Data Engineering Hub

TOP 20 AWS GLUE INTERVIEW Q&A

https://aws.sachin.cloud

Top 20 AWS Data Engineering Glue Interview Q & A

Curated by:

Sachin Chandrashekhar

Founder – Data Engineering Hub

🎯LinkedIn: https://www.linkedin.com/in/sachincw/

🎯WhatsApp Community: https://chat.whatsapp.com/FAqHgo4YpUsLFScpiMvtSF

🎯Top mate link: https://lnkd.in/d28ETqaN

🎯 AWS Program Waitlist: https://masterclass.sachin.cloud

1) What is AWS Glue Catalog?

Answer: AWS Glue Catalog is a fully managed metadata repository that is used to store metadata about data sources, transformations, and targets. It is a central repository for storing table definitions, schema information, and other metadata related to ETL jobs.

2) What is the purpose of the AWS Glue Catalog in the ETL process?

Answer: The AWS Glue Catalog plays a crucial role in the ETL (Extract, Transform, Load) process by providing a centralized metadata store. It stores information about data sources, target destinations, and transformations, enabling the Glue ETL service to discover, catalog, and transform data efficiently.

3) How are schemas defined in the AWS Glue Catalog?

Answer: Schemas in the AWS Glue Catalog are defined by creating metadata tables. Each table contains information about the data's structure, including column names, data types, and other relevant details.

4) Can AWS Glue Catalog be integrated with external data sources?

Answer: Yes, AWS Glue Catalog can be integrated with external data sources. It supports connections to various data stores, including Amazon S3, Amazon RDS, Amazon Redshift, and more. This allows Glue to catalog and process data from different sources.

5) How does AWS Glue Catalog handle changes in schema over time?

Answer: AWS Glue Catalog supports versioning of table definitions. When there are changes in the schema, a new version of the table is created in the catalog, preserving historical versions and allowing for schema evolution.

6) How can you optimize the performance of queries using the AWS Glue Catalog?

Answer: Performance optimization in AWS Glue Catalog involves partitioning tables, using column statistics to improve query planning, and optimizing transformations. Leveraging partitioning and indexing can significantly enhance query performance.

7) Explain the concept of partitioning in AWS Glue Catalog.

Answer: Partitioning in AWS Glue Catalog involves dividing large datasets into smaller, more manageable parts based on specific columns. It improves query performance by allowing the system to skip irrelevant data when querying based on the partition columns.

8) What is a custom classifier in Glue Crawler?

Answer: In AWS Glue, a custom classifier is a user-defined pattern or logic that the Glue Crawler uses to infer the schema of data stored in various formats within a data store. By default, Glue provides built-in classifiers for common data formats like JSON, CSV, Parquet, and others. However, in cases where your data follows a specific format or structure that is not covered by the built-in classifiers, you can create a custom classifier.

Here are the key points about custom classifiers in Glue:

User-Defined Patterns: With a custom classifier, you define patterns or regular expressions that match the structure of your data. This allows the Glue Crawler to identify and understand the schema of your data based on these patterns.

Flexible Schema Inference: Custom classifiers provide flexibility when dealing with data that might not conform to standard formats. They enable you to teach the crawler how to interpret and classify your data correctly.

Support for Multiple Formats: Custom classifiers can be applied to various data formats, including those not covered by the built-in classifiers. This is particularly useful when dealing with proprietary or custom data formats.

Example Use Case: Consider a scenario where your data is stored in log files, and the structure of each log entry follows a specific pattern. By creating a custom classifier, you can instruct the Glue Crawler on how to interpret the log file entries, allowing for accurate schema inference.

Integration with Crawlers: Once a custom classifier is created, you associate it with a Glue Crawler. When the Crawler runs, it uses the custom classifier along with built-in classifiers to infer the schema of the data in your data store.

Using custom classifiers is a way to extend the capabilities of AWS Glue to handle diverse and non-standard data formats. It allows you to tailor the schema inference process to the specific structure of your data, ensuring accurate and meaningful results during the ETL (Extract, Transform, Load) process.

Let's walk through an example of creating a custom classifier in AWS Glue for a hypothetical scenario where your data is stored in log files, and each log entry follows a specific pattern.

Scenario:

Assume you have log files stored in Amazon S3, and each log entry has the following format:

[2023-12-01 10:30:15] INFO: This is a log entry.

Here, the log entry includes a timestamp, log level (INFO in this case), and the log message.

Steps to Create a Custom Classifier:

Go to the AWS Glue Console:

- Open the AWS Management Console and navigate to the AWS Glue service.

Create a Custom Classifier:

- In the Glue Console, go to the "Classifiers" section.

- Click on "Add classifier" and choose "Custom classifier."
- Provide a name for your custom classifier (e.g., LogEntryClassifier).
- Specify the classification patterns using regular expressions. In this case, you might use a pattern like:

  \[\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\] ([A-Z]+): .*

Associate the Custom Classifier with a Crawler:

- After creating the custom classifier, go to the "Crawlers" section in the Glue Console.
- Edit the existing crawler or create a new one that points to your S3 location.
- In the crawler settings, associate the custom classifier you created.

Run the Crawler:

- Execute the crawler to scan and catalog the data in your S3 location.

Result:

When the Glue Crawler runs, it uses the custom classifier along with any built-in classifiers to infer the schema of the log files. The custom classifier helps Glue understand the structure of each log entry, allowing for accurate schema inference.

After the crawl is complete, you can view the cataloged tables in the Glue Data Catalog. The table schema should reflect the timestamp, log level, and message components of each log entry.

This example illustrates how a custom classifier in AWS Glue can be used to handle non-standard data formats, providing flexibility in schema inference for diverse datasets.

9) Explain partition indexes in Glue Catalog with an example.

Answer: In AWS Glue, partition indexes are a feature used in the Glue Data Catalog to improve the efficiency of query performance, especially when dealing with large

datasets. Partition indexes are metadata structures that store information about the partitions of a table, making it easier for the Glue ETL service to skip irrelevant data when executing queries.

Here's an explanation of partition indexes in Glue Catalog with an example:

Example Scenario:

Consider a scenario where you have a large dataset of sales records stored in Amazon S3, and the data is partitioned by the year and month of the sales transactions. Each sales record has attributes such as `product_id`, `sales_amount`, and `transaction_date`.

Without Partition Indexes:

Without partition indexes, querying the entire dataset might involve scanning all the data files, even if you are interested in a specific subset of the data, such as sales records for a particular month or year. This can result in slower query performance and increased resource usage.

With Partition Indexes:

Partitioning Setup:

- In the Glue Console, you define a table in the Data Catalog for the sales records.
- You set up partitioning on the `transaction_date` column, creating partitions for each year and month.

Partition Index Creation:

- When defining the table, you enable partition indexes. Glue will automatically create partition indexes based on the specified partition columns.

```
CREATE EXTERNAL TABLE sales_records (
  product_id INT,
  sales_amount DECIMAL(10, 2),
  transaction_date DATE
```

```
)
PARTITIONED BY (year INT, month INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://your-s3-bucket/sales-data/';
```

Query Optimization:

- With partition indexes in place, when you execute a query that filters data based on the partitioned columns (e.g., querying sales for a specific month or year), Glue can use the partition indexes to skip reading irrelevant data files. This results in faster query execution.

```
SELECT *
FROM sales_records
WHERE year = 2023 AND month = 1;
```

Reduced Data Scanning:

- Instead of scanning the entire dataset, Glue can narrow down the data files that need to be read based on the specified partition values. This significantly reduces the amount of data scanned, improving query performance.

Benefits of Partition Indexes:

- Improved Query Performance: Partition indexes help skip unnecessary data files during query execution, leading to faster query performance.
- Resource Efficiency: By reading only the relevant partitions, the query consumes fewer resources, reducing the cost and time required for query processing.
- Optimized for Filtered Queries: Partitioning is particularly effective when queries involve filtering data based on the partitioned columns.

In summary, partition indexes in the Glue Catalog optimize query performance by allowing the Glue ETL service to skip irrelevant data when querying partitioned tables.

Sachin Chandrashekhar - Data Engineering Hub

LinkedIn: https://www.linkedin.com/in/sachincw/

AWS Program Waitlist: https://masterclass.sachin.cloud

This is especially beneficial for large datasets where selective querying based on partition columns is common.

10) How does “Crawl based on events” work?

Answer: Suppose you want the crawler to crawl and then take necessary resulting actions only based on the new files getting added to the S3 bucket path which the crawler is set to crawl.

You can use the option Crawl based on events and provide a SQS queue to which the events about the new files are sent.

Crawler can then read the events from SQS queue and can then crawl depending on its schedule and then take necessary actions.

For more information, refer -
https://www.youtube.com/watch?v=HrppGyFdPjw&t=1598s

11) What is the difference between SparkContext, Spark session and GlueContext?

Answer: In the context of Apache Spark and AWS Glue, `SparkContext`, `SparkSession`, and `GlueContext` are important components that serve different purposes. Let's discuss each one:

SparkContext:

- `SparkContext` is the entry point for any Spark functionality. It sets up internal services and establishes a connection to a Spark cluster. In traditional Spark applications, you would create a `SparkContext` to interact with the Spark engine.

- In AWS Glue, when using PySpark for scripting, the `SparkContext` is often automatically created for you, and you don't need to explicitly create it. You can typically access it using `sc`.

  ```
  from pyspark.context import SparkContext


  sc = SparkContext.getOrCreate()
  ```

SparkSession:

- `SparkSession` is a higher-level abstraction on top of `SparkContext`. It is the entry point for structured and SQL functionality in Spark. A `SparkSession` is required for creating DataFrames & working with SQL queries
- In AWS Glue scripts, a `SparkSession` is typically automatically created for you, and you can access it using `spark`.

  ```
  from pyspark.sql import SparkSession


  spark = SparkSession.builder.appName("example").getOrCreate()
  ```

GlueContext:

- `GlueContext` is specific to AWS Glue and is part of the Glue ETL (Extract, Transform, Load) service. It extends the functionality of `SparkContext` and `SparkSession` to provide additional features that are specific to AWS Glue.
- `GlueContext` simplifies interactions with the AWS Glue Data Catalog, allowing you to read and write metadata about your data. It also provides additional methods for working with DynamicFrames, which are a Glue-specific abstraction on top of Spark DataFrames.

  ```
  from awsglue.context import GlueContext
  ```

```
from pyspark.context import SparkContext


sc = SparkContext()
glue_context = GlueContext(sc)
```

In summary:

- Use `SparkContext` for low-level Spark functionality to work with RDDs. In AWS Glue scripts, it's often created automatically.
- Use `SparkSession` for higher-level structured Spark functionality, including DataFrames and SQL operations. In AWS Glue scripts, it's often created automatically.
- Use `GlueContext` for additional Glue-specific functionality, especially when working with the AWS Glue Data Catalog and DynamicFrames in ETL jobs.

When working with AWS Glue scripts, you typically interact more with `SparkSession` and `GlueContext` as they provide higher-level abstractions for ETL tasks.

For more info, refer - https://www.youtube.com/watch?v=-Vdgkg-UqfY&pp=ygULZ2x1ZWNvbnRleHQ%3D

12) What is a Glue Dynamic Frame?

Answer: In AWS Glue, a DynamicFrame is a higher-level abstraction on top of Apache Spark DataFrames. It is part of the Glue ETL (Extract, Transform, Load) service and is designed to provide a more flexible and schema-less representation of data compared to traditional Spark DataFrames.

Key characteristics of Glue DynamicFrames include:

Schema Flexibility:

- Unlike Spark DataFrames, DynamicFrames are schema-less, allowing them to handle semi-structured or schema-less data more effectively. This

is particularly useful when dealing with data sources like JSON, where the schema may vary.

Built for ETL:

- DynamicFrames are specifically designed for ETL operations in the AWS Glue environment. They are the primary data structure used in Glue ETL jobs to perform data transformations.

Support for Nested Data Structures:

- Glue DynamicFrames can represent nested data structures more easily than traditional Spark DataFrames. This is advantageous when dealing with complex data formats like JSON or Avro.

Ease of Use:

- Glue DynamicFrames provide a higher-level, more user-friendly API compared to Spark DataFrames. They come with built-in transformations that simplify common ETL tasks.

Integration with Glue Catalog:

- DynamicFrames seamlessly integrate with the AWS Glue Data Catalog, which helps manage metadata and facilitates the discovery of data.

13) Why do we have to convert from dynamic frame to spark dataframe?

Answer: In AWS Glue, DynamicFrames and Spark DataFrames are two representations of distributed data in Apache Spark. Each has its own advantages and use cases. The conversion from a DynamicFrame to a Spark DataFrame, and vice versa, might be necessary in certain scenarios based on the operations and transformations you need to perform in your ETL (Extract, Transform, Load) processes.

Here are some reasons why you might need to convert from a DynamicFrame to a Spark DataFrame:

Spark SQL Operations:

- Spark DataFrames are particularly useful when you need to perform Spark SQL operations or use DataFrame-specific transformations. If you have SQL-like queries or need to leverage Spark's DataFrame API, converting to a DataFrame is beneficial.

Integration with Spark Libraries:

- Some Spark libraries and functions work directly with Spark DataFrames. If you plan to use such libraries or functions in your ETL job, you might need to convert your data to a DataFrame.

Complex Transformations:

- If you need to perform complex transformations that are more easily expressed using Spark SQL or DataFrame operations, converting to a DataFrame can simplify your code.

Parquet and ORC Formats:

- Certain file formats, such as Parquet and ORC, are commonly used with Spark DataFrames. If you plan to write your data to these formats, converting to a DataFrame before writing might be necessary.

Better Integration with Spark Ecosystem:

- In some cases, integration with other components of the Spark ecosystem, such as Spark MLlib for machine learning, is more seamless with Spark DataFrames.

Here is an example of converting from a DynamicFrame to a Spark DataFrame in AWS Glue:

```
# Assuming dynamic_frame is your DynamicFrame
data_frame = dynamic_frame.toDF()
```

Converting from a DataFrame to a DynamicFrame might be necessary when you want to take advantage of AWS Glue-specific transformations, especially when dealing with semi-structured or nested data. AWS Glue provides certain transformations and features that are more easily expressed using DynamicFrames.

```
# Assuming data_frame is your DataFrame and glue_context is your GlueContext
dynamic_frame = glue_context.create_dynamic_frame.fromDF(data_frame, glue_ctx, "dynamic_frame_name")
```

In summary, the choice between DynamicFrames and DataFrames depends on your specific use case and the operations you need to perform. Conversion allows you to leverage the strengths of each representation based on your requirements.

Sachin Chandrashekhar - Data Engineering Hub

LinkedIn: https://www.linkedin.com/in/sachincw/

AWS Program Waitlist: https://masterclass.sachin.cloud

14) What is Glue job bookmark? What is used for?

Answer: AWS Glue job bookmarks are used to keep track of the last processed record in a data source when an AWS Glue job runs. This allows the job to resume processing from where it left off in case of interruptions, failures, or incremental data updates.

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, Amazon Simple Storage Service (Amazon S3) sources.

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:

	• job-bookmark-from <from-value> is the run ID which represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input is ignored. • job-bookmark-to <to-value> is the run ID which represents all the input that was processed until the last successful run before and including the specified run ID. The corresponding input excluding the input identified by the <from-value> is processed by the job. Any input later than this input is also excluded for processing. The job bookmark state is not updated when this option set is specified. The sub-options are optional, however when used both the sub-options needs to be provided.

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.

There is lot more to Job bookmarks and to understand them in detail, I highly recommend the following resources:

Refer:

https://www.youtube.com/watch?v=AX2KC0_RZvA&t=844s

https://www.youtube.com/watch?v=XdkxI6Xs9RA

https://www.youtube.com/watch?v=BbWiomEsw5Y

https://docs.aws.amazon.com/glue/latest/dg/monitor-continuations.html

https://docs.aws.amazon.com/glue/latest/dg/programming-etl-connect-bookmarks.html

15) What are the 4 different types of workers in Glue Spark jobs and Glue Python? How do we determine which type of worker to chose?

Answer: In AWS Glue Spark jobs, there are four types of workers that can be used to process data: G 1X, G 2X, G 3X, and G 4X. G 1X has 4 vCPUs and 16 GB RAM, and the remaining ones have vCPUs and RAM multiplied by this base, with their factors mentioned in their names. So, for e.g. G 3X has 3 times the vCPU and RAM as that of G 1X.

In Glue Python Shell jobs, you cannot choose the number of workers currently. There are only two options to choose from - 1/16 DPU and 1 DPU where DPU stands for Data

Processing Unit. A DPU is a relative measure of processing power that consists of 4 vCPUs of compute capacity and 16 GB of memory.

The choice of worker type depends on factors such as the complexity of the ETL transformations, the size of the dataset, and the performance requirements of the job. When deciding which type of worker to choose for your AWS Glue Spark job, consider the following factors:

- Data Size and Complexity: If you are dealing with a large dataset or complex ETL transformations, you may benefit from choosing a worker type with higher compute resources (e.g., G 2X, G 3X, or G 4X).
- Performance Requirements: If your job has strict performance requirements and you need faster execution, you might choose a worker type with higher compute power.
- Cost Considerations: Worker types with higher compute resources are generally more expensive. Consider your budget constraints and choose a worker type that provides the necessary resources without exceeding your cost limits.
- Trial and Error: In some cases, it may be beneficial to experiment with different worker types to determine the optimal configuration for your specific job. AWS Glue allows you to adjust the worker type easily.

AWS Glue provides flexibility in choosing the appropriate worker type based on the specific requirements of your ETL job, allowing you to balance performance and cost effectively.

Sachin Chandrashekhar - Data Engineering Hub

LinkedIn: https://www.linkedin.com/in/sachincw/

AWS Program Waitlist: https://masterclass.sachin.cloud

16) What is Flex Execution Feature in Glue jobs?

Answer: Flex jobs supports only ETL type jobs, and no streaming sources.

Flexible Execution is ideal for customer workloads that don't require fast jobs start times or consistent execution times. By using Flexible Execution AWS Glue Studio jobs run on spare capacity in AWS instead of dedicated reserved or on-demand EC2 instances.

It is unlikely, but possible, that jobs will run significantly longer due to resource reallocation. We recommend preventing that by setting a shorter timeout; for example, 120 minutes.

Source: AWS Documentation

17) Explain auto scaling of Glue workers.

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

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.

Refer:

https://docs.aws.amazon.com/glue/latest/dg/auto-scaling.html

18) How do you optimize Glue jobs?

Answer: Optimizing AWS Glue jobs is essential to ensure efficient and cost-effective data processing. Here are some tips and best practices for optimizing Glue jobs:

Data Partitioning:

- Use data partitioning to reduce the amount of data that needs to be processed. Partitioning can significantly improve the performance of queries.
- Partition your data based on commonly used filters in your queries to minimize the data scanned.

Crawler Configuration:

- Optimize the Glue crawler settings to avoid unnecessary scans of the entire dataset. Use custom classifiers and exclusion patterns to focus on relevant data.

DynamicFrames and Pushdown Predicates:

- Leverage DynamicFrames to handle semi-structured and nested data efficiently.
- Use pushdown predicates to push filtering operations down to the underlying data storage, reducing the amount of data that needs to be processed.

Choose the Right Worker Type:

- Select an appropriate worker type based on the complexity and size of your ETL job. Experiment with different worker types to find the optimal balance between performance and cost.

Use Parquet or ORC Formats:

- When possible, use columnar storage formats like Parquet or ORC. These formats can significantly improve performance and reduce the amount of data scanned.

Optimize Transformations:

- Minimize unnecessary transformations and filters to reduce the processing time.
- Use built-in functions and expressions whenever possible, as they are optimized for performance.

Use Glue ETL Python and Scala Scripts:

- For complex transformations, consider using Glue ETL Python or Scala scripts instead of relying solely on the visual ETL interface. This can provide more control over the execution flow and optimize performance.

Manage Connections and Partitions in JDBC Sources:

- When dealing with JDBC sources, manage the number of connections and partitions appropriately to avoid overwhelming the source database and to ensure efficient data retrieval.

Monitoring and Logging:

- Regularly monitor job runs and use AWS CloudWatch Logs for detailed logging.
- Identify and address any performance bottlenecks or errors in the job runs.

Provision Adequate Resources:

- Ensure that your Glue job has sufficient resources (memory, CPU, and DPUs) based on the complexity and size of your data. Adjust the worker type and count accordingly.

Always test and iterate on your optimizations to find the best configuration for your specific use case.

19) What are the key components of AWS Glue?

Answer:

- AWS Glue Data Catalog: A centralized metadata repository for data assets.
- Crawlers: Discover and define schema for data sources.
- Jobs: Execute ETL workflows using Spark or custom code.
- Triggers: Schedule or event-based job execution.
- Workflows: 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. The AWS Glue console provides a visual representation of a workflow as a graph.
- Transformers: Built-in and custom data transformation functions.

20) What are the different types of triggers in AWS Glue?

Answer: On-demand, schedule-based, and event-based.

Thank you so much for reading this document. I genuinely wish you all the best in your AWS Data Engineering Interview interviews.

- Sachin Chandrashekhar

Follow me on LinkedIn and click the bell 🔔

🎯LinkedIn: https://www.linkedin.com/in/sachincw/

I conduct Real-world AWS Data Engineering (RADE) Programs.

Get on the waitlist

🎯 AWS RADE Waitlist: https://masterclass.sachin.cloud

I also post updates regularly on

🎯WhatsApp Community: https://chat.whatsapp.com/FAqHgo4YpUsLFScpiMvtSF

Look at other resources at:

🎯Top mate link: https://lnkd.in/d28ETqaN