWords** – An array of UTF-8 strings, not less than 1 or more than 20 strings.

A list of context words if specified when you created the custom pattern. If none of these context words are found within the vicinity of the regular expression the data will not be detected as sensitive data.

Errors

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

BatchGetCustomEntityTypes Action (Python: batch_get_custom_entity_types)

Retrieves the details for the custom patterns specified by a list of names.

Request

- **Names** – *Required:* An array of UTF-8 strings, not less than 1 or more than 50 strings.

A list of names of the custom patterns that you want to retrieve.

Response

- **CustomEntityTypes** – An array of [CustomEntityType \(p. 968\)](#) objects.

A list of `CustomEntityType` objects representing the custom patterns that have been created.

- **CustomEntityTypesNotFound** – An array of UTF-8 strings, not less than 1 or more than 50 strings.

A list of the names of custom patterns that were not found.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

ListCustomEntityTypes Action (Python: list_custom_entity_types)

Lists all the custom patterns that have been created.

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.

Response

- **CustomEntityTypes** – An array of [CustomEntityType \(p. 968\)](#) objects.
A list of `CustomEntityType` objects representing custom patterns.
- **NextToken** – UTF-8 string.
A pagination token, if more results are available.

Errors

- `InvalidArgumentException`
- `OperationTimeoutException`
- `InternalServiceException`

Tagging APIs in AWS Glue

Data Types

- [Tag Structure \(p. 972\)](#)

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 \(Python: tag_resource\) \(p. 973\)](#)
- [UntagResource Action \(Python: untag_resource\) \(p. 973\)](#)
- [GetTags Action \(Python: get_tags\) \(p. 974\)](#)

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 [Custom string pattern #17 \(p. 980\)](#).

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 [Custom string pattern #17 \(p. 980\)](#).

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 [Custom string pattern #17 \(p. 980\)](#).

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** – *Required:* Blob.

The unscaled numeric value.

- **Scale** – *Required:* 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 \(p. 980\)](#).

The code associated with this error.

- **ErrorMessage** – Description string, not more than 2048 bytes long, matching the [URI address multi-line string pattern \(p. 980\)](#).

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 \(p. 980\)](#).

The URI for accessing the resource.

ColumnStatistics Structure

Represents the generated column-level statistics for a table or partition.

Fields

- **ColumnName** – *Required*: UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

Name of column which statistics belong to.

- **ColumnType** – *Required*: Type name, not more than 20000 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The data type of the column.

- **AnalyzedTime** – *Required*: Timestamp.

The timestamp of when column statistics were generated.

- **StatisticsData** – *Required*: A [ColumnStatisticsData \(p. 977\)](#) object.

A ColumnStatisticData object that contains the statistics data values.

ColumnStatisticsError Structure

Encapsulates a ColumnStatistics object that failed and the reason for failure.

Fields

- **ColumnStatistics** – A [ColumnStatistics \(p. 976\)](#) object.

The ColumnStatistics of the column.

- **Error** – An [ErrorDetail \(p. 975\)](#) object.

An error message with the reason for the failure of an operation.

ColumnError Structure

Encapsulates a column name that failed and the reason for failure.

Fields

- **ColumnName** – UTF-8 string, not less than 1 or more than 255 bytes long, matching the [Single-line string pattern \(p. 980\)](#).

The name of the column that failed.

- **Error** – An [ErrorDetail \(p. 975\)](#) object.

An error message with the reason for the failure of an operation.

ColumnStatisticsData Structure

Contains the individual types of column statistics data. Only one data object should be set and indicated by the `Type` attribute.

Fields

- **Type** – *Required:* UTF-8 string (valid values: `BOOLEAN` | `DATE` | `DECIMAL` | `DOUBLE` | `LONG` | `STRING` | `BINARY`).

The type of column statistics data.

- **BooleanColumnStatisticsData** – A [BooleanColumnStatisticsData \(p. 977\)](#) object.

Boolean column statistics data.

- **DateColumnStatisticsData** – A [DateColumnStatisticsData \(p. 978\)](#) object.

Date column statistics data.

- **DecimalColumnStatisticsData** – A [DecimalColumnStatisticsData \(p. 978\)](#) object.

Decimal column statistics data.

- **DoubleColumnStatisticsData** – A [DoubleColumnStatisticsData \(p. 978\)](#) object.

Double column statistics data.

- **LongColumnStatisticsData** – A [LongColumnStatisticsData \(p. 979\)](#) object.

Long column statistics data.

- **StringColumnStatisticsData** – A [StringColumnStatisticsData \(p. 979\)](#) object.

String column statistics data.

- **BinaryColumnStatisticsData** – A [BinaryColumnStatisticsData \(p. 979\)](#) object.

Binary column statistics data.

BooleanColumnStatisticsData Structure

Defines column statistics supported for Boolean data columns.

Fields

- **NumberOfTrues** – *Required:* Number (long), not more than None.

- **NumberOfFalses** – *Required:* Number (long), not more than None.
The number of false values in the column.
- **NumberOfNulls** – *Required:* Number (long), not more than None.
The number of null values in the column.

DateColumnStatisticsData Structure

Defines column statistics supported for timestamp data columns.

Fields

- **MinimumValue** – Timestamp.
The lowest value in the column.
- **MaximumValue** – Timestamp.
The highest value in the column.
- **NumberOfNulls** – *Required:* Number (long), not more than None.
The number of null values in the column.
- **NumberOfDistinctValues** – *Required:* Number (long), not more than None.
The number of distinct values in a column.

DecimalColumnStatisticsData Structure

Defines column statistics supported for fixed-point number data columns.

Fields

- **MinimumValue** – A [DecimalNumber \(p. 975\)](#) object.
The lowest value in the column.
- **MaximumValue** – A [DecimalNumber \(p. 975\)](#) object.
The highest value in the column.
- **NumberOfNulls** – *Required:* Number (long), not more than None.
The number of null values in the column.
- **NumberOfDistinctValues** – *Required:* Number (long), not more than None.
The number of distinct values in a column.

DoubleColumnStatisticsData Structure

Defines column statistics supported for floating-point number data columns.

Fields

- **MinimumValue** – Number (double).

The lowest value in the column.

- **MinimumValue** – Number (double).

The highest value in the column.

- **MaximumValue** – Number (double).

The number of null values in the column.

- **NumberOfNulls** – *Required*: Number (long), not more than None.

The number of distinct values in a column.

LongColumnStatisticsData Structure

Defines column statistics supported for integer data columns.

Fields

- **MinimumValue** – Number (long).

The lowest value in the column.

- **MaximumValue** – Number (long).

The highest value in the column.

- **NumberOfNulls** – *Required*: Number (long), not more than None.

The number of null values in the column.

- **NumberOfDistinctValues** – *Required*: Number (long), not more than None.

The number of distinct values in a column.

StringColumnStatisticsData Structure

Defines column statistics supported for character sequence data values.

Fields

- **MaximumLength** – *Required*: Number (long), not more than None.

The size of the longest string in the column.

- **AverageLength** – *Required*: Number (double), not more than None.

The average string length in the column.

- **NumberOfNulls** – *Required*: Number (long), not more than None.

The number of null values in the column.

- **NumberOfDistinctValues** – *Required*: Number (long), not more than None.

The number of distinct values in a column.

BinaryColumnStatisticsData Structure

Defines column statistics supported for bit sequence data values.

Fields

- **MaximumLength** – *Required*: Number (long), not more than None.
The size of the longest bit sequence in the column.
- **AverageLength** – *Required*: Number (double), not more than None.
The average bit sequence length in the column.
- **NumberOfNulls** – *Required*: Number (long), not more than None.
The number of null values in the column.

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 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 – "[\p{L}\p{N}\p{P}]*"
- Custom string pattern #12 – "[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"
- Custom string pattern #13 – "[a-zA-Z0-9-_\$.]+"
- Custom string pattern #14 – "^\w+\.\w+\.\w+\$"
- Custom string pattern #15 – "^\w+\.\w+\$"
- Custom string pattern #16 – "^(2-3|3[.])9)\$"
- Custom string pattern #17 – "arn:(aws|aws-us-gov|aws-cn):glue:.*"
- Custom string pattern #18 – "(^arn:aws:iam::\w{12}:root)"
- Custom string pattern #19 – "arn:aws:kms:.*"
- Custom string pattern #20 – "arn:aws[^:]*:iam::[0-9]*:role/.+"
- Custom string pattern #21 – "[\.\-_A-Za-z0-9]+"
- Custom string pattern #22 – "^\w+//([^\w]+)/([^\w]+)*([^\w]+)\$"
- Custom string pattern #23 – ".*\\$\w.*"
- Custom string pattern #24 – "[a-zA-Z0-9+-=._./@]+"
- Custom string pattern #25 – "[1-9][0-9]*|[1-9][0-9]*-[1-9][0-9]*"
- Custom string pattern #26 – "[\s\S]*"
- Custom string pattern #27 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[\^\S\r\n'= ;])*"

- Custom string pattern #28 – "[*A-Za-z0-9_-]*"
- Custom string pattern #29 – "[A-Za-z0-9_-]*"
- Custom string pattern #30 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF] | [^\S\r\n])*"
- Custom string pattern #31 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF] | [^\S\r\n])*"
- Custom string pattern #32 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF] | [\s])*"
- Custom string pattern #33 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF] | [^\r\n])*"

Exceptions

This section describes AWS Glue exceptions that you can use to find the source of problems and fix them. For more information on HTTP error codes and strings for exceptions related to machine learning, see the section called “[AWS Glue Machine Learning Exceptions](#)” (p. 996).

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.

The 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.

IllegalWorkflowStateException Structure

The workflow is in an invalid state to perform a requested operation.

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.

InvalidStateException Structure

An error that indicates your data is in an invalid state.

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 \(p. 980\)](#).
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.

ResourceNotReadyException Structure

A resource was not ready for a transaction.

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 \(p. 988\)](#)
- [Troubleshooting Connection Issues in AWS Glue \(p. 988\)](#)
- [Troubleshooting Errors in AWS Glue \(p. 989\)](#)
- [AWS Glue Machine Learning Exceptions \(p. 996\)](#)
- [AWS Glue Quotas \(p. 1003\)](#)

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 \(p. 989\)](#)
- [Error: Could Not Find S3 Endpoint or NAT Gateway for subnetId in VPC \(p. 990\)](#)
- [Error: Inbound Rule in Security Group Required \(p. 990\)](#)
- [Error: Outbound Rule in Security Group Required \(p. 990\)](#)
- [Error: Job Run Failed Because the Role Passed Should Be Given Assume Role Permissions for the AWS Glue Service \(p. 990\)](#)
- [Error: DescribeVpcEndpoints Action Is Unauthorized. Unable to Validate VPC ID vpc-id \(p. 990\)](#)
- [Error: DescribeRouteTables Action Is Unauthorized. Unable to Validate Subnet Id: subnet-id in VPC id: vpc-id \(p. 991\)](#)
- [Error: Failed to Call ec2:DescribeSubnets \(p. 991\)](#)
- [Error: Failed to Call ec2:DescribeSecurityGroups \(p. 991\)](#)
- [Error: Could Not Find Subnet for AZ \(p. 991\)](#)
- [Error: Job Run Exception When Writing to a JDBC Target \(p. 991\)](#)
- [Error: Amazon S3 Timeout \(p. 992\)](#)
- [Error: Amazon S3 Access Denied \(p. 992\)](#)
- [Error: Amazon S3 Access Key ID Does Not Exist \(p. 992\)](#)
- [Error: Job Run Fails When Accessing Amazon S3 with an s3a:// URI \(p. 992\)](#)
- [Error: Amazon S3 Service Token Expired \(p. 993\)](#)
- [Error: No Private DNS for Network Interface Found \(p. 994\)](#)
- [Error: Development Endpoint Provisioning Failed \(p. 994\)](#)
- [Error: Notebook Server CREATE_FAILED \(p. 994\)](#)
- [Error: Local Notebook Fails to Start \(p. 994\)](#)
- [Error: Notebook Usage Errors \(p. 994\)](#)
- [Error: Running Crawler Failed \(p. 995\)](#)
- [Error: Partitions Were Not Updated \(p. 995\)](#)
- [Error: Upgrading Athena Data Catalog \(p. 995\)](#)
- [Error: A Job is Reprocessing Data When Job Bookmarks Are Enabled \(p. 995\)](#)

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 \(p. 32\)](#).

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 \(p. 32\)](#).
- 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 \(p. 32\)](#).

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 \(p. 32\)](#).
- 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 \(p. 32\)](#).

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 \(p. 17\)](#).

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 \(p. 16\)](#).

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 \(p. 513\)](#).

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 the instance profile name `iamInstanceProfile.name` that is not valid. 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: Partitions Were Not Updated

In case your partitions were not updated in the Data Catalog when you ran an ETL job, these log statements from the `DataSink` class in the CloudWatch logs may be helpful:

- "Attempting to fast-forward updates to the Catalog - `nameSpace`:" — Shows which database, table, and catalogId are attempted to be modified by this job. If this statement is not here, check if `enableUpdateCatalog` is set to true and properly passed as a `getSink()` parameter or in `additional_options`.
- "Schema change policy behavior:" — Shows which schema `updateBehavior` value you passed in.
- "Schemas qualify (schema compare):" — Will be true or false.
- "Schemas qualify (case-insensitive compare):" — Will be true or false.
- If both are false and your `updateBehavior` is not set to `UPDATE_IN_DATABASE`, then your `DynamicFrame` schema needs to be identical or contain a subset of the columns seen in the Data Catalog table schema.

For more information on updating partitions, see [Updating the Schema and Partitions in the Data Catalog](#).

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 \(p. 197\)](#). 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 Machine Learning Exceptions

This topic describes HTTP error codes and strings for AWS Glue exceptions related to machine learning. The error codes and error strings are provided for each machine learning activity that may occur when you perform an operation. Also, you can see whether it is possible to retry the operation that resulted in the error.

CreateMLTaskRunActivity

This activity has the following exceptions:

- `EntityNotFoundException` (400)
 - "Cannot find MLTransform in account [accountId] with handle [transformName]."
 - "No ML Task Run found for [taskRunId]: in account [accountId] for transform [transformName]."

OK to retry: No.

CancelMLTaskRunActivity

This activity has the following exceptions:

- `InvalidInputException` (400)

- “Internal service failure due to unexpected input.”
- “An AWS Glue Table input source should be specified in transform.”
- “Input source column [columnName] has an invalid data type defined in the catalog.”
- “Exactly one input record table must be provided.”
- “Should specify database name.”
- “Should specify table name.”
- “Schema is not defined on the transform.”
- “Schema should contain given primary key: [primaryKey].”
- “Problem fetching the data catalog schema: [message].”
- “Cannot set Max Capacity and Worker Num/Type at the same time.”
- “Both WorkerType and NumberOfWorkers should be set.”
- “MaxCapacity should be >= [maxCapacity].”
- “NumberOfWorkers should be >= [maxCapacity].”
- “Max retries should be non-negative.”
- “Find Matches parameters have not been set.”
- “A primary key must be specified in Find Matches parameters.”

OK to retry: No.

- AlreadyExistsException (400)
 - “Transform with name [transformName] already exists.”

OK to retry: No.

- IdempotentParameterMismatchException (400)
 - “Idempotent create request for transform [transformName] had mismatching parameters.”

OK to retry: No.

- InternalServiceException (500)
 - “Dependency failure.”

OK to retry: Yes.

- ResourceNumberLimitExceededException (400)
 - “ML Transforms count ([count]) has exceeded the limit of [limit] transforms.”

OK to retry: Yes, once you’ve deleted a transform to make room for this new one.

DeleteMLTransformActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName]”
- OK to retry: No.

GetMLTaskRunActivity

This activity has the following exceptions:

- EntityNotFoundException (400)

- “Cannot find MLTransform in account [accountId] with handle [transformName].”
- “No ML Task Run found for [taskRunId]: in account [accountId] for transform [transformName].”

OK to retry: No.

GetMLTaskRunsActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
- “Cannot find MLTransform in account [accountId] with handle [transformName].”
- “No ML Task Run found for [taskRunId]: in account [accountId] for transform [transformName].”

OK to retry: No.

GetMLTransformActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

GetMLTransformsActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- InvalidInputException (400)
 - “Account ID can't be blank.”
 - “Sorting not supported for column [column].”
 - “[column] can't be blank.”
 - “Internal service failure due to unexpected input.”

OK to retry: No.

GetSaveLocationForTransformArtifactActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- InvalidInputException (400)

- “Unsupported artifact type [artifactType].”
- “Internal service failure due to unexpected input.”

OK to retry: No.

GetTaskRunArtifactActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”
 - “No ML Task Run found for [taskRunId]: in account [accountId] for transform [transformName].”

OK to retry: No.

- InvalidInputException (400)
 - “File name ‘[fileName]’ is invalid for publish.”
 - “Cannot retrieve artifact for [taskType] task type.”
 - “Cannot retrieve artifact for [artifactType].”
 - “Internal service failure due to unexpected input.”

OK to retry: No.

PublishMLTransformModelActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”
 - “An existing model with version - [version] cannot be found for account id - [accountId] - and transform id - [transformId].”

OK to retry: No.

- InvalidInputException (400)
 - “File name ‘[fileName]’ is invalid for publish.”
 - “Illegal leading minus sign on unsigned string [string].”
 - “Bad digit at end of [string].”
 - “String value [string] exceeds range of unsigned long.”
 - “Internal service failure due to unexpected input.”

OK to retry: No.

PullLatestMLTransformModelActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- InvalidInputException (400)

- “Internal service failure due to unexpected input.”
OK to retry: No.
 - ConcurrentModificationException (400)
 - “Cannot create model version to train due to racing inserts with mismatching parameters.”
 - “The ML Transform model for transform id [transformId] is stale or being updated by another process; Please retry.”
- OK to retry: Yes.

PutJobMetadataForMLTransformActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”
 - “No ML Task Run found for [taskRunId]: in account [accountId] for transform [transformName].”
 - OK to retry: No.
 - InvalidInputException (400)
 - “Internal service failure due to unexpected input.”
 - “Unknown job metadata type [jobType].”
 - “Must provide a task run ID to update.”
- OK to retry: No.

StartExportLabelsTaskRunActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”
 - “No labelset exists for transformId [transformId] in account id [accountId].”
 - OK to retry: No.
 - InvalidInputException (400)
 - “[message].”
 - “S3 path provided is not in the same region as transform. Expecting region - [region], but got - [region].”
- OK to retry: No.

StartImportLabelsTaskRunActivity

This activity has the following exceptions:

- EntityNotFoundException (400)
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”
- OK to retry: No.

- **InvalidInputException (400)**
 - “[message].”
 - “Invalid label file path.”
 - “Cannot access the label file at [labelPath]. [message].”
 - “Cannot use IAM role provided in the transform. Role: [role].”
 - “Invalid label file of size 0.”
 - “S3 path provided is not in the same region as transform. Expecting region - [region], but got - [region].”

OK to retry: No.

- **ResourceNumberLimitExceededException (400)**
 - “Label file has exceeded the limit of [limit] MB.”

OK to retry: No. Consider breaking your label file into several smaller files.

StartMLEvaluationTaskRunActivity

This activity has the following exceptions:

- **EntityNotFoundException (400)**
 - “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- **InvalidInputException (400)**
 - “Exactly one input record table must be provided.”
 - “Should specify database name.”
 - “Should specify table name.”
 - “Find Matches parameters have not been set.”
 - “A primary key must be specified in Find Matches parameters.”

OK to retry: No.

- **MLTransformNotReadyException (400)**
 - “This operation can only be applied to a transform that is in a READY state.”

OK to retry: No.

- **InternalServiceException (500)**
 - “Dependency failure.”

OK to retry: Yes.

- **ConcurrentRunsExceededException (400)**
 - “ML Task Runs count [count] has exceeded the transform limit of [limit] task runs.”
 - “ML Task Runs count [count] has exceeded the limit of [limit] task runs.”

OK to retry: Yes, after waiting for task runs to finish.

StartMLLabelingSetGenerationTaskRunActivity

This activity has the following exceptions:

- **EntityNotFoundException (400)**

- “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- **InvalidInputException (400)**

- “Exactly one input record table must be provided.”

- “Should specify database name.”

- “Should specify table name.”

- “Find Matches parameters have not been set.”

- “A primary key must be specified in Find Matches parameters.”

OK to retry: No.

- **InternalServiceException (500)**

- “Dependency failure.”

OK to retry: Yes.

- **ConcurrentRunsExceededException (400)**

- “ML Task Runs count [count] has exceeded the transform limit of [limit] task runs.”

OK to retry: Yes, after task runs have completed.

UpdateMLTransformActivity

This activity has the following exceptions:

- **EntityNotFoundException (400)**

- “Cannot find MLTransform in account [accountId] with handle [transformName].”

OK to retry: No.

- **InvalidInputException (400)**

- “Another transform with name [transformName] already exists.”

- “[message].”

- “Transform name cannot be blank.”

- “Cannot set Max Capacity and Worker Num/Type at the same time.”

- “Both WorkerType and NumberOfWorkers should be set.”

- “MaxCapacity should be >= [minMaxCapacity].”

- “NumberOfWorkers should be >= [minNumWorkers].”

- “Max retries should be non-negative.”

- “Internal service failure due to unexpected input.”

- “Find Matches parameters have not been set.”

- “A primary key must be specified in Find Matches parameters.”

OK to retry: No.

- **AlreadyExistsException (400)**

- “Transform with name [transformName] already exists.”

OK to retry: No.

- **IdempotentParameterMismatchException (400)**

- “Idempotent create request for transform [transformName] had mismatching parameters.”

OK to retry: No.

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.

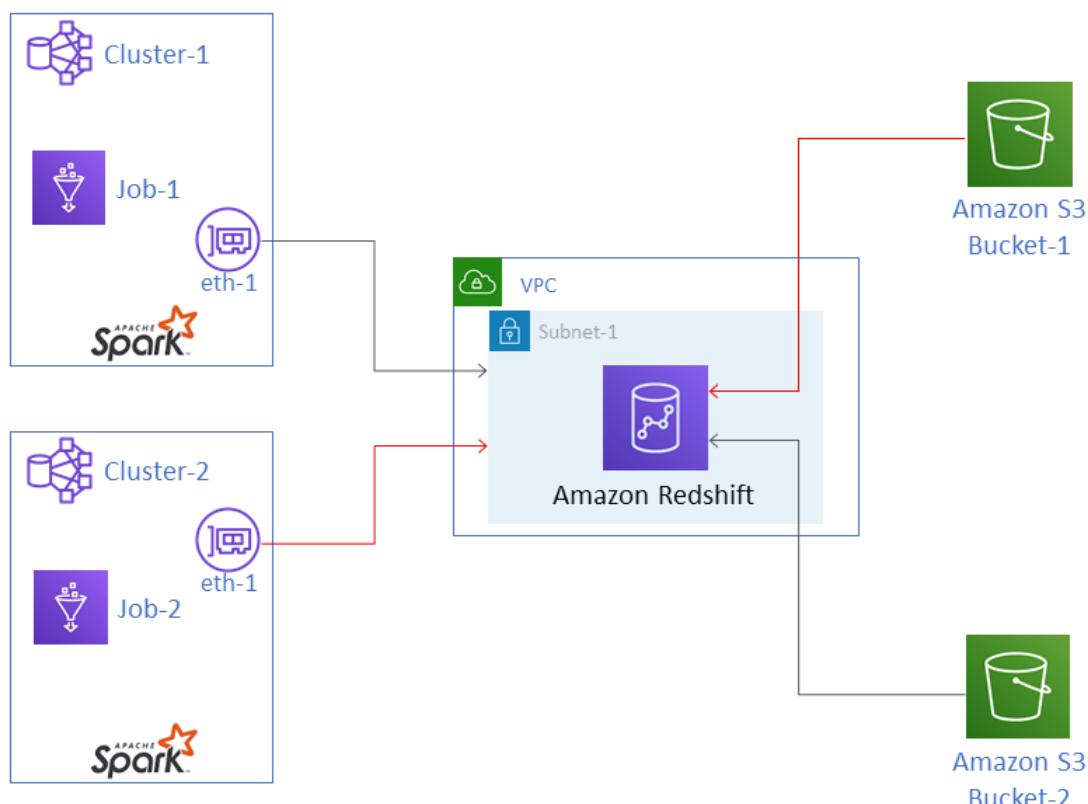
Topics

- Preventing Cross-Job Data Access (p. 1004)

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" \(p. 51\)](#).

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 AWS Glue version parameter is configured when adding or updating a job. AWS 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, the corresponding Spark and Python versions, and other changes in functionality.

AWS Glue Versions

AWS Glue version	Supported Spark and Python versions	Changes in Functionality
AWS Glue 0.9	<ul style="list-style-type: none"> Spark 2.2.1 Python 2.7 	<p>Jobs that were created without specifying a AWS Glue version default to AWS Glue 0.9.</p>
AWS Glue 1.0	<ul style="list-style-type: none"> Spark 2.4.3 Python 2.7 Python 3.6 	<p>You can maintain job bookmarks for Parquet and ORC formats in AWS Glue ETL jobs (using AWS 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.</p> <p>When setting format options for ETL inputs and outputs, you can specify to use Apache Avro reader/writer format 1.8 to support Avro logical type reading and writing (using AWS Glue version 1.0). Previously, only the version 1.7 Avro reader/writer format was supported.</p> <p>The DynamoDB connection type supports a writer option (using AWS Glue Version 1.0).</p>
AWS Glue 2.0	<ul style="list-style-type: none"> Spark 2.4.3 Python 3.7 	<p>In addition to the features provided in AWS Glue version 1.0, AWS Glue Version 2.0 also provides:</p> <ul style="list-style-type: none"> An upgraded infrastructure for running Apache Spark ETL jobs in AWS Glue with reduced startup times. Default logging is now realtime, with separate streams for drivers and

AWS Glue version	Supported Spark and Python versions	Changes in Functionality
		<p>executors, and outputs and errors.</p> <ul style="list-style-type: none">• Support for specifying additional Python modules or different versions at the job level. <p>Note AWS Glue version 2.0 differs from AWS Glue Version 1.0 for some dependencies and versions due to underlying architectural changes. Please validate your Glue jobs before migrating across major AWS Glue version releases.</p> <p>For more information about AWS Glue Version 2.0 features and limitations, see Running Spark ETL Jobs with Reduced Startup Times (p. 1009).</p>

AWS Glue version	Supported Spark and Python versions	Changes in Functionality
AWS Glue 3.0	<ul style="list-style-type: none"> • Spark 3.1.1 • Python 3.7 	<p>AWS Glue 3.0 is the new version of AWS Glue. In addition to the Spark engine upgrade to 3.0, there are optimizations and upgrades built into this AWS Glue release, such as:</p> <ul style="list-style-type: none"> • Builds the AWS Glue ETL Library against Spark 3.0, which is a major release for Spark. • Streaming jobs are supported on AWS Glue 3.0. • Includes new AWS Glue Spark runtime optimizations for performance and reliability: <ul style="list-style-type: none"> • Faster in-memory columnar processing based on Apache Arrow for reading CSV data. • SIMD based execution for vectorized reads with CSV data. • Spark upgrade also includes additional optimizations developed on Amazon EMR. • Upgraded EMRFS from 2.38 to 2.46 enabling new features and bug fixes for Amazon S3 access. • Upgraded several dependencies that were required for the new Spark version. See Appendix A: notable dependency upgrades (p. 1017). • Upgraded JDBC drivers for our natively supported data sources. See Appendix B: JDBC driver upgrades (p. 1017). <p>Limitations</p> <p>The following are limitations with AWS Glue 3.0:</p> <ul style="list-style-type: none"> • AWS Glue machine learning transforms are not yet available in AWS Glue 3.0. • Some custom Spark connectors do not work with AWS Glue 3.0 if they depend

AWS Glue version	Supported Spark and Python versions	Changes in Functionality
		<p>on Spark 2.4 and do not have compatibility with Spark 3.1.</p> <p>For more information about migrating to AWS Glue version 3.0, see Migrating AWS Glue jobs to AWS Glue version 3.0 (p. 1012).</p>

Running Spark ETL Jobs with Reduced Startup Times

AWS Glue versions 2.0 and later provide an upgraded infrastructure for running Apache Spark ETL (extract, transform, and load) jobs in AWS Glue with reduced startup times. With the reduced wait times, data engineers can be more productive and increase their interactivity with AWS Glue. The reduced variance in job start times can help you meet or exceed your SLAs of making data available for analytics.

To use this feature with your AWS Glue ETL jobs, choose **2.0** or a later version for the `Glue version` when creating your jobs.

Topics

- [New Features Supported \(p. 1009\)](#)
- [Logging Behavior \(p. 1011\)](#)
- [Features Not Supported \(p. 1012\)](#)

New Features Supported

This section describes new features supported with AWS Glue versions 2.0 and later.

Support for Specifying Additional Python Modules at the Job Level

AWS Glue versions 2.0 and later also let you provide additional Python modules or different versions at the job level. You can use the `--additional-python-modules` option with a list of comma-separated Python modules to add a new module or change the version of an existing module.

For example to update or to add a new `scikit-learn` module use the following key/value: `--additional-python-modules", "scikit-learn==0.21.3"`.

Also, within the `--additional-python-modules` option you can specify an Amazon S3 path to a Python wheel module. For example:

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

AWS Glue uses the Python Package Installer (`pip3`) to install the additional modules. You can pass additional options specified by the `python-modules-installer-option` to `pip3` for installing the modules. Any incompatibility or limitations from `pip3` will apply.

Python Modules Already Provided in AWS Glue Version 2.0

AWS Glue version 2.0 supports the following python modules out of the box:

- `setuptools`—45.2.0
- `subprocess32`—3.5.4
- `ptvsd`—4.3.2
- `pydevd`—1.9.0
- `PyMySQL`—0.9.3
- `docutils`—0.15.2
- `jmespath`—0.9.4
- `six`—1.14.0
- `python_dateutil`—2.8.1
- `urllib3`—1.25.8
- `botocore`—1.15.4
- `s3transfer`—0.3.3
- `boto3`—1.12.4
- `certifi`—2019.11.28
- `chardet`—3.0.4
- `idna`—2.9
- `requests`—2.23.0
- `pyparsing`—2.4.6
- `enum34`—1.1.9
- `pytz`—2019.3
- `numpy`—1.18.1
- `cycler`—0.10.0
- `kiwisolver`—1.1.0
- `scipy`—1.4.1
- `pandas`—1.0.1
- `pyarrow`—0.16.0
- `matplotlib`—3.1.3
- `pyhocon`—0.3.54
- `mpmath`—1.1.0
- `sympy`—1.5.1
- `patsy`—0.5.1
- `statsmodels`—0.11.1
- `fsspec`—0.6.2
- `s3fs`—0.4.0
- `Cython`—0.29.15
- `joblib`—0.14.1
- `pmdarima`—1.5.3
- `scikit-learn`—0.22.1
- `tbats`—1.0.9

Logging Behavior

AWS Glue versions 2.0 and later support different default logging behavior. The differences include:

- Logging occurs in realtime.
- There are separate streams for drivers and executors.
- For each driver and executor there are two streams, the output stream and the error stream.

Driver and Executor Streams

Driver streams are identified by the job run ID. Executor streams are identified by the job <*run id*>_<*executor task id*>. For example:

- "logStreamName":
"jr_8255308b426ffff1b4e09e00e0bd5612b1b4ec848d7884cebe61ed33a31789..._g-f65f617bd31d54bd94482af755b6cdf464542..."

Output and Errors Streams

The output stream has the standard output (stdout) from your code. The error stream has logging messages from your code/library.

- Log streams:
 - Driver log streams have <*jr*>, where <*jr*> is the job run ID.
 - Executor log streams have <*jr*>_<*g*>, where <*g*> is the task ID for the executor. You can look up the executor task ID in the driver error log.

The default log groups for AWS Glue version 2.0 are as follows:

- /aws-glue/jobs/logs/output for output
- /aws-glue/jobs/logs/error for errors

When a security configuration is provided, the log group names change to:

- /aws-glue/jobs/<*security configuration*>-role/<*Role Name*>/output
- /aws-glue/jobs/<*security configuration*>-role/<*Role Name*>/error

On the console the **Logs** link points to the output log group and the **Error** link points to the error log group. When continuous logging is enabled, the **Logs** links points to the continuous log group, and the **Output** link points to the output log group.

Logging Rules

Note

In AWS Glue versions 2.0 and later, an issue with continuous logging requires that you create the log group before using the continuous logging feature. To work around this issue, you must create the default logging groups or customer log groups specified by --continuous-log-logGroup, otherwise the continuous log streams will not show up. AWS is working to address this issue.

The default log groupname for continuous logging is /aws-glue/jobs/logs-v2.

In AWS Glue versions 2.0 and later, continuous logging has the same behavior as in AWS Glue version 1.0:

- Default log group: /aws-glue/jobs/logs-v2
- Driver log stream: <*jr*>-driver
- Executor log stream: <*jr*>-<*executor ID*>

The log group name can be changed by setting --continuous-log-logGroupName

The log streams name can be prefixed by setting --continuous-log-logStreamPrefix

Features Not Supported

The following AWS Glue features are not supported:

- Development endpoints
- AWS Glue versions 2.0 and later do not run on Apache YARN, so YARN settings do not apply
- AWS Glue versions 2.0 and later do not have a Hadoop Distributed File System (HDFS)
- AWS Glue versions 2.0 and later do not use dynamic allocation, hence the ExecutorAllocationManager metrics are not available
- For AWS Glue version 2.0 or later jobs, you specify the number of workers and worker type, but do not specify a maxCapacity.
- AWS Glue versions 2.0 and later do not support s3n out of the box. We recommend using s3 or s3a. If jobs need to use s3n for any reason, you can pass the following additional argument:

```
--conf spark.hadoop.fs.s3n.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem
```

Migrating AWS Glue jobs to AWS Glue version 3.0

This topic describes the changes between AWS Glue versions 0.9, 1.0, 2.0 and 3.0 to allow you to migrate your Spark applications and ETL jobs to AWS Glue 3.0.

To use this feature with your AWS Glue ETL jobs, choose **3.0** for the `Glue` version when creating your jobs.

Topics

- [New Features Supported \(p. 1012\)](#)
- [Actions to migrate to AWS Glue 3.0 \(p. 1013\)](#)
- [Migration check list \(p. 1013\)](#)
- [Migrating from AWS Glue 0.9 to AWS Glue 3.0 \(p. 1014\)](#)
- [Migrating from AWS Glue 1.0 to AWS Glue 3.0 \(p. 1015\)](#)
- [Migrating from AWS Glue 2.0 to AWS Glue 3.0 \(p. 1016\)](#)
- [Appendix A: notable dependency upgrades \(p. 1017\)](#)
- [Appendix B: JDBC driver upgrades \(p. 1017\)](#)

New Features Supported

This section describes new features and advantages of AWS Glue version 3.0.

- It is based on Apache Spark 3.1.1, which has optimizations from open-source Spark and developed by the AWS Glue and EMR services such as adaptive query execution, vectorized readers, and optimized shuffles and partition coalescing.

- Upgraded JDBC drivers for all Glue native sources including MySQL, Microsoft SQL Server, Oracle, PostgreSQL, MongoDB, and upgraded Spark libraries and dependencies brought in by Spark 3.1.1.
- Optimized Amazon S3 access with upgraded EMRFS and enabled Amazon S3 optimized output committers by default.
- Optimized Data Catalog access with partition indexes, push down predicates, partition listing, and upgraded Hive metastore client.
- Integration with Lake Formation for governed catalog tables with cell-level filtering and data lake transactions.
- Improved Spark UI experience with Spark 3.1.1 with new Spark executor memory metrics and Spark structured streaming metrics.
- Reduced startup latency improving overall job completion times and interactivity, similar to AWS Glue 2.0.
- Spark jobs are billed in 1-second increments with a 10x lower minimum billing duration—from a 10-minute minimum to a 1-minute minimum, similar to AWS Glue 2.0.

Actions to migrate to AWS Glue 3.0

For existing jobs, change the `Glue version` from the previous version to `Glue 3.0` in the job configuration.

- In the console, choose `Spark 3.1, Python 3 (Glue Version 3.0)` or `Spark 3.1, Scala 2 (Glue Version 3.0)` in `Glue version`.
- In AWS Glue Studio, choose `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3` in `Glue version`.
- In the API, choose `3.0` in the `GlueVersion` parameter in the [UpdateJob API](#).

For new jobs, choose `Glue 3.0` when you create a job.

- In the console, choose `Spark 3.1, Python 3 (Glue Version 3.0)` or `Spark 3.1, Scala 2 (Glue Version 3.0)` in `Glue version`.
- In AWS Glue Studio, choose `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3` in `Glue version`.
- In the API, choose `3.0` in the `GlueVersion` parameter in the [CreateJob API](#).

To view Spark event logs of AWS Glue 3.0, [launch an upgraded Spark history server for Glue 3.0 using CloudFormation or Docker](#).

Migration check list

Review this checklist for migration.

- Does your job depend on HDFS? If yes, try replacing HDFS with S3.
 - Search the file system path starting with `hdfs://` or `/` as DFS path in the job script code.
 - Check if your default file system is not configured with HDFS. If it is configured explicitly, you need to remove the `fs.defaultFS` configuration.
 - Check if your job contains any `dfs.*` parameters. If it contains any, you need to verify it is okay to disable the parameters.
- Does your job depend on YARN? If yes, verify the impacts by checking if your job contains the following parameters. If it contains any, you need to verify it is okay to disable the parameters.
 - `spark.yarn.*`

For example:

```
spark.yarn.executor.memoryOverhead  
spark.yarn.driver.memoryOverhead  
spark.yarn.scheduler.reporterThread.maxFailures
```

- `yarn.*`

For example:

```
yarn.scheduler.maximum-allocation-mb  
yarn.nodemanager.resource.memory-mb
```

- Does your job depend on Spark 2.2.1 or Spark 2.4.3? If yes, verify the impacts by checking if your job uses features changed in Spark 3.1.1.
 - <https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-22-to-23>

For example the `percentile_approx` function, or the `SparkSession` with `SparkSession.builder.getOrCreate()` when there is an existing `SparkContext`.

- <https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-23-to-24>

For example the `array_contains` function, or the `CURRENT_DATE`, `CURRENT_TIMESTAMP` function with `spark.sql.caseSensitive=true`.

- Do your job's extra jars conflict in Glue 3.0?
 - From AWS Glue 0.9/1.0: Extra jars supplied in existing AWS Glue 0.9/1.0 jobs may bring in classpath conflicts due to upgraded or new dependencies available in Glue 3.0. You can avoid classpath conflicts in AWS Glue 3.0 with the `--user-jars-first` AWS Glue job parameter or by shading your dependencies.
 - From AWS Glue 2.0: You can still avoid classpath conflicts in AWS Glue 3.0 with the `--user-jars-first` AWS Glue job parameter or by shading your dependencies.
- Do your jobs depend on Scala 2.11?
 - AWS Glue 3.0 uses Scala 2.12 so you need to rebuild your libraries with Scala 2.12 if your libraries depend on Scala 2.11.
- Do your job's external Python libraries depend on Python 2.7/3.6?
 - Use the `--additional-python-modules` parameters instead of setting the egg/wheel/zip file in the Python library path.
 - Update the dependent libraries from Python 2.7/3.6 to Python 3.7 as Spark 3.1.1 removed Python 2.7 support.

Migrating from AWS Glue 0.9 to AWS Glue 3.0

Note the following changes when migrating:

- AWS Glue 0.9 uses open-source Spark 2.2.1 and AWS Glue 3.0 uses EMR-optimized Spark 3.1.1.
 - Several Spark changes alone may require revision of your scripts to ensure removed features are not being referenced.
 - For example, Spark 3.1.1 does not enable Scala-untyped UDFs but Spark 2.2 does allow them.
- All jobs in AWS Glue 3.0 will be executed with significantly improved startup times. Spark jobs will be billed in 1-second increments with a 10x lower minimum billing duration since startup latency will go from 10 minutes maximum to 1 minute maximum.

- Logging behavior has changed since AWS Glue 2.0.
- Several dependency updates, highlighted in
- Scala is also updated to 2.12 from 2.11, and Scala 2.12 is not backwards compatible with Scala 2.11.
- Python 3.7 is also the default version used for Python scripts, as AWS Glue 0.9 was only utilizing Python 2.
 - Python 2.7 is not supported with Spark 3.1.1.
 - A new mechanism of installing additional Python modules is available.
- AWS Glue 3.0 does not run on Apache YARN, so YARN settings do not apply.
- AWS Glue 3.0 does not have a Hadoop Distributed File System (HDFS).
- Any extra jars supplied in existing AWS Glue 0.9 jobs may bring in conflicting dependencies since there were upgrades in several dependencies in 3.0 from 0.9. You can avoid classpath conflicts in AWS Glue 3.0 with the `--user-jars-first` AWS Glue job parameter.
- AWS Glue 3.0 does not yet support dynamic allocation, hence the `ExecutorAllocationManager` metrics are not available.
- In AWS Glue version 3.0 jobs, you specify the number of workers and worker type, but do not specify a `maxCapacity`.
- AWS Glue 3.0 does not yet support machine learning transforms.
- AWS Glue 3.0 does not yet support development endpoints.

Refer to the Spark migration documentation:

- see [Upgrading from Spark SQL 2.2 to 2.3](#)
- see [Upgrading from Spark SQL 2.3 to 2.4](#)
- see [Upgrading from Spark SQL 2.4 to 3.0](#)
- see [Upgrading from Spark SQL 3.0 to 3.1](#)
- see [Changes in Datetime behavior to be expected since Spark 3.0](#).

Migrating from AWS Glue 1.0 to AWS Glue 3.0

Note the following changes when migrating:

- AWS Glue 1.0 uses open-source Spark 2.4 and AWS Glue 3.0 uses EMR-optimized Spark 3.1.1.
 - Several Spark changes alone may require revision of your scripts to ensure removed features are not being referenced.
 - For example, Spark 3.1.1 does not enable Scala-untyped UDFs but Spark 2.4 does allow them.
- All jobs in AWS Glue 3.0 will be executed with significantly improved startup times. Spark jobs will be billed in 1-second increments with a 10x lower minimum billing duration since startup latency will go from 10 minutes maximum to 1 minute maximum.
- Logging behavior has changed since AWS Glue 2.0.
- Several dependency updates, highlighted in
- Scala is also updated to 2.12 from 2.11, and Scala 2.12 is not backwards compatible with Scala 2.11.
- Python 3.7 is also the default version used for Python scripts, as AWS Glue 0.9 was only utilizing Python 2.
 - Python 2.7 is not supported with Spark 3.1.1.
 - A new mechanism of installing additional Python modules is available.
- AWS Glue 3.0 does not run on Apache YARN, so YARN settings do not apply.
- AWS Glue 3.0 does not have a Hadoop Distributed File System (HDFS).

- Any extra jars supplied in existing AWS Glue 1.0 jobs may bring in conflicting dependencies since there were upgrades in several dependencies in 3.0 from 1.0. You can avoid classpath conflicts in AWS Glue 3.0 with the `--user-jars-first` AWS Glue job parameter.
- AWS Glue 3.0 does not yet support dynamic allocation, hence the `ExecutorAllocationManager` metrics are not available.
- In AWS Glue version 3.0 jobs, you specify the number of workers and worker type, but do not specify a `maxCapacity`.
- AWS Glue 3.0 does not yet support machine learning transforms.
- AWS Glue 3.0 does not yet support development endpoints.

Refer to the Spark migration documentation:

- see [Upgrading from Spark SQL 2.4 to 3.0](#)
- see [Changes in Datetime behavior to be expected since Spark 3.0](#).

Migrating from AWS Glue 2.0 to AWS Glue 3.0

Note the following changes when migrating:

- All existing job parameters and major features that exist in AWS Glue 2.0 will exist in AWS Glue 3.0.
 - The EMRFS S3-optimized committer for writing Parquet data into Amazon S3 is enabled by default in AWS Glue 3.0. However, you can still disable it by setting `--enable-s3-parquet-optimized-committer` to `false`.
 - AWS Glue 2.0 uses open-source Spark 2.4 and AWS Glue 3.0 uses EMR-optimized Spark 3.1.1.
 - Several Spark changes alone may require revision of your scripts to ensure removed features are not being referenced.
 - For example, Spark 3.1.1 does not enable Scala-untyped UDFs but Spark 2.4 does allow them.
 - AWS Glue 3.0 also features an update to EMRFS, updated JDBC drivers, and inclusions of additional optimizations onto Spark itself provided by AWS Glue.
 - All jobs in AWS Glue 3.0 will be executed with significantly improved startup times. Spark jobs will be billed in 1-second increments with a 10x lower minimum billing duration since startup latency will go from 10 minutes maximum to 1 minute maximum.
 - Python 2.7 is not supported with Spark 3.1.1.
 - Several dependency updates, highlighted in [Appendix A: notable dependency upgrades \(p. 1017\)](#).
 - Scala is also updated to 2.12 from 2.11, and Scala 2.12 is not backwards compatible with Scala 2.11.
 - Any extra jars supplied in existing AWS Glue 2.0 jobs may bring in conflicting dependencies since there were upgrades in several dependencies in 3.0 from 2.0. You can avoid classpath conflicts in AWS Glue 3.0 with the `--user-jars-first` AWS Glue job parameter.
 - AWS Glue 3.0 has different Spark task parallelism for driver/executor configuration compared to AWS Glue 2.0 and improves the performance and better utilizes the available resources. Both `spark.driver.cores` and `spark.executor.cores` are configured to number of cores on AWS Glue 3.0 (4 on the standard and G.1X worker, and 8 on the G.2X worker). These configurations do not change the worker type or hardware for the AWS Glue job. You can use these configurations to calculate the number of partitions or splits to match the Spark task parallelism in your Spark application.
 - AWS Glue 3.0 uses Spark 3.1, which changes the behavior to loading/saving of timestamps from/to parquet files. For more details, see [Upgrading from Spark SQL 3.0 to 3.1](#).

We recommend to set the following parameters when reading/writing parquet data that contains timestamp columns. Setting those parameters can resolve the calendar incompatibility issue that

occurs during the Spark 2 to Spark 3 upgrade, for both the AWS Glue Dynamic Frame and Spark Data Frame. Use the CORRECTED option to read the datetime value as it is; and the LEGACY option to rebase the datetime values with regard to the calendar difference during reading.

```
- Key: --conf
- Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --
  conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf
  spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
```

Refer to the Spark migration documentation:

- see [Upgrading from Spark SQL 2.4 to 3.0](#)
- see [Changes in Datetime behavior to be expected since Spark 3.0](#).

Appendix A: notable dependency upgrades

The following are dependency upgrades:

Dependency	Version in AWS Glue 0.9	Version in AWS Glue 1.0	Version in AWS Glue 2.0	Version in AWS Glue 3.0
Spark	2.2.1	2.4.3	2.4.3	3.1.1-amzn-0
Hadoop	2.7.3-amzn-6	2.8.5-amzn-1	2.8.5-amzn-5	3.2.1-amzn-3
Scala	2.11	2.11	2.11	2.12
Jackson	2.7.x	2.7.x	2.7.x	2.10.x
Hive	1.2	1.2	1.2	2.3.7-amzn-4
EMRFS	2.20.0	2.30.0	2.38.0	2.46.0
Json4s	3.2.x	3.5.x	3.5.x	3.6.6
Arrow	N/A	0.10.0	0.10.0	2.0.0
AWS Glue Catalog client	N/A	N/A	1.10.0	3.0.0

Appendix B: JDBC driver upgrades

The following are JDBC driver upgrades:

Driver	JDBC driver version in past AWS Glue versions	JDBC driver version in AWS Glue 3.0
MySQL	5.1	8.0.23
Microsoft SQL Server	6.1.0	7.0.0
Oracle Databaes	11.2	21.1
PostgreSQL	42.1.0	42.2.18

Driver	JDBC driver version in past AWS Glue versions	JDBC driver version in AWS Glue 3.0
MongoDB	2.0.0	4.0.0

AWS Glue version support policy

AWS Glue is a serverless data integration service that makes it easy to discover, prepare, and combine data for analytics, machine learning, and application development. An *AWS Glue job* contains the business logic that performs the data integration work in AWS Glue. There are two types of jobs in AWS Glue: *Spark (batch and streaming)* and *Python shell*. A Spark job is run in an Apache Spark environment managed by AWS Glue. A Python shell job runs Python scripts as a shell. When you define your job, you specify the AWS Glue version which configures the Spark and Python version in the runtime environment. For example: an AWS Glue version 2.0 job supports Spark 2.4.3 and Python 3.7.

Support Policy

Occasionally AWS Glue discontinues support for old AWS Glue versions. You can continue to create, update and run jobs on those versions. However, jobs running on deprecated versions are no longer eligible for technical support. AWS Glue will no longer apply security patches or other updates to deprecated versions. AWS Glue will also not honor SLAs when jobs are run on deprecated versions.

The following AWS Glue versions have reached or are scheduled for end of support. End of support starts at midnight (Pacific time zone) on the specified date.

Type	Glue Version	End of support
Spark	Spark 2.2, Scala 2 (Glue Version 0.9)	6/1/2022
Spark	Spark 2.2, Python 2 (Glue Version 0.9)	6/1/2022
Spark	Spark 2.4, Python 2 (Glue Version 1.0)	6/1/2022
Type	Python Version	End of support
Python shell	Python 2 (Glue Version 1.0)	6/1/2022

AWS strongly recommends that you migrate your jobs to supported versions.

For information on migrating your Spark jobs to the latest AWS Glue version, see [Migrating AWS Glue jobs to AWS Glue version 3.0](#).

For migrating your Python shell jobs to the latest AWS Glue version:

- In the console, choose Python 3 (Glue Version 1.0).
- In the [CreateJob/UpdateJob](#) API, set the `GlueVersion` parameter to `1.0`, and the `PythonVersion` to `3` under the `Command` parameter.
- You need to make your job script compatible with Python 3.

Document History for AWS Glue

update-history-change	update-history-description	update-history-date
Support for Auto Scaling for AWS Glue jobs (GA) (p. 1019)	Added information on using Auto Scaling for jobs in AWS Glue version 3.0 to dynamically scale compute resources. For more information, see Using Auto Scaling for AWS Glue .	April 14, 2022
Update to the documentation for AWS Glue developing and testing AWS Glue job scripts (p. 1019)	Reorganized and added information on the available development and testing methods for AWS Glue, including instructions for developing with Docker. For more information, see Developing and testing AWS Glue job scripts .	March 14, 2022
Addition of Protocol Buffers (Protobuf) as a supported data format for the AWS Glue Schema Registry (p. 1019)	Added information about Protobuf as a supported data format (in addition to AVRO and JSON). For more information, see AWS Glue Schema Registry .	February 25, 2022
Support for crawling Delta Lake tables (p. 1019)	Added information about using AWS Glue to crawl Delta Lake tables. For more information, see How to specify configuration options for a Delta Lake data store .	February 24, 2022
Support for AWS Glue job insights (p. 1019)	Added information about using AWS Glue job insights to simplify job debugging and optimization for your AWS Glue jobs. For more information, see Monitoring with AWS Glue job insights .	February 8, 2022
Support for crawling Amazon S3 backed Data Catalog tables using a VPC endpoint (p. 1019)	In addition to Amazon S3 data stores, you can configure your Amazon S3 backed Data Catalog tables to be accessed only by an Amazon Virtual Private Cloud environment (Amazon VPC), for security, auditing, or control purposes. For more information, see Crawling an Amazon S3 Data Store or Amazon S3 backed Data Catalog tables using a VPC Endpoint .	February 3, 2022

Support for Lake Formation governed tables (p. 1019)	Added information about AWS Glue support for Lake Formation governed tables, which support ACID transactions, automatic data compaction, and time-travel queries. For more information, see AWS Glue API and the AWS Lake Formation developer guide .	November 30, 2021
New AWS managed policies added for interactive sessions and notebooks (p. 1019)	New managed policies for IAM provided enhanced security for using AWS Glue with interactive sessions and notebooks. For more information, see AWS Managed (Predefined) Policies for AWS Glue .	November 30, 2021
Documentation for public preview features (p. 1019)	Described features available in preview release for AWS Glue and AWS Glue Studio. For more information, see AWS Glue and AWS Glue Studio preview features .	November 23, 2021
Glue Schema Registry now supported with streaming jobs (p. 1019)	You can create streaming jobs that access tables that are part of the Glue Schema Registry. For more information see AWS Glue Schema Registry and Adding Streaming ETL Jobs in AWS Glue .	November 15, 2021
Support for new machine learning features (p. 1019)	Added information about new features for the Find matches machine learning transform, including incremental matching and match scoring. For more information, see Finding Incremental Matches and Estimating the Quality of Matches using Match Confidence Scores .	October 31, 2021
Support for accelerating crawls using Amazon S3 event notifications (p. 1019)	Added information about accelerating crawls using Amazon S3 event notifications. For more information, see Accelerating Crawls Using Amazon S3 Event Notifications .	October 15, 2021

Additional security configuration options related to access-control and VPCs (p. 1019)	Added information about how you can configure new access control permissions on AWS Glue and configuration of VPCs. For more information, see AWS Tags in AWS Glue, Identity-Based Policies (IAM Policies) that Control Settings Using Condition Keys or Context Keys , and Configuring all AWS calls to go through your VPC .	October 13, 2021
Support for VPC endpoint policies (p. 1019)	Added information about support for Virtual Private Cloud (VPC) endpoint policies in AWS Glue. For more information, see AWS Glue and interface VPC endpoints (AWS PrivateLink) .	October 11, 2021
Documented the AWS Glue version support policy (p. 1019)	Added information about the AWS Glue version support policy and the end of life phases for certain AWS Glue versions. For more information, see AWS Glue version support policy .	September 24, 2021
Support for AWS Glue interactive sessions (Private preview) (p. 1019)	(Private preview) Added information about using AWS Glue interactive sessions to run Spark workloads in the cloud from any Jupyter Notebook. Interactive sessions are the preferred method for developing your AWS Glue extract, transform, and load (ETL) code when you use AWS Glue 2.0 or later. For more information, see Setting Up and Running AWS Glue Interactive Sessions for Jupyter Notebook .	August 24, 2021
Support for creating workflows from blueprints (GA) (p. 1019)	Added information about coding common extract, transform, and load (ETL) use cases in blueprints and then creating workflows from blueprints. Enables data analysts to easily create and run complex ETL processes. For more information, see Performing Complex ETL Activities Using Blueprints and Workflows in AWS Glue .	August 23, 2021

Support for AWS Glue version 3.0. (p. 1019)	Added information about support for AWS Glue version 3.0 which supports the Apache Spark 3.0 engine upgrade for running Apache Spark ETL jobs, and other optimizations and upgrades. For more information, see AWS Glue Release Notes and Migrating AWS Glue jobs to AWS Glue version 3.0 . Other features in this release include the AWS Glue shuffle manager, a SIMD vectorized CSV reader, and catalog partition predicates. For more information see AWS Glue Spark shuffle manager with Amazon S3, Format Options for ETL Inputs and Outputs in AWS Glue, and Server-side filtering using catalog partition predicates .	August 18, 2021
Support for starting a workflow with an Amazon EventBridge event (p. 1019)	Added information about how AWS Glue can be an event consumer in an event-driven architecture. For more information, see Starting an AWS Glue Workflow with an Amazon EventBridge Event and Viewing the EventBridge Events That Started a Workflow .	July 14, 2021
Addition of JSON as a supported data format for the AWS Glue Schema Registry (p. 1019)	Added information about JSON as a supported data format (in addition to AVRO). For more information, see AWS Glue Schema Registry .	June 30, 2021
Create AWS Glue streaming jobs without a Data Catalog table (p. 1019)	The <code>create_data_frame_from_options</code> Python function or <code>getSource</code> for Scala scripts support creating streaming ETL jobs that reference the data streams directly instead of requiring a Data Catalog table.	June 15, 2021
AWS Glue Machine Learning transforms now support AWS Key Management Service keys (p. 1019)	You can specify a security configuration or AWS KMS key when configuring AWS Glue Machine Learning transforms with the console, the CLI, or the AWS Glue APIs. For more information, see Using Data Encryption with Machine Learning Transforms and AWS Glue Machine Learning API .	June 15, 2021

Update to the AWSGlueConsoleFullAccess AWS managed policy (p. 1019)	Added information about a minor update to the AWSGlueConsoleFullAccess AWS managed policy. For more information, see AWS Glue Updates to AWS Managed Policies .	June 10, 2021
Support for specifying a value that indicates the table location for the crawler output. (p. 1019)	Added information about specifying a value that indicates the table location when configuring the crawler's output. For more information, see How to specify the table location .	June 4, 2021
Support for crawling a sample of files in a dataset when crawling an Amazon S3 data store (p. 1019)	Added information about how to crawl a sample of files when crawling Amazon S3. For more information, see Crawler Properties .	May 10, 2021
Support for the AWS Glue optimized parquet writer (p. 1019)	Added information about using the AWS Glue optimized parquet writer for DynamicFrames to create or update tables with the parquet classification. For more information, see Creating Tables, Updating Schema, and Adding New Partitions in the Data Catalog from AWS Glue ETL Jobs and Format Options for ETL Inputs and Outputs in AWS Glue .	May 4, 2021
Support for Kafka client authentication passwords (p. 1019)	Added information about how streaming ETL jobs in AWS Glue support SSL client certificate authentication with Apache Kafka stream producers. You can now provide a custom certificate while defining an AWS Glue connection to an Apache Kafka cluster, which AWS Glue will use when authenticating with it. For more information, see AWS Glue Connection Properties and Connection API .	April 28, 2021
Support for consuming data from Amazon Kinesis Data Streams in another account in streaming ETL jobs (p. 1019)	Added information about to create a streaming ETL job to consume data from Amazon Kinesis Data Streams in another account. For more information, see Adding Streaming ETL Jobs in AWS Glue .	March 30, 2021

Support for creating workflows from blueprints (Public Preview) (p. 1019)	(Public preview) Added information about coding common extract, transform, and load (ETL) use cases in blueprints and then creating workflows from blueprints. Enables data analysts to easily create and run complex ETL processes. For more information, see Performing Complex ETL Activities Using Blueprints and Workflows in AWS Glue .	March 22, 2021
Support for column importance metrics for AWS Glue machine learning transforms (p. 1019)	Added information about viewing column importance metrics when working with AWS Glue machine learning transforms. For more information see Working with Machine Learning Transforms on the AWS Glue Console	February 5, 2021
Support for running streaming ETL jobs in Glue version 2.0 (p. 1019)	Added information about support for running streaming ETL jobs in Glue version 2.0. For more information, see Adding Streaming ETL Jobs in AWS Glue .	December 18, 2020
Support for workload partitioning with bounded execution (p. 1019)	Added information about enabling workload partitioning to configure the upper bounds on the dataset size, or the number of files processed on ETL job runs. For more information, see Workload Partitioning with Bounded Execution .	November 23, 2020
Support for enhanced partition management (p. 1019)	Added information about how to use new APIs to add or delete a partition index to/from an existing table. For more information, see Working with Partition Indexes .	November 23, 2020
Support for the AWS Glue Schema Registry (p. 1019)	Added information about using the AWS Glue Schema Registry to centrally discover, control, and evolve schemas. For more information, see AWS Glue Schema Registry .	November 19, 2020
Support for the Grok input format in streaming ETL jobs (p. 1019)	Added information about applying Grok patterns to streaming sources such as log files. For more information, see Applying Grok Patterns to Streaming Sources .	November 17, 2020

Support for adding tags to workflows on the AWS Glue console (p. 1019)	Added information about adding tags when creating a workflow using the AWS Glue console. For more information, see Creating and Building Out a Workflow Using the AWS Glue Console .	October 27, 2020
Support for incremental crawler runs (p. 1019)	Added information about support for incremental crawler runs, which crawl only Amazon S3 folders added since the last run. For more information, see Incremental Crawls .	October 21, 2020
Support for schema detection for streaming ETL data sources. Support for Avro streaming ETL data sources and self-managed Kafka (p. 1019)	Streaming extract, transform, and load (ETL) jobs in AWS Glue can now automatically detect the schema of incoming records and handle schema changes on a per-record basis. Self-managed Kafka data sources are now supported. Streaming ETL jobs now support the Avro format in data sources. For more information, see Streaming ETL in AWS Glue , Defining Job Properties for a Streaming ETL Job , and Notes and Restrictions for Avro Streaming Sources .	October 7, 2020
Support for Crawling MongoDB and DocumentDB data sources (p. 1019)	Added information about support for crawling MongoDB and Amazon DocumentDB (with MongoDB compatibility) data sources. For more information, see Defining Crawlers .	October 5, 2020
Support for FIPS compliance (p. 1019)	Added information about FIPS endpoints for customers who require FIPS 140-2 validated cryptographic modules when accessing data using AWS Glue. For more information, see FIPS Compliance .	September 23, 2020
AWS Glue Studio provides an easy to use visual interface for creating and monitoring jobs (p. 1019)	You can now use a simple graph-based interface to compose jobs that move and transform data and run them on AWS Glue. You can then use the job run dashboard in AWS Glue Studio to monitor ETL execution and ensure that your jobs are operating as intended. For more information, see AWS Glue Studio User Guide .	September 23, 2020

Support for creating table indexes to improve query performance (p. 1019)	Added information about creating table indexes to allow you to retrieve a subset of the partitions from a table. For more information, see Working with Partition Indexes .	September 9, 2020
Support for reduced startup times when running Apache Spark ETL jobs in AWS Glue version 2.0. (p. 1019)	Added information about support for AWS Glue version 2.0 which provides an upgraded infrastructure for running Apache Spark ETL jobs with reduced startup times, changes in logging, and support for specifying additional Python modules at the job level. For more information, see AWS Glue Release Notes and Running Spark ETL Jobs with Reduced Startup Times .	August 10, 2020
Support for limiting the number of concurrent workflow runs. (p. 1019)	Added information about how to limit the number of concurrent workflow runs for a particular workflow. For more information, see Creating and Building Out a Workflow Using the AWS Glue Console .	August 10, 2020
Support for crawling an Amazon S3 data store using a VPC endpoint (p. 1019)	Added information about configuring your Amazon S3 data store to be accessed only by an Amazon Virtual Private Cloud environment (Amazon VPC), for security, auditing, or control purposes. For more information, see Crawling an Amazon S3 Data Store using a VPC Endpoint .	August 7, 2020
Support for resuming workflow runs (p. 1019)	Added information about how to resume workflow runs that only partially completed because one or more nodes (jobs or crawlers) did not complete successfully. For more information, see Repairing and Resuming a Workflow Run .	July 27, 2020
Support for enabling private CA certificates in Kafka connections in AWS Glue. (p. 1019)	Added information about new connection options that support enabling private CA certificates for Kafka connections in AWS Glue. For more information, see Connection Types and Options for ETL in AWS Glue and Special Parameters Used by AWS Glue .	July 20, 2020

Support for reading DynamoDB data in another account (p. 1019)	Added information about AWS Glue support for reading data from another AWS account's DynamoDB table. For more information, see Reading from DynamoDB Data in Another Account .	July 17, 2020
Support for a DynamoDB writer connection in AWS Glue version 1.0 or later (p. 1019)	Added information about support for DynamoDB writer, and new or updated connection options for DynamoDB to read or write. For more information, see Connection Types and Options for ETL in AWS Glue .	July 17, 2020
Support for resource links and for cross-account access control using both AWS Glue and Lake Formation (p. 1019)	Added content about new Data Catalog objects called resource links, and about how to manage sharing Data Catalog resources across accounts with both AWS Glue and AWS Lake Formation. For more information, see Granting Cross-Account Access and Table Resource Links .	July 7, 2020
Support for sampling records when crawling DynamoDB data stores (p. 1019)	Added information about new properties that you can configure when crawling a DynamoDB data store. For more information, see Crawler Properties .	June 12, 2020
Support for stopping a workflow run. (p. 1019)	Added information about how to stop a workflow run for a particular workflow. For more information, see Stopping a Workflow Run .	May 14, 2020
Support for Spark streaming ETL jobs (p. 1019)	Added information about creating extract, transform, and load (ETL) jobs with streaming data sources. For more information, see Adding Streaming ETL Jobs in AWS Glue .	April 27, 2020
Support for creating tables, updating the schema, and adding new partitions in the Data Catalog after running an ETL job (p. 1019)	Added information about how you can enable creating tables, updating the schema, and adding new partitions to see the results of your ETL job in the Data Catalog. For more information, see Creating Tables, Updating Schema, and Adding New Partitions in the Data Catalog from AWS Glue ETL Jobs .	April 2, 2020

Support for specifying a version for the Apache Avro data format as an ETL input and output in AWS Glue (p. 1019)	Added information about specifying a version for the Apache Avro data format as an ETL input and output in AWS Glue. The default version 1.7. You can use the <code>version</code> format option to specify Avro version 1.8 to enable logical reading/writing. For more information, see Format Options for ETL Inputs and Outputs in AWS Glue .	March 31, 2020
Support for the EMRFS S3-optimized committer for writing Parquet data into Amazon S3 (p. 1019)	Added information about how to set a new flag to enable the EMRFS S3-optimized committer for writing Parquet data into Amazon S3 when creating or updating an AWS Glue job. For more information, see Special Parameters Used by AWS Glue .	March 30, 2020
Support for machine learning transforms as a resource managed by AWS resource tags (p. 1019)	Added information about using AWS resource tags to manage and control access to your machine learning transforms in AWS Glue. You can assign AWS resource tags to jobs, triggers, endpoints, crawlers, and machine learning transforms in AWS Glue. For more information, see AWS Tags in AWS Glue .	March 2, 2020
Support for non-overrideable job arguments (p. 1019)	Added information about support for special job parameters that cannot be overridden in triggers or when you run the job. For more information see Adding Jobs in AWS Glue .	February 12, 2020
Support for new transforms to work with datasets in Amazon S3 (p. 1019)	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. 1019)	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 SageMaker notebook (p. 1019)	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
Support for reading from MongoDB and Amazon DocumentDB (with MongoDB Compatibility) (p. 1019)	Added information about new connection types and connection options for reading from and writing to MongoDB and Amazon DocumentDB (with MongoDB Compatibility). For more information, see Connection Types and Options for ETL in AWS Glue .	December 17, 2019
Various corrections and clarifications (p. 1019)	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 security configurations. Added a note that AWS Glue does not support writing to Amazon DynamoDB.	December 9, 2019
Support for custom JDBC drivers (p. 1019)	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 SageMaker notebooks to different development endpoints (p. 1019)	Added information about how you can connect an SageMaker notebook to different development endpoints. Updates to describe the new console action for switching to a new development endpoint, and the new 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 AWS Glue version in machine learning transforms (p. 1019)	Added information about defining the AWS 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. 1019)	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 sub-options 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. 1019)	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. 1019)	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. 1019)	Added information about defining the Glue version in development endpoints. Glue version 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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	Added information about defining the <code>Glue_version</code> in job properties. AWS 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. 1019)	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. 1019)	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 run and track as a 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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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 Data Catalog for Spark SQL jobs (p. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	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 SageMaker notebooks (p. 1019)	Added information about using SageMaker notebooks with AWS Glue development endpoints. For more information, see Managing Notebooks .	October 5, 2018
Support for encryption (p. 1019)	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. 1019)	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. 1019)	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. 1019)	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. 1019)	You can now subscribe to an RSS feed to receive notifications about updates to the <i>AWS Glue Developer Guide</i> .	June 25, 2018
Support delay notifications for jobs (p. 1019)	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. 1019)	Added information about new configuration option for crawlers, MergeNewColumns. For more information, see Configuring a Crawler .	May 7, 2018
Support timeout of jobs (p. 1019)	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 \(p. 1019\)](#)

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*.