AWS Prescriptive Guidance

Getting started with serverless ETL on AWS Glue

AWS Prescriptive Guidance: Getting started with serverless ETL on AWS Glue

Table of Contents

Getting started with serverless ETL on AWS Glue

Dheer Toprani and Adnan Alvee, Amazon Web Services (AWS)

March 2023 (document history (p. 14))

On the Amazon Web Services (AWS) Cloud, AWS Glue is a fully managed serverless environment where you can extract, transform, and load (ETL) data at scale. With AWS Glue, you can categorize data, clean it, enrich it, and move it reliably across various data stores and streams in a cost-effective manner.

AWS Glue is serverless, so you don't have to worry about provisioning or managing servers. With AWS Glue, you pay only for the resources you use, and you can scale up or down as needed.

AWS Glue consists of the following components:

- **AWS Glue ETL** – AWS Glue ETL provides batch and streaming options to extract, transform, and load data from one source to another.
- **AWS Glue Data Catalog** – Data Catalog is a central repository for organizing the metadata of all your data assets. Data Catalog provides a unified interface where you can search, discover, and share data assets across data analytics services.
- **AWS Glue DataBrew** – DataBrew is a no-code data preparation tool that you can use to visually explore, clean, and transform data. You can choose from more than 250 prebuilt transformations to automate data preparation tasks without writing any code.

This guide provides a high-level introduction to AWS Glue, including how it works and how you can get started using it. It covers the key concepts that you need to know before authoring AWS Glue jobs, such as automation, monitoring, and integrating with other AWS services. The *Next steps* (p. 12) section will get you up to speed with writing code in AWS Glue. If you already have some experience using AWS Glue, the *Best practices* (p. 9) section will help you fill in any gaps in your knowledge. By the end of this guide, you will be equipped with the knowledge and resources you need to start using AWS Glue effectively.

AWS Glue ETL

AWS Glue ETL supports extracting data from various sources, transforming it to meet your business needs, and loading it into a destination of your choice. This service uses the Apache Spark engine to distribute big data workloads across worker nodes, enabling faster transformations with in-memory processing.

AWS Glue supports a variety of data sources, including Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, and Amazon Relational Database Service (Amazon RDS). To learn more about supported data sources, see Connection types and options for ETL in AWS Glue.

Authoring in AWS Glue ETL

AWS Glue provides multiple ways to author ETL jobs, depending on your experience and use-case:

- Python shell jobs are designed for running basic ETL scripts written in Python. These jobs run on a single machine, and are better suited for small or medium-sized datasets.
- Apache Spark jobs can be written in either Python or Scala. These jobs use Spark to horizontally scale workloads across many worker nodes, so that they can handle large datasets and complex transformations.
- AWS Glue streaming ETL uses the Apache Spark Structured Streaming engine to transform streaming data in microbatch jobs using exactly-once You can author AWS Glue streaming jobs in either Python or Scala.
- AWS Glue Studio is a visual boxes-and-arrows style interface to make Spark-based ETL accessible to developers who are new to Apache Spark programming.

Data processing units

AWS Glue uses data processing units (DPUs) to measure the compute resources allocated to an ETL job and calculate cost. Each DPU is equivalent to 4 vCPUs and 16 GB memory. DPUs should be allocated to your AWS Glue job depending on its complexity and data volume. Allocating the appropriate amount of DPUs will allow you to balance performance needs with cost constraints.

AWS Glue provides four worker types that are optimized for various workloads:

- G.1X (for memory-intensive workloads)
- G.2X (for workloads with ML transforms)
- G.025X (for low volume and sporadic data streams)
- Standard (for AWS Glue versions 1.0 or earlier; not recommended for later versions of AWS Glue)

Using Python shell

For a Python shell job, you can use either 1 DPU to use 16 GB of memory or 0.0625 DPU to use 1 GB of memory. Python shell is intended for basic ETL jobs with small or medium-sized datasets.

Comparing AWS Glue worker types

The following table shows the different AWS Glue worker types for batch, streaming, and AWS Glue Studio ETL workloads using the Apache Spark environment.

	G.1X	**G.2X**	**G.025X**	**Standard**
vCPU	4	8	2	4
Memory	16 GB	32 GB	4 GB	16 GB
Disk space	64 GB	128 GB	64 GB	50 GB
Executor per worker	1	1	1	2
DPU	1	2	0.25	1

Note that the Standard worker type is not recommended for AWS Glue version 2.0 and later. With AWS Glue Studio, you can use only G.1X and G.2X worker types. The G.025X worker type is available only for AWS Glue version 3.0 streaming jobs.

AWS Glue Data Catalog

The AWS Glue Data Catalog is a centralized metadata repository for all your data assets across various data sources. It provides a unified interface to store and query information about data formats, schemas, and sources. When an AWS Glue ETL job runs, it uses this catalog to understand information about the data and ensure that it is transformed correctly.

The Data Catalog is composed of the following components:

- Databases and tables
- Crawlers and classifiers
- Connections
- Schema Registry

AWS Glue databases and tables

The AWS Glue Data Catalog is organized into databases and tables to provide a logical structure for storing and managing metadata. This structure supports precise data access control at a table or database level by using AWS Identity and Access Management (IAM) policies.

An AWS Glue database can contain many tables, and each table must be associated with a single database. These tables contain references to the actual data, which can be stored in any of the various data sources that AWS Glue supports. AWS Glue tables also store essential metadata such as column names, data types, and partition keys.

There are several different methods for creating a table in AWS Glue:

- AWS Glue crawler
- AWS Glue ETL job
- AWS Glue console
- `CreateTable` operation in the AWS Glue API
- AWS CloudFormation template
- AWS Cloud Development Kit (AWS CDK)
- A migrated Apache Hive metastore

AWS Glue crawlers and classifiers

An AWS Glue crawler automatically discovers and extracts metadata from a data store, and then it updates the AWS Glue Data Catalog accordingly. The crawler connects to the data store to infer the schema of the data. It then creates or updates tables within the Data Catalog with the schema information that it discovered. A crawler can crawl both file-based and table-based data stores. To learn more about supported data stores, see Which data stores can I crawl?

The crawler uses classifiers to accurately recognize the format of data and determine how it should be processed. By default, the crawler uses a set of common built-in classifiers provided by AWS Glue, but you can also write custom classifiers to handle specific use-cases.

AWS Glue connections

You can use AWS Glue connections to define connection parameters that enable AWS Glue to connect to various data sources. Adding connections centralizes and simplifies the configuration required to connect to these sources.

When defining a connection, you specify the connection type, the connection endpoint, and any required credentials. After a connection is defined, it can be reused by multiple AWS Glue jobs and crawlers. Using connections with AWS Glue reduces the need to repeatedly enter the same connection information, such as login credentials or virtual private cloud (VPC) IDs.

AWS Glue Schema Registry

The AWS Glue Schema Registry provides a centralized location for managing and enforcing data stream schemas. It enables disparate systems, such as data producers and consumers, to share a schema for serialization and deserialization. Sharing a schema helps these systems to communicate effectively and avoid errors during transformation.

The Schema Registry ensures that downstream data consumers can handle changes made upstream, because they are aware of the expected schema. It supports schema evolution, so that a schema can change over time while maintaining compatibility with previous versions of the schema.

The Schema Registry integrates with many AWS services, including Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose, and Amazon Managed Streaming for Apache Kafka. For examples of use cases and integrations, see Integrating with AWS Glue Schema Registry.

Important features and concepts

Logging and monitoring

AWS Glue has several logging and monitoring options. By default, AWS Glue sends logs to the `aws-glue` log group in Amazon CloudWatch. These logs include information such as start and end time, configuration settings, and any errors or warnings that might have occurred.

Additionally, AWS Glue Spark ETL jobs provide the following options, which must be enabled for advanced monitoring:

- Job metrics report job-specific metrics to the AWS Glue namespace in CloudWatch every 30 seconds. These job-specific metrics, such as processed records, total input/output data size, and runtime, provide insights into a job's performance. They can help identify bottlenecks or opportunities to optimize configurations.
- Continuous logging streams real-time Apache Spark job logs to the `/aws-glue/jobs/logs-v2` log group in CloudWatch. By using real-time logs, you can dynamically monitor AWS Glue jobs while they are running.
- Spark UI provides a Spark history server web interface for viewing information about the Spark job, such as the event timeline of each stage, a directed acyclic graph, and job environment variables. The persisted Spark UI event logs are stored in Amazon S3, and you can use them in real time or after the job is complete.
- Job run insights simplifies job debugging and optimization by listening for common Spark exceptions, performing root cause analysis, and providing recommended actions to fix issues. The insights are stored in CloudWatch.

Automation

AWS Glue provides two main ways for you to automate ETL jobs: triggers and workflows.

AWS Glue triggers

When fired, AWS Glue triggers start specified jobs and crawlers. A trigger can be fired on demand, based on a predefined schedule, or based on specific events. You can use triggers to design a chain of dependent jobs and crawlers. For more information, see AWS Glue triggers.

AWS Glue workflows

For more complex workloads, you can use AWS Glue workflows to create directed acyclic graphs and to build dependencies between separate AWS Glue entities (triggers, crawlers, and jobs). Workflows also provide a unified interface where you can share parameters, monitor progress, and troubleshoot issues across associated entities.

Setting up many associated entities within AWS Glue workflows can grow increasingly complex. Developers can create AWS Glue blueprints for sharing complex data pipelines with data scientists and business analysts. These templates allow for the consistent and repeatable creation of AWS Glue workflows, abstracting away the technical details.

To learn more about AWS Glue blueprints and workflows, see Performing complex ETL activities using blueprints and workflows in AWS Glue.

Orchestrating AWS Glue jobs with other AWS services

For more automation options, AWS Glue integrates with other AWS services, such as AWS Lambda, AWS Step Functions, and Amazon Managed Workflow for Apache Airflow (Amazon MWAA).

To compare the different orchestration methods for AWS Glue ETL jobs, see Building an operationally excellent data pipeline.

Job bookmarks

Job bookmarks in AWS Glue are used to keep track of the progress of ETL jobs, which prevents the need to reprocess data in subsequent job runs. When job bookmarks are enabled, AAWS Glue maintains a record of data that has already been processed. Then with each run, it processes only the new data in the data source. For more information, see Tracking processed data using job bookmarks.

AWS Glue DataBrew

AWS Glue DataBrew is a fully managed visual data preparation service for cleaning, normalizing, and transforming data. It differs from AWS Glue ETL in that you don't have write code to work with it. DataBrew provides more than 250 built-in transformations, with a visual point-and-click interface for creating and managing data transformation jobs.

DataBrew is available in a separate console view from AWS Glue. It is natively integrated with several AWS services and supports many different file formats. For more information, see Product and service integrations.

DataBrew is based on the following six core concepts:

Project

The entire data preparation workspace in DataBrew

Dataset

A collection of structured or semi-structured data

Recipe

A set of data transformation steps; each step can contain many actions

Job

A set of instructions to run a recipe or a data profile job

Data lineage

The tracking of data in a visual interface to identify its origin

Data profile

A summary view of the shape of your data

To get started with DataBrew, use the AWS Glue DataBrew sample project tutorial.

Best practices

When developing with AWS Glue, consider the following best practices.

Develop locally first

To save on cost and time while building your ETL jobs, test your code and business logic locally first. For instructions on setting up a Docker container that can help you test AWS Glue ETL jobs both in a shell and in an integrated development environment (IDE), see the blog post Develop and test AWS Glue jobs locally using a Docker container.

Use AWS Glue interactive sessions

AWS Glue interactive sessions provide a serverless Spark backend, coupled with an open-source Jupyter kernel that integrates with notebooks and IDEs such as PyCharm, IntelliJ, and VS Code. By using interactive sessions, you can test your code on real datasets with the AWS Glue Spark backend and the IDE of your choice. To get started, follow the steps in Getting started with AWS Glue interactive sessions.

Use partitioning to query exactly what you need

Partitioning refers to dividing a large dataset into smaller partitions based on specific columns or keys. When data is partitioned, AWS Glue can perform selective scans on a subset of data that satisfies specific partitioning criteria, rather than scanning the entire dataset. This results in faster and more efficient query processing, especially when working with large datasets.

Partition data based on the queries that will be run against it. For example, if most queries filter on a particular column, partitioning on that column can greatly reduce query time. To learn more about partitioning data, see Work with partitioned data in AWS Glue.

Optimize memory management

Memory management is crucial when writing AWS Glue ETL jobs because they run on the Apache Spark engine, which is optimized for in-memory processing. The blog post Optimize memory management in AWS Glue provides details on the following memory management techniques:

- Amazon S3 list implementation of AWS Glue
- Grouping
- Excluding irrelevant Amazon S3 paths and storage classes
- Spark and AWS Glue read partitioning
- Bulk inserts
- Join optimizations
- PySpark user-defined functions (UDFs)
- Incremental processing

Use efficient data storage formats

When authoring ETL jobs, we recommend outputting transformed data in a column-based data format. Columnar data formats, such as Apache Parquet and ORC, are commonly used in big data storage and analytics. They are designed to minimize data movement and maximize compression. These formats also enable splitting data to multiple readers for increased parallel reads during query processing.

Compressing data also helps reduce the amount of data stored, and it improves read/write operation performance. AWS Glue supports multiple compression formats natively. For more information about efficient data storage and compression, see Building a performance efficient data pipeline.

Use the appropriate type of scaling

Understanding when to scale horizontally (change the number of workers) or scale vertically (change the worker type) is important for AWS Glue because it can impact the cost and performance of the ETL jobs.

Generally, ETL jobs with complex transformations are more memory-intensive and require vertical scaling (for example, moving from the G.1X to the G.2X worker type). For compute-intensive ETL jobs with large volumes of data, we recommend horizontal scaling because AWS Glue is designed to process that data in parallel across multiple nodes.

Closely monitoring AWS Glue job metrics in Amazon CloudWatch helps you determine whether a performance bottleneck is caused by a lack of memory or compute. For more information about AWS Glue worker types and scaling, see Best practices to scale Apache Spark jobs and partition data with AWS Glue.

Serverless ETL on AWS GlueFAQ

This section provides answers to commonly raised questions about serverless ETL on AWS Glue.

When should I use AWS Glue Python shell instead of AWS Glue with Spark for AWS Glue jobs?

Use Python shell when you have basic ETL jobs or small datasets that don't require the distributed computing capabilities of Apache Spark. Use Apache Spark for more complex ETL jobs or large datasets that require the high processing power that Spark is optimized to handle.

What is the recommended AWS Glue version for my project?

We generally recommend using the latest version of AWS Glue. The AWS Glue versions page lists the differences between versions, along with their compatibility with various versions of Python and Spark.

Next steps

Understanding AWS Glue transformations

For more efficient data processing, AWS Glue includes built-in transformation functions. The functions pass from transform to transform in a data structure called a DynamicFrame, which is an extension to an Apache Spark SQL DataFrame. A DynamicFrame is similar to a DataFrame, except that each record is self-describing, so no schema is required initially.

To get acquainted with several AWS Glue PySpark built-in functions, see the blog post Building an AWS Glue ETL pipeline locally without an AWS account.

Authoring your first ETL job

If you haven't written an ETL job before, you can get started by using the Three AWS Glue ETL job types for converting data to Apache Parquet pattern.

If you have experience writing ETL jobs, you can use the AWS Glue GitHub examples to explore more deeply.

Pricing

For pricing information, see AWS Glue pricing. You can also use the AWS Pricing Calculator to estimate your monthly cost for using different AWS Glue components.

Additional resources

References

- AWS Glue Developer Guide
- AWS Glue DataBrew Developer Guide
- AWS Glue API
- AWS Glue streaming ETL
- AWS Glue Studio
- Python shell jobs in AWS Glue
- Apache Spark jobs
- AWS Glue PySpark transforms reference
- Data Catalog and crawlers in AWS Glue
- AWS Glue Schema Registry
- Logging and monitoring in AWS Glue
- AWS Glue versions

Blog posts

- Develop and test AWS Glue jobs locally using a Docker container
- Optimize memory management in AWS Glue
- Best practices to scale Apache Spark jobs and partition data with AWS Glue
- Building an AWS Glue ETL pipeline locally without an AWS account
- Work with partitioned data in AWS Glue

Examples

- AWS Glue DataBrew sample project
- Getting started with AWS Glue interactive sessions
- AWS Glue ETL Code Samples repository

Patterns

- Build an ETL service pipeline to load data incrementally from Amazon S3 to Amazon Redshift using AWS Glue
- Deploy and manage a serverless data lake on the AWS Cloud by using infrastructure as code
- Three AWS Glue ETL job types for converting data to Apache Parquet

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Updated (p. 14)	Content updated throughout the guide.	March 15, 2023
Added a best practice (p. 14)	We added information about using partitioning to query for specific data.	February 4, 2021
Initial publication (p. 14)	—	January 29, 2021