CertyIQ

CertyIQ

Premium exam material

Get certification quickly with the CertyIQ Premium exam material.
Everything you need to prepare, learn & pass your certification exam easily. Lifetime free updates
First attempt guaranteed success.

https://www.CertyIQ.com

About CertyIQ

We here at CertyIQ eventually got enough of the industry's greedy exam paid for. Our team of IT professionals comes with years of experience in the IT industry Prior to training CertIQ we worked in test areas where we observed the horrors of the paywall exam preparation system.

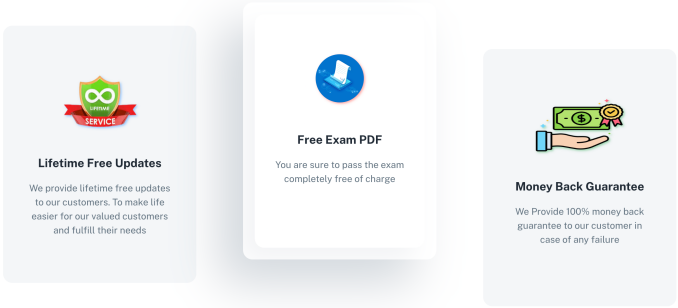
The misuse of the preparation system has left our team disillusioned. And for that reason, we decided it was time to make a difference. We had to make In this way, CertyIQ was created to provide quality materials without stealing from everyday people who are trying to make a living.

Doubt Support

We have developed a very scalable solution using which we are able to solve 400+ doubts every single day with an average rating of 4.8 out of 5.

https://www.certyiq.com

Mail us on - certyiqofficial@gmail.com



John

October 19, 2022

★★★★★

Thanks you so much for your help. I scored 972 in my exam today. More than 90% were from your PDFs!

Dana

September 04, 2022

★★★★★

Thanks a lot for this updated AZ-900 Q&A. I just passed my exam and got 974, I followed both of your Az-900 videos and the 6 PDF, the PDFs are very much valid, all answers are correct. Could you please create a similar video/PDF for DP900, your content/PDF's is really awesome. The team did a really good job. Thank You 😊.

Ahamed Shibly

2 months ago

★★★★★

Customer support is realy fast and helpful, I just finished my exam and this video along with the 6 PDF helped me pass! Definitely recommend getting the PDFs. Thank you!

October 22, 2022

★★★★★

Passed my exam today with 891 marks. Out of 52 questions, 51 were from certyiq PDFs including Contoso case study.
Thank You certyiq team!

Henry Rome

2 months ago

★★★★★

These questions are real and 100 % valid. Thank you so much for your efforts, also your 4 PDFs are awesome, I passed the DP900 exam on 1 Sept. With 968 marks. Thanks a lot, buddy!

Esmaria

2 months ago

★★★★★

Simple easy to understand explanations. To anyone out there wanting to write AZ900, I highly recommend 6 PDF's.Thank you so much, appreciate all your hard work in having such great content. Passed my exam Today - 3 September with 942 score.

Amazon

(AWS Certified Data Engineer - Associate DEA-C01)

AWS Certified Data Engineer - Associate DEA-C01

Total: **207 Questions**

Link: https://certyiq.com/papers/amazon/aws-certified-data-engineer-associate-dea-c01

Question: 1 CertyIQ

A data engineer is configuring an AWS Glue job to read data from an Amazon S3 bucket. The data engineer has set up the necessary AWS Glue connection details and an associated IAM role. However, when the data engineer attempts to run the AWS Glue job, the data engineer receives an error message that indicates that there are problems with the Amazon S3 VPC gateway endpoint.
The data engineer must resolve the error and connect the AWS Glue job to the S3 bucket.
Which solution will meet this requirement?

A.Update the AWS Glue security group to allow inbound traffic from the Amazon S3 VPC gateway endpoint.

B.Configure an S3 bucket policy to explicitly grant the AWS Glue job permissions to access the S3 bucket.

C.Review the AWS Glue job code to ensure that the AWS Glue connection details include a fully qualified domain name.

D.Verify that the VPC's route table includes inbound and outbound routes for the Amazon S3 VPC gateway endpoint.

Answer: D

Explanation:

The error message indicating problems with the Amazon S3 VPC gateway endpoint suggests a network connectivity issue between the AWS Glue job and the S3 bucket within the VPC. VPC gateway endpoints allow resources within a VPC to access S3 without traversing the public internet, enhancing security and reducing costs.

Option A is incorrect because security groups control inbound and outbound traffic to AWS resources (like EC2 instances or Glue ETL endpoints), not the VPC gateway endpoint itself. The endpoint facilitates the connection.

Option B is incorrect because while S3 bucket policies are essential for controlling access, the VPC endpoint's route configuration precedes access control. The Glue job cannot even attempt to access the bucket if routing isn't properly set up. IAM roles handle permission, but routing determines the connection path.

Option C is incorrect because while fully qualified domain names (FQDNs) are generally crucial for resolving network addresses, they are not the core issue when a VPC endpoint connection problem is reported. The core of the problem lies in the gateway endpoint route.

Option D is the correct solution. VPC route tables dictate how traffic is routed within the VPC. To use a VPC endpoint, the route table associated with the subnet where the AWS Glue job is running must include a route that directs traffic destined for S3 (specifically, the prefix list associated with S3) to the VPC gateway endpoint. This ensures that traffic from the Glue job to S3 uses the private, direct connection provided by the endpoint. Without this route, traffic might attempt to go over the internet, bypass the endpoint, or simply fail to resolve. Properly configured inbound and outbound routes for the S3 VPC gateway endpoint are vital for private connectivity within the VPC. Inbound routes ensure requests reach the endpoint, and outbound routes ensure responses return correctly.

Further Reading:

AWS VPC Endpoints: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-endpoints.html
AWS Glue Security Configuration: https://docs.aws.amazon.com/glue/latest/dg/security-getting-access.html

Question: 2 CertyIQ

A retail company has a customer data hub in an Amazon S3 bucket. Employees from many countries use the data hub to support company-wide analytics. A governance team must ensure that the company's data analysts can access data only for customers who are within the same country as the analysts.

Which solution will meet these requirements with the LEAST operational effort?

A.Create a separate table for each country's customer data. Provide access to each analyst based on the country that the analyst serves.

B.Register the S3 bucket as a data lake location in AWS Lake Formation. Use the Lake Formation row-level security features to enforce the company's access policies.

C.Move the data to AWS Regions that are close to the countries where the customers are. Provide access to each analyst based on the country that the analyst serves.

D.Load the data into Amazon Redshift. Create a view for each country. Create separate IAM roles for each country to provide access to data from each country. Assign the appropriate roles to the analysts.

Answer: B

Explanation:

The correct answer is B because it provides a centralized and scalable solution for managing data access based on row-level security using AWS Lake Formation. Lake Formation simplifies the process of building, securing, and managing data lakes. Option B leverages Lake Formation's row-level security feature which allows filtering data based on attributes. In this case, it can restrict access to customer data based on the analyst's country, meeting the governance team's requirement of analysts only seeing data for their respective countries.Option A involves creating separate tables per country, leading to data duplication, increased storage costs, and higher operational overhead for managing multiple tables. This approach is less efficient than using row-level security. Option C involves moving data to different AWS Regions, which adds complexity in data management, data transfer costs, and potential latency issues. Option D suggests using Amazon Redshift and creating views. While viable, it necessitates loading data into Redshift (adding complexity and cost) and managing access via IAM roles. It is also more operationally intensive than Lake Formation's row-level security features, which are specifically designed for this type of access control. Lake Formation provides fine-grained access control without modifying the underlying data storage in S3, minimizing operational effort.

Therefore, using Lake Formation for row-level security is the most efficient and scalable solution for enforcing the required access policies with minimal operational overhead.

Relevant Documentation:

AWS Lake Formation Row-Level Security
AWS Lake Formation Fine-Grained Access Control

Question: 3 CertyIQ

A media company wants to improve a system that recommends media content to customer based on user behavior and preferences. To improve the recommendation system, the company needs to incorporate insights from third-party datasets into the company's existing analytics platform.
The company wants to minimize the effort and time required to incorporate third-party datasets.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use API calls to access and integrate third-party datasets from AWS Data Exchange.

B.Use API calls to access and integrate third-party datasets from AWS DataSync.

C.Use Amazon Kinesis Data Streams to access and integrate third-party datasets from AWS CodeCommit repositories.

D.Use Amazon Kinesis Data Streams to access and integrate third-party datasets from Amazon Elastic Container Registry (Amazon ECR).

Answer: A

Explanation:

The correct answer is A, utilizing AWS Data Exchange for integrating third-party datasets. Here's why:

Justification:

AWS Data Exchange is a service designed specifically for finding, subscribing to, and using third-party data in the cloud. Its primary benefit is the simplification of the data ingestion process. Publishers provide datasets through the exchange, and subscribers (like the media company) can easily access them via API calls or direct download (depending on the publisher's configuration).

Option A minimizes operational overhead because it eliminates the need for complex data pipelines or custom ETL processes. AWS Data Exchange handles the delivery and management of the data. The media company simply needs to integrate the data into their existing analytics platform.

Option B, AWS DataSync, is for transferring large amounts of data between on-premises storage and AWS services, or between AWS services. It is not primarily designed for accessing and integrating third-party datasets available on a marketplace like fashion.

Options C and D, using Amazon Kinesis Data Streams to ingest data from AWS CodeCommit or Amazon ECR, are completely irrelevant. CodeCommit is for version control of code, and ECR is a container registry. Neither is a source for third-party data, and Kinesis Data Streams is designed for real-time streaming data, which is unnecessary for the scenario of incorporating external datasets.

Therefore, using API calls to access third-party datasets from AWS Data Exchange provides the simplest and most efficient way to incorporate external data into the company's analytics platform with minimal operational overhead. AWS Data Exchange handles the complexity of data acquisition and delivery, allowing the company to focus on deriving insights from the data.

Authoritative Links:

AWS Data Exchange: https://aws.amazon.com/data-exchange/
AWS DataSync: https://aws.amazon.com/datasync/
Amazon Kinesis Data Streams: https://aws.amazon.com/kinesis/data-streams/
AWS CodeCommit: https://aws.amazon.com/codecommit/
Amazon ECR: https://aws.amazon.com/ecr/

Question: 4 CertyIQ

A financial company wants to implement a data mesh. The data mesh must support centralized data governance, data analysis, and data access control. The company has decided to use AWS Glue for data catalogs and extract, transform, and load (ETL) operations.
Which combination of AWS services will implement a data mesh? (Choose two.)

A.Use Amazon Aurora for data storage. Use an Amazon Redshift provisioned cluster for data analysis.
B.Use Amazon S3 for data storage. Use Amazon Athena for data analysis.
C.Use AWS Glue DataBrew for centralized data governance and access control.
D.Use Amazon RDS for data storage. Use Amazon EMR for data analysis.
E.Use AWS Lake Formation for centralized data governance and access control.

Answer: BE

Explanation:

The correct answer is BE. Here's why:

B. Use Amazon S3 for data storage. Use Amazon Athena for data analysis.

Amazon S3: S3 is a highly scalable, durable, and cost-effective object storage service ideal for storing large volumes of structured and unstructured data in a data lake architecture. Data meshes often rely on decentralized data ownership, and S3 allows each domain to store their data independently in separate buckets or prefixes. https://aws.amazon.com/s3/
Amazon Athena: Athena is a serverless query service that allows you to analyze data directly in S3 using standard SQL. This is critical for data analysis in a data mesh, as it allows data consumers to query data from various domains stored in S3 without the need for data warehousing. Its serverless nature reduces operational overhead. https://aws.amazon.com/athena/
E. Use AWS Lake Formation for centralized data governance and access control.

AWS Lake Formation: Lake Formation simplifies building, securing, and managing data lakes. A key tenet of a data mesh is federated governance, and Lake Formation facilitates this by providing centralized control over data access, security, and compliance across the data lake. It integrates with AWS Glue for data cataloging and allows you to define fine-grained permissions on data in S3. Centralized governance becomes possible because Lake Formation offers fine grained access control over data stored in S3.
https://aws.amazon.com/lake-formation/

Here's why the other options are less suitable:

A. Use Amazon Aurora for data storage. Use an Amazon Redshift provisioned cluster for data analysis. Aurora is suitable for transactional workloads, not generally for storing data in a data lake environment like in S3, although Aurora can be used in conjunction with S3. A Redshift provisioned cluster requires capacity planning and ongoing management, conflicting with the ease of use and scalability desired in a data mesh.
C. Use AWS Glue DataBrew for centralized data governance and access control. While Glue DataBrew can be used to cleanse and normalize data, it does not provide the centralized data governance and access control capabilities on the scale of a data lake in S3 that Lake Formation does. It's designed more for data preparation, not comprehensive governance.
D. Use Amazon RDS for data storage. Use Amazon EMR for data analysis. RDS is designed for relational databases, and like Aurora, not for storing data lakes as efficiently as S3. EMR is powerful for big data processing but might be overkill and more complex to manage compared to Athena for simpler querying use cases common in a data mesh. EMR focuses on processing, while Athena focuses on querying.

Question: 5 CertyIQ

A data engineer maintains custom Python scripts that perform a data formatting process that many AWS Lambda functions use. When the data engineer needs to modify the Python scripts, the data engineer must manually update all the Lambda functions.
The data engineer requires a less manual way to update the Lambda functions.
Which solution will meet this requirement?

A.Store a pointer to the custom Python scripts in the execution context object in a shared Amazon S3 bucket.
B.Package the custom Python scripts into Lambda layers. Apply the Lambda layers to the Lambda functions.
C.Store a pointer to the custom Python scripts in environment variables in a shared Amazon S3 bucket.
D.Assign the same alias to each Lambda function. Call reach Lambda function by specifying the function's alias.

Answer: B

Explanation:

The correct answer is **B. Package the custom Python scripts into Lambda layers. Apply the Lambda layers to the Lambda functions.**

Lambda layers offer a way to centrally manage and share code dependencies across multiple Lambda functions. Instead of embedding the custom Python scripts directly within each Lambda function's

deployment package, the data engineer can package these scripts as a Lambda layer. Then, each Lambda function can be configured to use this layer.

When the Python scripts need to be updated, the data engineer only needs to update the Lambda layer. All Lambda functions using that layer will automatically receive the updated code upon their next invocation, without requiring individual redeployments. This significantly reduces the manual effort involved in updating multiple functions.

Option A is incorrect because storing pointers to scripts in S3 would require each Lambda function to fetch the script dynamically at runtime. This adds latency, complexity in terms of error handling (S3 unavailability), and requires each function to have the necessary S3 permissions. It also doesn't address dependency management.

Option C is similar to Option A in that it relies on external storage and introduces runtime dependencies. Environment variables are intended for configuration parameters, not code.

Option D, aliases, are primarily for version management and traffic routing, not for sharing code dependencies. While aliases can point to different versions of a Lambda function, they don't address the underlying problem of code duplication. Each version would still need to contain the shared Python scripts.

Lambda Layers streamline dependency management, reduce deployment package size for individual functions, and simplify updates across numerous functions, making it the best solution.

Refer to the AWS documentation for more details:

AWS Lambda Layers
Using Lambda Layers

Question: 6 CertyIQ

A company created an extract, transform, and load (ETL) data pipeline in AWS Glue. A data engineer must crawl a table that is in Microsoft SQL Server. The data engineer needs to extract, transform, and load the output of the crawl to an Amazon S3 bucket. The data engineer also must orchestrate the data pipeline.
Which AWS service or feature will meet these requirements MOST cost-effectively?

A.AWS Step Functions
B.AWS Glue workflows
C.AWS Glue Studio
D.Amazon Managed Workflows for Apache Airflow (Amazon MWAA)

Answer: B

Explanation:

Here's a detailed justification for why AWS Glue workflows are the most cost-effective solution:

The scenario requires crawling a SQL Server table, ETL processing using AWS Glue, storing the output in S3, and orchestrating the entire pipeline.

AWS Glue Crawlers: Glue Crawlers are designed to discover the schema of data sources like Microsoft SQL Server and register them in the AWS Glue Data Catalog. This addresses the need to crawl the table and prepare its metadata for ETL.

AWS Glue Jobs: Glue jobs are used to perform the ETL operations, extracting data from SQL Server (using the crawler's metadata as a source), transforming it, and loading it into the S3 bucket.

AWS Glue Workflows: Glue Workflows are a feature within AWS Glue specifically designed for orchestrating Glue Crawlers and Glue Jobs. They allow you to define dependencies and trigger sequential or parallel execution of these tasks in a managed, serverless environment. Workflows manage dependencies effectively.

Now, let's compare the options:

A. AWS Step Functions: Step Functions can orchestrate various AWS services, including Glue, but it adds an extra layer of complexity and cost. It is better suited for orchestrating more complex, heterogeneous workloads involving multiple AWS services beyond just Glue itself. For a pure Glue-based pipeline, Glue workflows are more efficient.

B. AWS Glue workflows: This option directly addresses the need for orchestration within the AWS Glue ecosystem, managing the crawling and ETL process efficiently. This reduces operational overhead, provides monitoring, and offers a serverless orchestration environment tailored for Glue tasks.

C. AWS Glue Studio: Glue Studio provides a visual interface for building ETL jobs, but it doesn't directly handle the orchestration of the entire pipeline (crawl, ETL, load). While you can create the ETL job itself in Glue Studio, you still need a separate orchestration tool.

D. Amazon Managed Workflows for Apache Airflow (Amazon MWAA): MWAA is a fully managed Apache Airflow service. While Airflow is a powerful orchestration tool, it is considerably more complex and expensive than Glue workflows for a simple Glue-centric pipeline. MWAA is generally preferred for organizations already using Airflow or requiring the full flexibility and extensibility of Airflow's ecosystem.

Considering the cost-effectiveness requirement, AWS Glue workflows are the optimal choice because they provide a managed orchestration service that is tightly integrated with AWS Glue Crawlers and Jobs, avoiding the overhead of managing a separate orchestration service like Step Functions or MWAA. Also, orchestration is the native intention of AWS Glue Workflows.

Supporting links:

AWS Glue Workflows: https://docs.aws.amazon.com/glue/latest/dg/workflows-using.html
AWS Glue Pricing: https://aws.amazon.com/glue/pricing/
AWS Step Functions Pricing: https://aws.amazon.com/step-functions/pricing/
Amazon MWAA Pricing: https://aws.amazon.com/managed-workflows-for-apache-airflow/pricing/

Question: 7 CertyIQ

A financial services company stores financial data in Amazon Redshift. A data engineer wants to run real-time queries on the financial data to support a web-based trading application. The data engineer wants to run the queries from within the trading application.
Which solution will meet these requirements with the LEAST operational overhead?

A.Establish WebSocket connections to Amazon Redshift.
B.Use the Amazon Redshift Data API.
C.Set up Java Database Connectivity (JDBC) connections to Amazon Redshift.
D.Store frequently accessed data in Amazon S3. Use Amazon S3 Select to run the queries.

Answer: B

Explanation:

The correct answer is B, using the Amazon Redshift Data API. Here's why:

Real-time Queries: The scenario necessitates a solution that can execute queries quickly to support the web-based trading application.

Least Operational Overhead: The key is minimizing the burden on the data engineer for managing and maintaining the connection infrastructure.

Redshift Data API: This API provides a serverless, HTTP-based interface to execute SQL commands on Amazon Redshift clusters. It removes the need to manage persistent connections, connection pools, or drivers within the trading application, simplifying development and reducing operational overhead. You simply invoke the API with your SQL statement and retrieve the results.

WebSocket Connections (Option A): While WebSockets can provide real-time communication, managing them directly with Amazon Redshift would require significant custom development to handle authentication, connection management, and error handling. This adds complexity and operational overhead.

JDBC Connections (Option C): JDBC requires the application to maintain connections to the Redshift cluster. This involves managing connection pools, dealing with connection failures, and potentially impacting the Redshift cluster's performance due to the overhead of managing a large number of persistent connections. While viable, it's more complex to manage than the Data API.

Amazon S3 Select (Option D): S3 Select is suitable for querying data directly within S3 but requires you to first load your Redshift data into S3. This adds an ETL process and is not efficient or appropriate for "real-time" queries on the data already residing in Redshift. It would also be slower compared to querying directly within the Redshift data warehouse.

In summary, the Redshift Data API offers the simplest and most efficient way to run real-time queries from an application on Amazon Redshift with minimal overhead. It provides a serverless, managed interface for executing SQL commands without the complexities of managing persistent connections.

Authoritative Links:

Amazon Redshift Data API: https://docs.aws.amazon.com/redshift/latest/dg/data-api.html
AWS Documentation on JDBC connections to Redshift: https://docs.aws.amazon.com/redshift/latest/dg/java-jdbc.html
Amazon S3 Select: https://docs.aws.amazon.com/AmazonS3/latest/userguide/selecting-content-from-objects.html

Question: 8 CertyIQ

A company uses Amazon Athena for one-time queries against data that is in Amazon S3. The company has several use cases. The company must implement permission controls to separate query processes and access to query history among users, teams, and applications that are in the same AWS account.
Which solution will meet these requirements?

A.Create an S3 bucket for each use case. Create an S3 bucket policy that grants permissions to appropriate individual IAM users. Apply the S3 bucket policy to the S3 bucket.

B.Create an Athena workgroup for each use case. Apply tags to the workgroup. Create an IAM policy that uses the tags to apply appropriate permissions to the workgroup.

C.Create an IAM role for each use case. Assign appropriate permissions to the role for each use case. Associate the role with Athena.

D.Create an AWS Glue Data Catalog resource policy that grants permissions to appropriate individual IAM users for each use case. Apply the resource policy to the specific tables that Athena uses.

Answer: B

Explanation:

The correct answer is B. Here's a detailed justification:

Athena workgroups are designed to isolate queries and their history, providing a logical grouping for users, teams, or applications. Each workgroup has its own settings, including query result location, query metrics, and cost controls. Creating a workgroup for each use case directly addresses the requirement to separate query processes.

IAM policies can be used to control access to Athena workgroups based on tags. Tagging workgroups allows you to create granular permissions, ensuring that only authorized users or applications can access specific workgroups. This mechanism facilitates the separation of access based on use case, satisfying the permission control requirement. The condition elements in an IAM policy will use the tag keys to assign permissions to specific workgroups that the requesting principal has access to.

Option A is not ideal. Using S3 buckets to separate data can work, but it doesn't address the need to separate query processes or query history. Also, managing bucket policies for individual IAM users is cumbersome and doesn't scale well.

Option C is insufficient. While IAM roles are important for granting Athena permissions, creating a separate role for each use case doesn't directly separate query processes and history within Athena itself. It may address general data access permissions but falls short of the fine-grained control needed for Athena-specific usage.

Option D might seem viable, but Glue Data Catalog resource policies mainly govern access to the metadata (tables, databases) rather than the query history and query execution context. It also does not separate query processes. While important for data access, it doesn't fully address the problem of separating query processes.

Therefore, the best solution is to utilize Athena workgroups coupled with tag-based IAM policies to achieve the desired separation of query processes, query history, and access control for each use case.

Here are some links for further reading:

Amazon Athena Workgroups
Identity-based policy examples for Amazon Athena
AWS Resource Tags

Question: 9 CertyIQ

A data engineer needs to schedule a workflow that runs a set of AWS Glue jobs every day. The data engineer does not require the Glue jobs to run or finish at a specific time.
Which solution will run the Glue jobs in the MOST cost-effective way?

A.Choose the FLEX execution class in the Glue job properties.
B.Use the Spot Instance type in Glue job properties.
C.Choose the STANDARD execution class in the Glue job properties.
D.Choose the latest version in the GlueVersion field in the Glue job properties.

Answer: A

Explanation:

The most cost-effective solution for running AWS Glue jobs daily without strict timing requirements is to utilize the FLEX execution class. The FLEX execution class in AWS Glue is designed for workloads that are less time-sensitive and can tolerate variations in execution time. This allows AWS Glue to leverage spare capacity, which translates into significant cost savings.

Options B, C, and D are not the most cost-effective. While using Spot Instances (option B) could save money,

Glue jobs do not have this configuration option directly, and Spot Instances can be terminated, leading to job failures. Moreover, this strategy introduces complexity related to handling potential interruptions. The STANDARD execution class (option C) doesn't offer any cost optimization specifically. Specifying the latest Glue version (option D) ensures you have the latest features and bug fixes, but it doesn't directly impact the cost of running the job.

The FLEX execution class allows AWS Glue to run your jobs when resources are most available and affordable. This is beneficial when specific start or end times aren't critical, matching the use case mentioned in the question. Therefore, option A is the best strategy for cost optimization in this scenario. The other options might improve stability or ensure up-to-date software but do not offer the same level of cost efficiency.https://aws.amazon.com/glue/pricing/https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-execution.html

Question: 10 CertyIQ

A data engineer needs to create an AWS Lambda function that converts the format of data from .csv to Apache Parquet. The Lambda function must run only if a user uploads a .csv file to an Amazon S3 bucket.
Which solution will meet these requirements with the LEAST operational overhead?

A.Create an S3 event notification that has an event type of s3:ObjectCreated:*. Use a filter rule to generate notifications only when the suffix includes .csv. Set the Amazon Resource Name (ARN) of the Lambda function as the destination for the event notification.

B.Create an S3 event notification that has an event type of s3:ObjectTagging:* for objects that have a tag set to .csv. Set the Amazon Resource Name (ARN) of the Lambda function as the destination for the event notification.

C.Create an S3 event notification that has an event type of s3:*. Use a filter rule to generate notifications only when the suffix includes .csv. Set the Amazon Resource Name (ARN) of the Lambda function as the destination for the event notification.

D.Create an S3 event notification that has an event type of s3:ObjectCreated:*. Use a filter rule to generate notifications only when the suffix includes .csv. Set an Amazon Simple Notification Service (Amazon SNS) topic as the destination for the event notification. Subscribe the Lambda function to the SNS topic.

Answer: A

Explanation:

Here's a detailed justification for why option A is the most suitable solution for triggering a Lambda function on S3 .csv uploads with minimal operational overhead:

Option A directly leverages S3 event notifications for object creation, specifically the s3:ObjectCreated:* event type. This ensures that the Lambda function is invoked only when a new object is uploaded to the S3 bucket. Crucially, the filter rule based on the .csv suffix refines this trigger, ensuring that only relevant file uploads trigger the function. This prevents unnecessary Lambda invocations and reduces costs. Direct invocation through the ARN keeps things simple and efficient.

Option B is incorrect because s3:ObjectTagging:* is for events related to object tags, which adds unnecessary complexity of setting tags on objects during upload. The scenario requires reacting to the file extension, not object tags.

Option C is incorrect because the s3:* event type is too broad and will trigger the Lambda function for all S3 events, including those unrelated to object creation (e.g., deletions, modifications). This results in unnecessary Lambda invocations and increased costs.

Option D is incorrect because while it uses the correct s3:ObjectCreated:* event type and suffix filter, it introduces an SNS topic as an intermediary. Using SNS adds an extra layer of complexity and potential points of failure. Direct invocation is more efficient.

Therefore, option A provides the most direct and efficient solution for triggering the Lambda function only when a .csv file is uploaded to the S3 bucket, minimizing operational overhead and costs compared to other options. It directly links the event to the Lambda function using the ARN for simplicity and performance.

Supporting Concepts and Links:

S3 Event Notifications: These are a core mechanism for triggering actions based on S3 events. https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html

Lambda Event Sources: S3 is a common event source for Lambda functions, enabling serverless processing of uploaded data. https://docs.aws.amazon.com/lambda/latest/dg/services-s3.html

S3 Event Filtering: Using prefix/suffix filters on S3 event notifications is an efficient way to narrow down the scope of triggered events. https://docs.aws.amazon.com/AmazonS3/latest/userguide/notification-content-structure.html

Question: 11 CertyIQ

A data engineer needs Amazon Athena queries to finish faster. The data engineer notices that all the files the Athena queries use are currently stored in uncompressed .csv format. The data engineer also notices that users perform most queries by selecting a specific column.
Which solution will MOST speed up the Athena query performance?

A.Change the data format from .csv to JSON format. Apply Snappy compression.

B.Compress the .csv files by using Snappy compression.

C.Change the data format from .csv to Apache Parquet. Apply Snappy compression.

D.Compress the .csv files by using gzip compression.

Answer: C

Explanation:

The correct answer is C because it combines columnar storage and compression, which are both highly effective in optimizing Athena query performance, especially given the scenario described.

Here's a detailed justification:

1. **Columnar Storage (Parquet):** Athena works best with columnar storage formats like Apache Parquet. In a columnar format, data for each column is stored contiguously. When users query a specific column, Athena only needs to read the data for that column, significantly reducing I/O operations compared to row-oriented formats like CSV. This is especially relevant because the data engineer observed that users primarily select specific columns.

2. **Compression (Snappy):** Compression reduces the size of the data that needs to be read from storage, leading to faster query execution. Snappy compression is known for its high speed and is a good compromise between compression ratio and processing speed. Applying Snappy compression alongside Parquet ensures efficient storage and retrieval of data.

3. **Why other options are less optimal:**

   A (JSON and Snappy): JSON is a row-oriented format. It doesn't offer the same performance benefits as columnar storage when querying specific columns.
   B (CSV and Snappy): Compressing CSV files will reduce the storage footprint and improve read speeds to some extent, but it doesn't address the inefficiencies of the row-oriented CSV format when querying specific columns.

D (CSV and Gzip): Gzip provides better compression than Snappy, but it's generally slower. While it would reduce storage costs further, the improved compression comes at the cost of processing speed, which is less desirable than Parquet + Snappy. More importantly, it does not address the fundamental performance issue of a row-oriented format.

In summary, changing the data format to Apache Parquet allows Athena to efficiently read only the necessary columns. Applying Snappy compression reduces data volume, further accelerating query performance. Therefore, converting to Parquet and applying Snappy compression is the most effective approach.

Authoritative Links:

Amazon Athena Best Practices: https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/
Apache Parquet: https://parquet.apache.org/
Snappy Compression: https://github.com/google/snappy

Question: 12 CertyIQ

A manufacturing company collects sensor data from its factory floor to monitor and enhance operational efficiency. The company uses Amazon Kinesis Data Streams to publish the data that the sensors collect to a data stream. Then Amazon Kinesis Data Firehose writes the data to an Amazon S3 bucket.
The company needs to display a real-time view of operational efficiency on a large screen in the manufacturing facility.
Which solution will meet these requirements with the LOWEST latency?

A.Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to process the sensor data. Use a connector for Apache Flink to write data to an Amazon Timestream database. Use the Timestream database as a source to create a Grafana dashboard.

B.Configure the S3 bucket to send a notification to an AWS Lambda function when any new object is created. Use the Lambda function to publish the data to Amazon Aurora. Use Aurora as a source to create an Amazon QuickSight dashboard.

C.Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to process the sensor data. Create a new Data Firehose delivery stream to publish data directly to an Amazon Timestream database. Use the Timestream database as a source to create an Amazon QuickSight dashboard.

D.Use AWS Glue bookmarks to read sensor data from the S3 bucket in real time. Publish the data to an Amazon Timestream database. Use the Timestream database as a source to create a Grafana dashboard.

Answer: A

Explanation:

The most suitable solution for displaying real-time operational efficiency data with the lowest latency involves a combination of Flink, Timestream, and Grafana. Here's a detailed justification:

A. Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to process the sensor data. Use a connector for Apache Flink to write data to an Amazon Timestream database. Use the Timestream database as a source to create a Grafana dashboard.

Lowest Latency Processing: Apache Flink excels at real-time stream processing. Using Flink to analyze sensor data directly from Kinesis Data Streams minimizes delays compared to batch-oriented approaches or triggering processes upon file creation in S3. It processes data as it arrives.
Time Series Database: Amazon Timestream is specifically designed for time-series data, making it ideal for storing sensor data collected over time. Its optimized for high ingestion rates and fast queries on time-based data.
Real-Time Visualization: Grafana is well-suited for visualizing time-series data from Timestream. It offers flexible dashboarding capabilities for real-time monitoring.

Why other options are less suitable:

B. S3 notifications + Lambda + Aurora + QuickSight: This approach introduces latency due to the involvement of S3 event notifications, Lambda execution, and writing to Aurora. Aurora is a general-purpose relational database and less optimized for the characteristics of time series data.
C. Flink + Data Firehose to Timestream + QuickSight: Although this option uses Flink and Timestream, inserting Firehose between Flink and Timestream increases latency. Firehose is designed for buffering and batching data, which goes against the need for real-time updates. Firehose introduces latency because it buffers data before writing it to the destination.
D. Glue bookmarks + S3 + Timestream + Grafana: Glue bookmarks are not designed for real-time or near-real-time data extraction. Glue is a batch-oriented ETL service that polls for changes to the data source.

Conclusion:

Option A uses a combination of services that are specifically designed for real-time stream processing (Flink), time-series data storage (Timestream), and real-time visualization (Grafana), resulting in the lowest latency.

Authoritative Links:

Amazon Managed Service for Apache Flink: https://aws.amazon.com/flink/
Amazon Timestream: https://aws.amazon.com/timestream/
Grafana: https://grafana.com/

Question: 13 CertyIQ

A company stores daily records of the financial performance of investment portfolios in .csv format in an Amazon S3 bucket. A data engineer uses AWS Glue crawlers to crawl the S3 data.
The data engineer must make the S3 data accessible daily in the AWS Glue Data Catalog.
Which solution will meet these requirements?

A.Create an IAM role that includes the AmazonS3FullAccess policy. Associate the role with the crawler. Specify the S3 bucket path of the source data as the crawler's data store. Create a daily schedule to run the crawler. Configure the output destination to a new path in the existing S3 bucket.

B.Create an IAM role that includes the AWSGlueServiceRole policy. Associate the role with the crawler. Specify the S3 bucket path of the source data as the crawler's data store. Create a daily schedule to run the crawler. Specify a database name for the output.

C.Create an IAM role that includes the AmazonS3FullAccess policy. Associate the role with the crawler. Specify the S3 bucket path of the source data as the crawler's data store. Allocate data processing units (DPUs) to run the crawler every day. Specify a database name for the output.

D.Create an IAM role that includes the AWSGlueServiceRole policy. Associate the role with the crawler. Specify the S3 bucket path of the source data as the crawler's data store. Allocate data processing units (DPUs) to run the crawler every day. Configure the output destination to a new path in the existing S3 bucket.

Answer: B

Explanation:

The correct answer is B. Here's why:

AWS Glue Service Role: AWS Glue requires specific permissions to access data stores like S3 and write metadata to the AWS Glue Data Catalog. The AWSGlueServiceRole IAM policy provides these necessary permissions. Using AmazonS3FullAccess is overly permissive and violates the principle of least privilege.
https://docs.aws.amazon.com/glue/latest/dg/glue-security.html

Crawler Configuration: The AWS Glue crawler needs to know the location of the data. Specifying the S3 bucket path as the crawler's data store tells the crawler where to find the .csv files.

Scheduled Execution: The requirement is to make the data accessible daily. Scheduling the crawler to run daily ensures that any changes to the .csv files in S3 are reflected in the Glue Data Catalog.

Data Catalog Output: Specifying a database name for the output tells the crawler where to store the metadata (table definitions) discovered from the .csv files. This makes the data accessible for querying and other data processing tasks using services like Athena or Redshift Spectrum.

Why other options are incorrect:

A and C: Using AmazonS3FullAccess is overly permissive. Allocating DPUs daily isn't directly related to scheduling; DPUs determine the computational power for the crawler. While allocating DPUs is necessary for the crawler to function, the question is about the best way to keep the catalog updated daily. Specifying a new path is also incorrect, since the crawler updates metadata.
D: While using the AWSGlueServiceRole is correct, allocating DPUs daily isn't directly related to setting the schedule for the crawler execution. The issue is about scheduling, not about resource allocation. The crawler updates metadata, so specifying a new output path is incorrect.

Question: 14 CertyIQ

A company loads transaction data for each day into Amazon Redshift tables at the end of each day. The company wants to have the ability to track which tables have been loaded and which tables still need to be loaded.
A data engineer wants to store the load statuses of Redshift tables in an Amazon DynamoDB table. The data engineer creates an AWS Lambda function to publish the details of the load statuses to DynamoDB.
How should the data engineer invoke the Lambda function to write load statuses to the DynamoDB table?

A.Use a second Lambda function to invoke the first Lambda function based on Amazon CloudWatch events.

B.Use the Amazon Redshift Data API to publish an event to Amazon EventBridge. Configure an EventBridge rule to invoke the Lambda function.

C.Use the Amazon Redshift Data API to publish a message to an Amazon Simple Queue Service (Amazon SQS) queue. Configure the SQS queue to invoke the Lambda function.

D.Use a second Lambda function to invoke the first Lambda function based on AWS CloudTrail events.

Answer: B

Explanation:

Here's a detailed justification for why option B is the most appropriate solution for invoking the Lambda function and updating the DynamoDB table with Redshift load statuses:

Option B: Use the Amazon Redshift Data API to publish an event to Amazon EventBridge. Configure an EventBridge rule to invoke the Lambda function.

This approach offers a loosely coupled, event-driven architecture that is highly scalable and maintainable. Here's a breakdown of why this is the best solution:

1. **Redshift Data API for Event Emission:** The Redshift Data API enables you to interact with Redshift clusters programmatically without the need for direct JDBC/ODBC connections within your Lambda function. Critically, it can be configured to publish events on command completion. This allows Redshift to signal when a table load operation completes, making it an ideal trigger for our workflow.
2. **Amazon EventBridge for Event Routing:** EventBridge is a serverless event bus service that simplifies the building of event-driven applications. It allows you to define rules that match specific event patterns and route them to different targets, like Lambda functions. In this scenario, we can configure EventBridge to listen for events emitted by the Redshift Data API upon completion of a load operation for a specific table.

3. **Lambda Invocation by EventBridge:** Once EventBridge receives an event matching our defined rule, it will automatically invoke the Lambda function. The event data (containing information about the table, load status, timestamp, etc.) will be passed to the Lambda function as an argument.

4. **DynamoDB Update:** Inside the Lambda function, the event data can be parsed, and the corresponding load status for the specific table can be written to the DynamoDB table.

Why other options are not optimal:

A. Use a second Lambda function to invoke the first Lambda function based on Amazon CloudWatch events. While CloudWatch can monitor Redshift, triggering a second Lambda based on metrics or logs might lead to delays or inaccuracies. Directly leveraging the Redshift Data API for events is more accurate and timely. CloudWatch events are better suited for monitoring cluster health than specific table load operations.
C. Use the Amazon Redshift Data API to publish a message to an Amazon Simple Queue Service (Amazon SQS) queue. Configure the SQS queue to invoke the Lambda function. While this approach can also work, it introduces an extra component (SQS) when EventBridge offers a more direct event-driven solution. EventBridge provides richer event filtering and routing capabilities, making it better suited for this scenario. SQS adds complexity without significant benefit.
D. Use a second Lambda function to invoke the first Lambda function based on AWS CloudTrail events. CloudTrail records API calls made to AWS services. It is not the intended mechanism for tracking table load statuses in Redshift. CloudTrail events can be too verbose, and relying on them for this purpose would be less reliable and more difficult to manage. Furthermore, it's less directly related to the event of a table load completing.

In summary:

Option B leverages the strengths of the Redshift Data API for event emission, EventBridge for efficient event routing, and Lambda for serverless processing, providing a scalable, reliable, and manageable solution for updating the DynamoDB table with Redshift load statuses.

Authoritative Links:

Amazon Redshift Data API: https://docs.aws.amazon.com/redshift-data-api/latest/APIReference/Welcome.html
Amazon EventBridge: https://aws.amazon.com/eventbridge/
AWS Lambda: https://aws.amazon.com/lambda/
Amazon DynamoDB: https://aws.amazon.com/dynamodb/

Question: 15 CertyIQ

A data engineer needs to securely transfer 5 TB of data from an on-premises data center to an Amazon S3 bucket. Approximately 5% of the data changes every day. Updates to the data need to be regularly proliferated to the S3 bucket. The data includes files that are in multiple formats. The data engineer needs to automate the transfer process and must schedule the process to run periodically.
Which AWS service should the data engineer use to transfer the data in the MOST operationally efficient way?

A.AWS DataSync
B.AWS Glue
C.AWS Direct Connect
D.Amazon S3 Transfer Acceleration

Answer: A

Explanation:

Here's a detailed justification for why AWS DataSync is the most operationally efficient service for transferring the described dataset to S3, considering the constraints:

AWS DataSync is specifically designed for efficiently and securely transferring large datasets between on-premises storage and AWS storage services like S3. Its incremental transfer capability is key here. Because only 5% of the data changes daily, DataSync can identify and transfer only the modified data, minimizing transfer time and costs. AWS Glue is more suitable for ETL (Extract, Transform, Load) operations, data cataloging, and generating code for data transformations, and isn't optimized for bulk data transfer like this scenario.

AWS Direct Connect establishes a dedicated network connection between your on-premises environment and AWS. While it can improve network performance and security compared to transferring data over the public internet, it doesn't provide the data transfer management features of DataSync. You'd still need a separate tool to handle the actual data transfer. Amazon S3 Transfer Acceleration utilizes the AWS global network to accelerate transfers to S3. However, it is mainly for transferring data over the public internet and does not provide automated scheduling or incremental transfer functionalities. Therefore, DataSync handles the crucial requirements of periodic, automated, incremental data transfer.

DataSync offers built-in scheduling, allowing the engineer to automate the transfer process to run at regular intervals, which satisfies the "schedule the process to run periodically" requirement. It also handles various file formats, allowing easy handling of this variety data. DataSync also handles encryption and integrity verification during transfer, making it a secure solution. The service reduces the operational overhead significantly because it automates most aspects of data transfer.

Therefore, because DataSync automates and schedules the transfer, incrementally transfers updates, and is designed specifically to handle on-premises to cloud migrations, it is the most operationally efficient solution compared to the other options.

Reference Links:

AWS DataSync: https://aws.amazon.com/datasync/
AWS Glue: https://aws.amazon.com/glue/
AWS Direct Connect: https://aws.amazon.com/directconnect/
Amazon S3 Transfer Acceleration: https://aws.amazon.com/s3/transfer-acceleration/

Question: 16 CertyIQ

A company uses an on-premises Microsoft SQL Server database to store financial transaction data. The company migrates the transaction data from the on-premises database to AWS at the end of each month. The company has noticed that the cost to migrate data from the on-premises database to an Amazon RDS for SQL Server database has increased recently.
The company requires a cost-effective solution to migrate the data to AWS. The solution must cause minimal downtown for the applications that access the database.
Which AWS service should the company use to meet these requirements?

A.AWS Lambda
B.AWS Database Migration Service (AWS DMS)
C.AWS Direct Connect
D.AWS DataSync

Answer: B

Explanation:

The correct answer is **B. AWS Database Migration Service (AWS DMS)**.

AWS DMS is specifically designed for database migration, making it the most suitable choice for migrating the on-premises SQL Server database to Amazon RDS for SQL Server. It supports heterogeneous migrations, meaning it can migrate between different database engines. DMS offers continuous data replication, which minimizes downtime as the database is continuously synchronized. It allows for a cutover at a convenient time, resulting in minimal disruption to applications. The continuous replication and the option for a cutover window address the requirement for minimal downtime.

AWS Lambda (A) is a serverless compute service used for running code in response to events. While it could potentially be used for data migration, it would require significantly more custom coding and would be less efficient and more complex to manage than DMS for this specific task.

AWS Direct Connect (C) provides a dedicated network connection from on-premises to AWS. While it can improve network performance and security, it doesn't directly migrate the data. It would only help in faster and more secure data transfer, but the migration tool is still needed, and it doesn't address the downtime requirement. It primarily reduces network costs, not the overall migration cost.

AWS DataSync (D) is primarily used for transferring large datasets between on-premises storage systems and AWS storage services like S3, EFS, and FSx. It's optimized for file-based data transfer, not for migrating database schemas and data directly to a managed database service like RDS.

DMS optimizes costs for ongoing migrations. It charges based on the compute resources used during the migration process, which can be cost-effective compared to manual or custom solutions. Therefore, DMS is the most appropriate and cost-effective option that minimizes downtime.

Further Research:

AWS DMS Documentation: https://aws.amazon.com/dms/
AWS DMS Best Practices: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_BestPractices.html

Question: 17 CertyIQ

A data engineer is building a data pipeline on AWS by using AWS Glue extract, transform, and load (ETL) jobs. The data engineer needs to process data from Amazon RDS and MongoDB, perform transformations, and load the transformed data into Amazon Redshift for analytics. The data updates must occur every hour.
Which combination of tasks will meet these requirements with the LEAST operational overhead? (Choose two.)

A.Configure AWS Glue triggers to run the ETL jobs every hour.
B.Use AWS Glue DataBrew to clean and prepare the data for analytics.
C.Use AWS Lambda functions to schedule and run the ETL jobs every hour.
D.Use AWS Glue connections to establish connectivity between the data sources and Amazon Redshift.
E.Use the Redshift Data API to load transformed data into Amazon Redshift.

Answer: AD

Explanation:

The correct answer is AD. Here's why:

A. Configure AWS Glue triggers to run the ETL jobs every hour: AWS Glue triggers are a native and straightforward way to schedule and execute Glue ETL jobs. They can be configured to run on a schedule (time-based), based on events (like the completion of another job), or on demand. Using Glue triggers directly addresses the requirement for hourly data updates with minimal operational overhead, as it's a managed feature of AWS Glue itself, requiring no additional services or custom code for scheduling. Lambda (option C) would introduce unnecessary complexity for a simple scheduled execution.

D. Use AWS Glue connections to establish connectivity between the data sources and Amazon Redshift: AWS Glue connections provide a centralized and managed way to store and manage connection information to various data sources, including Amazon RDS, MongoDB, and Amazon Redshift. This simplifies the ETL job configuration by allowing you to reference connections instead of hardcoding connection details in each job. This reduces the need to manually configure connections within each ETL script, leading to easier maintenance and reduced operational overhead.

Why other options are not suitable:

B. Use AWS Glue DataBrew to clean and prepare the data for analytics: While DataBrew can be used for data preparation, it's primarily focused on interactive data exploration and visual data transformations, which aren't as suitable for automated, scheduled ETL pipelines as Glue ETL jobs. It does not have the same programmatic flexibility and scaling capabilities as Glue ETL for this use case.

C. Use AWS Lambda functions to schedule and run the ETL jobs every hour: Using Lambda to schedule Glue jobs introduces additional complexity and overhead. You would need to manage the Lambda function, its execution role, and ensure its reliability. Glue triggers provide a more direct and managed approach to scheduling Glue jobs.

E. Use the Redshift Data API to load transformed data into Amazon Redshift: While the Redshift Data API can be used to load data, it's often better suited for executing SQL queries and interacting with Redshift rather than high-volume data loading within an ETL pipeline. Glue ETL jobs, especially with options like dynamicframes.toDF().write.format("redshift"), offer better performance and integration for loading data from other data sources. The Glue connector is also optimized for data loading into Redshift.

Supporting Documentation:

AWS Glue Triggers: https://docs.aws.amazon.com/glue/latest/dg/trigger-definition.html
AWS Glue Connections: https://docs.aws.amazon.com/glue/latest/dg/connections-api.html
AWS Glue DataBrew: https://aws.amazon.com/databrew/
Redshift Data API: https://docs.aws.amazon.com/redshift-data-api/latest/APIReference/Welcome.html

Question: 18 CertyIQ

A company uses an Amazon Redshift cluster that runs on RA3 nodes. The company wants to scale read and write capacity to meet demand. A data engineer needs to identify a solution that will turn on concurrency scaling. Which solution will meet this requirement?

A. Turn on concurrency scaling in workload management (WLM) for Redshift Serverless workgroups.
B. Turn on concurrency scaling at the workload management (WLM) queue level in the Redshift cluster.
C. Turn on concurrency scaling in the settings during the creation of any new Redshift cluster.
D. Turn on concurrency scaling for the daily usage quota for the Redshift cluster.

Answer: B

Explanation:

The correct answer is B: Turn on concurrency scaling at the workload management (WLM) queue level in the Redshift cluster.

Here's a detailed justification:

Amazon Redshift concurrency scaling automatically adds compute capacity to your Redshift cluster to handle increases in concurrent read and write queries. This ensures consistent performance even during peak demand. Concurrency scaling is not a cluster-wide setting enabled during cluster creation (option C) nor is it

directly configured through a daily usage quota (option D). RA3 nodes are specifically designed to utilize concurrency scaling effectively.

Workload Management (WLM) allows you to prioritize and manage queries based on their importance. Concurrency scaling is configured at the WLM queue level. By enabling concurrency scaling for specific WLM queues, you allow Redshift to automatically spin up additional compute resources when queries assigned to that queue experience contention due to high concurrency. This distributes the workload across more resources, improving query performance. Redshift Serverless, mentioned in option A, is a different deployment option than a provisioned Redshift cluster using RA3 nodes. While Redshift Serverless also offers concurrency scaling features, the context specifically refers to an existing Redshift cluster. Therefore, the focus should be on the settings within that cluster.

For more information, refer to the AWS documentation on Amazon Redshift concurrency scaling:

Amazon Redshift Concurrency Scaling
Configuring workload management (WLM) for concurrency scaling

Question: 19 CertyIQ

A data engineer must orchestrate a series of Amazon Athena queries that will run every day. Each query can run for more than 15 minutes.
Which combination of steps will meet these requirements MOST cost-effectively? (Choose two.)

A.Use an AWS Lambda function and the Athena Boto3 client start_query_execution API call to invoke the Athena queries programmatically.

B.Create an AWS Step Functions workflow and add two states. Add the first state before the Lambda function. Configure the second state as a Wait state to periodically check whether the Athena query has finished using the Athena Boto3 get_query_execution API call. Configure the workflow to invoke the next query when the current query has finished running.

C.Use an AWS Glue Python shell job and the Athena Boto3 client start_query_execution API call to invoke the Athena queries programmatically.

D.Use an AWS Glue Python shell script to run a sleep timer that checks every 5 minutes to determine whether the current Athena query has finished running successfully. Configure the Python shell script to invoke the next query when the current query has finished running.

E.Use Amazon Managed Workflows for Apache Airflow (Amazon MWAA) to orchestrate the Athena queries in AWS Batch.

Answer: AB

Explanation:

The most cost-effective solution for orchestrating long-running Athena queries daily involves a combination of AWS Lambda and AWS Step Functions. Lambda provides a serverless execution environment to invoke the Athena queries using the Boto3 start_query_execution API call. This approach is cost-effective because you only pay for the compute time consumed by the function.

AWS Step Functions manages the workflow and handles the asynchronous nature of Athena queries. Adding a Wait state in Step Functions is crucial. This state allows the workflow to pause execution and periodically check the status of the Athena query using the get_query_execution API call. By using a Wait state, the workflow efficiently polls for completion without consuming excessive compute resources, compared to a continuously running process. Once the query completes, the Step Functions workflow triggers the next query in the series.

Let's examine why other options are less suitable. Using AWS Glue Python shell jobs (options C and D) can be more expensive. Glue jobs are generally intended for data transformation and ETL processes, and maintaining a constantly running Glue job to check query status is inefficient and costly. Option E, using Amazon MWAA

and AWS Batch, would introduce significant overhead and cost for this specific use case. MWAA is typically employed for more complex, large-scale data pipelines, making it overkill for a series of Athena queries. Batch is suitable for compute-intensive tasks, not for orchestrating Athena queries.

Therefore, the combination of Lambda for invoking Athena queries and Step Functions with a Wait state for monitoring and orchestration offers the optimal balance of cost and efficiency.

Supporting Links:

AWS Lambda: https://aws.amazon.com/lambda/
AWS Step Functions: https://aws.amazon.com/step-functions/
Amazon Athena: https://aws.amazon.com/athena/
Athena Boto3 Documentation:
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/athena.html

Question: 20 CertyIQ

A company is migrating on-premises workloads to AWS. The company wants to reduce overall operational overhead. The company also wants to explore serverless options.
The company's current workloads use Apache Pig, Apache Oozie, Apache Spark, Apache Hbase, and Apache Flink. The on-premises workloads process petabytes of data in seconds. The company must maintain similar or better performance after the migration to AWS.
Which extract, transform, and load (ETL) service will meet these requirements?

A.AWS Glue
B.Amazon EMR
C.AWS Lambda
D.Amazon Redshift

Answer: B

Explanation:

The correct answer is B (Amazon EMR) because it best fits the requirements of migrating complex workloads utilizing Apache Pig, Oozie, Spark, HBase, and Flink to AWS while aiming for similar or better performance and reduced operational overhead.

Here's why:

Amazon EMR provides a managed Hadoop framework: EMR simplifies the setup, operation, and scaling of big data frameworks like Hadoop, Spark, HBase, and Flink. It directly supports the existing workloads utilizing these technologies. (https://aws.amazon.com/emr/)
Performance: EMR can leverage EC2 instances optimized for compute and memory, allowing for processing petabytes of data in seconds, mirroring the on-premises performance.
Reduced Operational Overhead: EMR handles the underlying infrastructure, operating system patching, and framework updates, freeing the company from these tasks.
Cost Optimization: EMR supports spot instances to reduce costs for fault-tolerant workloads. It also offers various instance types tailored for specific workloads.
Suitable for complex workloads: EMR is designed for running complex, distributed data processing applications.

Now, let's analyze why the other options are less suitable:

AWS Glue: Glue is primarily a serverless ETL service focused on data cataloging, transformation, and loading. While useful, it's not a direct replacement for the diverse processing capabilities of Spark, Flink, and HBase.

Although Glue supports Spark, it might not be as performant or flexible for the company's specific use cases.
AWS Lambda: Lambda is suitable for event-driven, serverless compute tasks, but it is not designed for large-scale data processing with frameworks like Spark or Flink. Its execution time limits and memory constraints make it unsuitable for petabyte-scale workloads.
Amazon Redshift: Redshift is a data warehouse service, ideal for analytical queries and reporting. It is not a direct replacement for the processing frameworks the company currently uses and is more of a destination for processed data rather than an ETL platform in this context. While Redshift can perform some transformations, it's not optimized for the complex operations performed by Spark or Flink.

Therefore, Amazon EMR is the most appropriate ETL service to meet the company's requirements for migrating their on-premises workloads to AWS while maintaining performance and reducing operational overhead, due to its native support for the technologies they are already using at scale.

Question: 21 CertyIQ

A data engineer must use AWS services to ingest a dataset into an Amazon S3 data lake. The data engineer profiles the dataset and discovers that the dataset contains personally identifiable information (PII). The data engineer must implement a solution to profile the dataset and obfuscate the PII.
Which solution will meet this requirement with the LEAST operational effort?

A.Use an Amazon Kinesis Data Firehose delivery stream to process the dataset. Create an AWS Lambda transform function to identify the PII. Use an AWS SDK to obfuscate the PII. Set the S3 data lake as the target for the delivery stream.

B.Use the Detect PII transform in AWS Glue Studio to identify the PII. Obfuscate the PII. Use an AWS Step Functions state machine to orchestrate a data pipeline to ingest the data into the S3 data lake.

C.Use the Detect PII transform in AWS Glue Studio to identify the PII. Create a rule in AWS Glue Data Quality to obfuscate the PII. Use an AWS Step Functions state machine to orchestrate a data pipeline to ingest the data into the S3 data lake.

D.Ingest the dataset into Amazon DynamoDB. Create an AWS Lambda function to identify and obfuscate the PII in the DynamoDB table and to transform the data. Use the same Lambda function to ingest the data into the S3 data lake.

Answer: B

Explanation:

Here's a detailed justification for why option B is the best solution, along with supporting explanations and links:

Option B utilizes the "Detect PII" transform within AWS Glue Studio for identifying and obfuscating Personally Identifiable Information (PII) directly within the data integration process. AWS Glue Studio provides a visual interface to design and run ETL (Extract, Transform, Load) jobs, simplifying the data transformation pipeline. This minimizes operational overhead as it avoids writing custom code for PII detection. AWS Glue's PII detection capabilities use machine learning algorithms, specifically designed to identify sensitive data types, thus reducing the need for complex regex patterns or manual configurations.

An AWS Step Functions state machine is used to orchestrate the overall data pipeline, providing a managed, serverless environment to control the flow of data from source to S3 data lake. This orchestrates the data ingestion process, ensuring that the PII detection and obfuscation occur before the data lands in the S3 data lake. Step Functions provide built-in error handling, retries, and monitoring features, further reducing the operational effort of the data pipeline.

Options A, C, and D are less optimal.

Option A involves Kinesis Data Firehose and Lambda. While Firehose is suitable for real-time streaming, it

might be overkill for batch ingestion. Writing a Lambda function to identify and obfuscate PII adds significant operational overhead compared to using the built-in capabilities of AWS Glue.

Option C utilizes AWS Glue Data Quality rules to obfuscate the data, requiring creation and maintenance of custom rules. While it's feasible, it is more work than leveraging the built-in PII detection features in Glue Studio. Also, the question asks for obfuscation, not just detection.

Option D using DynamoDB as an intermediate store is inefficient and adds unnecessary complexity. DynamoDB is not primarily intended as an ETL staging area for data lake ingestion, and requires managing another database.

In essence, option B provides the LEAST operational effort by leveraging AWS Glue's built-in PII detection and obfuscation and AWS Step Functions to orchestrate the pipeline. It avoids the complexity of managing custom code, using less suitable services (DynamoDB), or using the wrong service for the job (Data Firehose when the job isn't real-time).

Supporting Links:

AWS Glue Studio: https://aws.amazon.com/glue/studio/
AWS Step Functions: https://aws.amazon.com/step-functions/
AWS Glue Data Quality: https://aws.amazon.com/blogs/big-data/validating-data-quality-with-aws-glue-data-quality/

Question: 22 CertyIQ

A company maintains multiple extract, transform, and load (ETL) workflows that ingest data from the company's operational databases into an Amazon S3 based data lake. The ETL workflows use AWS Glue and Amazon EMR to process data.
The company wants to improve the existing architecture to provide automated orchestration and to require minimal manual effort.
Which solution will meet these requirements with the LEAST operational overhead?

A.AWS Glue workflows
B.AWS Step Functions tasks
C.AWS Lambda functions
D.Amazon Managed Workflows for Apache Airflow (Amazon MWAA) workflows

Answer: B

Explanation:

The best answer is **B. AWS Step Functions tasks**. Here's why:

Orchestration: Both AWS Glue workflows and Step Functions can orchestrate ETL tasks. However, Step Functions excels at this due to its visual workflow designer, state management, and error handling capabilities.

Automation: Step Functions allows you to define workflows using state machines that automatically trigger and manage the execution of AWS services like Glue and EMR, reducing manual intervention.

Minimal Operational Overhead: While Glue workflows offer some orchestration, they are primarily focused on Glue jobs. Step Functions is a dedicated orchestration service, specifically designed for complex workflows, making it easier to manage and monitor ETL pipelines.

Lambda: AWS Lambda functions can be part of an ETL process, but managing complex ETL workflows solely with Lambda would result in a highly distributed, difficult-to-manage architecture.

Amazon MWAA: Amazon MWAA is a powerful orchestration tool, but it introduces more operational overhead compared to Step Functions. MWAA requires managing an Apache Airflow environment, including infrastructure, scaling, and maintenance. While powerful, it's overkill for simple to moderately complex ETL orchestration scenarios.

Step Functions offers: retry mechanisms, branching logic, and integration with other AWS services for monitoring and alerting. It's also serverless, meaning you don't have to manage any infrastructure.

Step Functions and Glue: A common pattern is to use Step Functions to orchestrate Glue jobs. Step Functions triggers the Glue jobs, monitors their progress, and handles any errors.

Glue workflows tend to be simpler and more appropriate for orchestrating related Glue jobs, whereas Step Functions is a more versatile and robust solution for orchestrating complex workflows involving different AWS services. Given the need for automated orchestration and minimal manual effort for multiple ETL workflows involving Glue and EMR, Step Functions offers the least operational overhead.

Supporting Links:

AWS Step Functions: Official AWS documentation for Step Functions.
AWS Glue Workflows: Official AWS documentation for Glue workflows.
AWS Whitepaper - Building Data Lakes on AWS: Provides guidance on designing and implementing data lakes, including ETL orchestration.

Question: 23 CertyIQ

A company currently stores all of its data in Amazon S3 by using the S3 Standard storage class.
A data engineer examined data access patterns to identify trends. During the first 6 months, most data files are accessed several times each day. Between 6 months and 2 years, most data files are accessed once or twice each month. After 2 years, data files are accessed only once or twice each year.
The data engineer needs to use an S3 Lifecycle policy to develop new data storage rules. The new storage solution must continue to provide high availability.
Which solution will meet these requirements in the MOST cost-effective way?

A.Transition objects to S3 One Zone-Infrequent Access (S3 One Zone-IA) after 6 months. Transfer objects to S3 Glacier Flexible Retrieval after 2 years.

B.Transition objects to S3 Standard-Infrequent Access (S3 Standard-IA) after 6 months. Transfer objects to S3 Glacier Flexible Retrieval after 2 years.

C.Transition objects to S3 Standard-Infrequent Access (S3 Standard-IA) after 6 months. Transfer objects to S3 Glacier Deep Archive after 2 years.

D.Transition objects to S3 One Zone-Infrequent Access (S3 One Zone-IA) after 6 months. Transfer objects to S3 Glacier Deep Archive after 2 years.

Answer: C

Explanation:

The correct answer is C because it provides the most cost-effective solution while maintaining high availability as defined by the problem constraints. Here's why:

S3 Standard-IA after 6 months: The data is accessed once or twice a month between 6 months and 2 years. S3 Standard-IA is designed for infrequently accessed data but offers rapid access when needed. It's more cost-effective than S3 Standard for this usage pattern while still providing high availability (data stored in multiple Availability Zones).

S3 Glacier Deep Archive after 2 years: After 2 years, the data is accessed only once or twice a year. S3 Glacier Deep Archive is the lowest-cost storage option within S3, ideal for long-term archiving where retrieval

times of up to 12 hours are acceptable.

Why other options are incorrect:

A & D (Using S3 One Zone-IA): S3 One Zone-IA stores data in a single Availability Zone. While cheaper than S3 Standard-IA, it sacrifices availability. If that Availability Zone becomes unavailable, the data is lost. The problem states that high availability must be maintained, so this violates that requirement.

B (Using S3 Glacier Flexible Retrieval): S3 Glacier Flexible Retrieval (formerly S3 Glacier) is suitable for infrequently accessed data with retrieval times ranging from minutes to hours. While it is cheaper than Standard-IA, Glacier Deep Archive offers a lower cost for the given access pattern of once or twice a year. Choosing Glacier Flexible Retrieval over Glacier Deep Archive would thus be less cost-effective.

In summary, option C correctly balances the need for cost optimization with the requirement for high availability by leveraging S3 Standard-IA for the period of monthly access and S3 Glacier Deep Archive for long-term, infrequently accessed data.

Supporting Links:

S3 Storage Classes: https://aws.amazon.com/s3/storage-classes/
S3 Lifecycle Policies: https://docs.aws.amazon.com/AmazonS3/latest/userguide/lifecycle-configuration-examples.html

Question: 24 CertyIQ

A company maintains an Amazon Redshift provisioned cluster that the company uses for extract, transform, and load (ETL) operations to support critical analysis tasks. A sales team within the company maintains a Redshift cluster that the sales team uses for business intelligence (BI) tasks.
The sales team recently requested access to the data that is in the ETL Redshift cluster so the team can perform weekly summary analysis tasks. The sales team needs to join data from the ETL cluster with data that is in the sales team's BI cluster.
The company needs a solution that will share the ETL cluster data with the sales team without interrupting the critical analysis tasks. The solution must minimize usage of the computing resources of the ETL cluster.
Which solution will meet these requirements?

A.Set up the sales team BI cluster as a consumer of the ETL cluster by using Redshift data sharing.

B.Create materialized views based on the sales team's requirements. Grant the sales team direct access to the ETL cluster.

C.Create database views based on the sales team's requirements. Grant the sales team direct access to the ETL cluster.

D.Unload a copy of the data from the ETL cluster to an Amazon S3 bucket every week. Create an Amazon Redshift Spectrum table based on the content of the ETL cluster.

Answer: A

Explanation:

The correct answer is A, using Redshift data sharing. Here's why:

Redshift Data Sharing: This feature allows you to securely share live data across Redshift clusters without data duplication or movement. The sales team's BI cluster can directly query the data residing in the ETL cluster, without impacting the performance of the ETL cluster's operations. This minimizes computing resource usage on the ETL cluster, fulfilling the requirement.

Why other options are not optimal:

B. Materialized Views & Direct Access: While materialized views could provide pre-computed summaries,

they involve data duplication and require refreshing, consuming ETL cluster resources. Granting direct access increases the risk of unintended interference with ETL operations.

C. Database Views & Direct Access: Database views don't materialize data but still place a load on the ETL cluster when the sales team queries them, impacting the ETL cluster's performance. Direct access again poses security and operational risks.

D. Unload to S3 and Redshift Spectrum: This approach involves significant overhead: unloading data to S3 (consuming ETL resources), maintaining an S3 bucket, creating and managing Redshift Spectrum tables, and incurring Spectrum query costs for the sales team. It's a complex and inefficient solution for simple data sharing.

Benefits of Data Sharing:

Zero data duplication: Saves storage costs.
Real-time access: The sales team gets the latest data directly from the ETL cluster.
Security: Granular access control through data sharing policies.
Minimal impact on ETL cluster: Queries are executed against the consumer cluster (sales team's cluster).

In conclusion: Redshift data sharing provides a secure, efficient, and cost-effective way to share data between Redshift clusters without disrupting existing ETL operations, aligning perfectly with the given requirements.

Authoritative Links:

Amazon Redshift Data Sharing
Working with data sharing in Amazon Redshift

Question: 25 CertyIQ

A data engineer needs to join data from multiple sources to perform a one-time analysis job. The data is stored in Amazon DynamoDB, Amazon RDS, Amazon Redshift, and Amazon S3.
Which solution will meet this requirement MOST cost-effectively?

A.Use an Amazon EMR provisioned cluster to read from all sources. Use Apache Spark to join the data and perform the analysis.

B.Copy the data from DynamoDB, Amazon RDS, and Amazon Redshift into Amazon S3. Run Amazon Athena queries directly on the S3 files.

C.Use Amazon Athena Federated Query to join the data from all data sources.

D.Use Redshift Spectrum to query data from DynamoDB, Amazon RDS, and Amazon S3 directly from Redshift.

Answer: C

Explanation:

The most cost-effective solution for a one-time data analysis job involving multiple data sources (DynamoDB, RDS, Redshift, S3) is to use Amazon Athena Federated Query. Athena Federated Query allows querying data residing in different data stores directly without moving or transforming the data.

Option A, using Amazon EMR, is overkill for a one-time analysis. Provisioning and managing an EMR cluster incurs significant costs, even for short durations. While Spark can join data from various sources, the operational overhead and cost of an EMR cluster make it less suitable.

Option B, copying data into S3 and using Athena, requires significant data movement. Extracting data from DynamoDB, RDS, and Redshift incurs egress charges and adds complexity to the process. Moreover, copying the data duplicates storage, increasing storage costs. Athena can then only directly query S3.

Option D, using Redshift Spectrum, is not the best fit. Spectrum is designed primarily for querying data in S3 from Redshift. While Redshift can be federated to some of these services, Athena is generally better suited to perform the analysis because Redshift is typically used for persistent data warehousing and its cost structure is less optimal for one-off queries.

Athena Federated Query is the most economical choice because it eliminates data movement, minimizing network costs and storage duplication. It leverages a pay-per-query pricing model, making it suitable for one-time analysis jobs. Athena connectors are available to directly query DynamoDB, RDS, Redshift, and S3, allowing a consolidated view of data across different systems. This eliminates the need for complex ETL processes and minimizes operational overhead.

In summary, the pay-per-query pricing, ability to query data in place, and ease of use make Athena Federated Query the most cost-effective solution for the scenario.

Relevant Documentation:

Amazon Athena Federated Query
Using Amazon Athena with Amazon DynamoDB
Querying data in relational databases with Amazon Athena Federated Query

Question: 26 CertyIQ

A company is planning to use a provisioned Amazon EMR cluster that runs Apache Spark jobs to perform big data analysis. The company requires high reliability. A big data team must follow best practices for running cost-optimized and long-running workloads on Amazon EMR. The team must find a solution that will maintain the company's current level of performance.
Which combination of resources will meet these requirements MOST cost-effectively? (Choose two.)

A.Use Hadoop Distributed File System (HDFS) as a persistent data store.
B.Use Amazon S3 as a persistent data store.
C.Use x86-based instances for core nodes and task nodes.
D.Use Graviton instances for core nodes and task nodes.
E.Use Spot Instances for all primary nodes.

Answer: BD

Explanation:

Here's a detailed justification for choosing options B and D to meet the company's requirements for a cost-optimized, reliable, and long-running Amazon EMR cluster for big data analysis:

B. Use Amazon S3 as a persistent data store:

HDFS (Option A) is typically used as the default file system within an EMR cluster. However, for long-running, cost-optimized workloads, it's not ideal as it couples data storage to the lifespan of the EMR cluster. If the cluster terminates (intentionally or unintentionally), the data stored in HDFS is lost, leading to data loss and increased costs to reload data every time. Amazon S3 provides a persistent, durable, and cost-effective object storage service that is decoupled from the EMR cluster's lifecycle. The EMR cluster can read and write data directly to S3, and the data persists even if the cluster terminates. This improves reliability and reduces costs in the long run, especially with long-running workloads. This approach aligns with AWS best practices for EMR. [https://docs.aws.amazon.com/emr/latest/best-practices/best-practices-data-storage.html]

D. Use Graviton instances for core nodes and task nodes:

Graviton instances are based on the Arm architecture and offer significant price-performance benefits over x86-based instances (Option C) for many workloads. They are designed by AWS and optimized for cloud

workloads. Using Graviton instances can reduce the cost of running core and task nodes without sacrificing performance. Spark and other big data tools are increasingly well-optimized for Arm architectures. While the exact price-performance benefit will vary depending on the specific workload, Graviton instances generally offer a cost-effective alternative to x86. For EMR specifically, AWS has been promoting the use of Graviton for cost savings. [https://aws.amazon.com/ec2/graviton/]

Why other options are incorrect:

A. Use Hadoop Distributed File System (HDFS) as a persistent data store: Already explained above.
C. Use x86-based instances for core nodes and task nodes: While perfectly viable, it's less cost-effective than option D.
E. Use Spot Instances for all primary nodes: Spot Instances can provide cost savings, but using them for all primary nodes significantly reduces reliability. Primary nodes (master and core) are crucial for the cluster's functionality. If the Spot Instances for these nodes are terminated, the cluster may fail, which directly contradicts the requirement for high reliability. Spot instances are better suited to task nodes.

Question: 27 CertyIQ

A company wants to implement real-time analytics capabilities. The company wants to use Amazon Kinesis Data Streams and Amazon Redshift to ingest and process streaming data at the rate of several gigabytes per second. The company wants to derive near real-time insights by using existing business intelligence (BI) and analytics tools.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use Kinesis Data Streams to stage data in Amazon S3. Use the COPY command to load data from Amazon S3 directly into Amazon Redshift to make the data immediately available for real-time analysis.

B.Access the data from Kinesis Data Streams by using SQL queries. Create materialized views directly on top of the stream. Refresh the materialized views regularly to query the most recent stream data.

C.Create an external schema in Amazon Redshift to map the data from Kinesis Data Streams to an Amazon Redshift object. Create a materialized view to read data from the stream. Set the materialized view to auto refresh.

D.Connect Kinesis Data Streams to Amazon Kinesis Data Firehose. Use Kinesis Data Firehose to stage the data in Amazon S3. Use the COPY command to load the data from Amazon S3 to a table in Amazon Redshift.

Answer: C

Explanation:

The most efficient solution for real-time analytics using Kinesis Data Streams and Redshift, while minimizing operational overhead, is option C. Here's why:

Option C leverages Redshift's capabilities to directly access and query data within Kinesis Data Streams without intermediate staging. The creation of an external schema allows Redshift to treat the Kinesis stream almost like a table. A materialized view built on top of this external schema then provides a continuously updated snapshot of the stream data. Setting the materialized view to auto-refresh ensures that the view reflects the most recent stream data automatically. This approach avoids the need for manual data loading and transformation, reducing operational complexity.

Option A introduces unnecessary complexity by staging data in S3 before loading it into Redshift. This requires managing S3 buckets and configuring the COPY command, increasing operational overhead. Moreover, the "real-time analysis" requirement is not met using batch loading via COPY.

Option B, directly querying Kinesis Data Streams with SQL and materialized views, is not natively supported. Kinesis Data Streams primarily pushes data to consumers; it doesn't inherently support SQL-based queries on the stream itself from a data warehouse like Redshift.

Option D also introduces unnecessary complexity. While Kinesis Data Firehose can deliver data to S3, this introduces an additional service and staging step, adding to operational overhead. The COPY command would still need to be configured and managed.

The key is to leverage Redshift's external tables and materialized views for direct access and near real-time insights with minimal operational effort. Auto-refreshing materialized views simplify the ingestion and transformation process by automatically updating the view as new data becomes available in the Kinesis Data Streams, without needing to implement custom update logic.

Here are resources for further reading:

Amazon Redshift External Tables:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_TABLE.html
Amazon Redshift Materialized Views: https://docs.aws.amazon.com/redshift/latest/dg/materialized-views.html
Kinesis Data Streams: https://aws.amazon.com/kinesis/data-streams/

Question: 28 CertyIQ

A company uses an Amazon QuickSight dashboard to monitor usage of one of the company's applications. The company uses AWS Glue jobs to process data for the dashboard. The company stores the data in a single Amazon S3 bucket. The company adds new data every day.
A data engineer discovers that dashboard queries are becoming slower over time. The data engineer determines that the root cause of the slowing queries is long-running AWS Glue jobs.
Which actions should the data engineer take to improve the performance of the AWS Glue jobs? (Choose two.)

A.Partition the data that is in the S3 bucket. Organize the data by year, month, and day.
B.Increase the AWS Glue instance size by scaling up the worker type.
C.Convert the AWS Glue schema to the DynamicFrame schema class.
D.Adjust AWS Glue job scheduling frequency so the jobs run half as many times each day.
E.Modify the IAM role that grants access to AWS glue to grant access to all S3 features.

Answer: AB

Explanation:

The correct answer is AB. Here's why:

A. Partition the data that is in the S3 bucket. Organize the data by year, month, and day.

Partitioning data in S3 based on date (year, month, day) is a fundamental optimization technique for data lakes. AWS Glue and QuickSight can leverage these partitions to drastically reduce the amount of data scanned during queries and job processing. When Glue jobs or QuickSight dashboards query the data, they can filter based on the partition keys (year, month, day), allowing them to read only the relevant data for a specific time period. Without partitioning, the jobs need to scan the entire dataset to find the data needed, which leads to longer execution times and slower dashboard performance. This concept is aligned with best practices for data lake design and query optimization in cloud environments.https://docs.aws.amazon.com/glue/latest/dg/partitioning.html

B. Increase the AWS Glue instance size by scaling up the worker type.

Increasing the AWS Glue instance size (worker type) provides more computational resources (CPU, memory, disk) to the Glue jobs. This helps accelerate data processing tasks such as data transformation, aggregation, and loading. By increasing the instance size, the Glue jobs can handle larger volumes of data and perform more complex computations in parallel. This directly addresses the problem of long-running Glue jobs and

significantly improves processing time. Scaling worker type will allow Glue jobs to process a larger number of partitions in parallel.https://docs.aws.amazon.com/glue/latest/dg/monitor-performance.html

Why other options are incorrect:

C. Converting to DynamicFrame schema class does not necessarily improve job performance. The benefit of DynamicFrames mainly relates to managing schema evolution and complex data structures but not directly to runtime speed.D. Adjusting job scheduling frequency by reducing the number of times a job is run each day will not improve performance. It reduces resource usage, but the jobs themselves will still be slow.E. Modifying the IAM role to grant access to all S3 features does not improve Glue job performance and violates the principle of least privilege, creating a security risk.

Question: 29 CertyIQ

A data engineer needs to use AWS Step Functions to design an orchestration workflow. The workflow must parallel process a large collection of data files and apply a specific transformation to each file.
Which Step Functions state should the data engineer use to meet these requirements?

A.Parallel state
B.Choice state
C.Map state
D.Wait state

Answer: C

Explanation:

The correct answer is C, the Map state. The Map state in AWS Step Functions is specifically designed for dynamic parallelism, enabling you to iterate over a collection of items (like data files) and execute the same steps for each item concurrently. This perfectly aligns with the requirement to process a large collection of data files and apply the same transformation to each. The Map state receives an array as input and then iterates over each element, executing a set of substates (defined within the Map state) for each element. This parallelism significantly reduces the overall processing time compared to sequential processing.

A Parallel state (option A) also executes multiple branches of execution in parallel, but it requires you to predefine those branches. It's not designed for iterating over a dynamically sized collection like the Map state. A Choice state (option B) adds conditional branching based on the input, which isn't relevant to processing each file in parallel. A Wait state (option D) simply delays the execution of the workflow for a specified duration, which doesn't contribute to parallel processing. Therefore, the Map state is the only option that directly facilitates parallel processing of data files based on iterating through a collection, fulfilling the stated requirements.

For further reading, refer to the AWS Step Functions documentation on Map state: https://docs.aws.amazon.com/step-functions/latest/dg/amazon-states-language-map-state.html and an overview of state types: https://docs.aws.amazon.com/step-functions/latest/dg/concepts-states.html.

Question: 30 CertyIQ

A company is migrating a legacy application to an Amazon S3 based data lake. A data engineer reviewed data that is associated with the legacy application. The data engineer found that the legacy data contained some duplicate information.
The data engineer must identify and remove duplicate information from the legacy application data.
Which solution will meet these requirements with the LEAST operational overhead?

A.Write a custom extract, transform, and load (ETL) job in Python. Use the DataFrame.drop_duplicates() function by importing the Pandas library to perform data deduplication.

B.Write an AWS Glue extract, transform, and load (ETL) job. Use the FindMatches machine learning (ML) transform to transform the data to perform data deduplication.

C.Write a custom extract, transform, and load (ETL) job in Python. Import the Python dedupe library. Use the dedupe library to perform data deduplication.

D.Write an AWS Glue extract, transform, and load (ETL) job. Import the Python dedupe library. Use the dedupe library to perform data deduplication.

Answer: B

Explanation:

The most efficient solution for identifying and removing duplicate information from legacy data during migration to an Amazon S3 data lake, with the least operational overhead, is to utilize an AWS Glue ETL job with the FindMatches ML Transform.

Option B is superior because AWS Glue is a fully managed ETL service, reducing the operational burden associated with managing infrastructure and scaling resources. The FindMatches transform within Glue specifically leverages machine learning to identify near-duplicate records, even if they aren't exact matches, which is a common scenario in legacy data. This automated, ML-driven approach avoids the need for writing complex custom deduplication logic.

Options A and C involve writing custom Python ETL jobs. While these are viable, they require significantly more development and maintenance effort compared to using a pre-built Glue transform. Pandas' drop_duplicates() function (Option A) only removes exact duplicates, which might not be sufficient for the legacy data if there are inconsistencies. The dedupe library (Option C and D) is powerful but needs to be integrated and managed, adding to the overhead.

Option D, while using the dedupe library, is less efficient than using the FindMatches transform because it requires more configuration and fine-tuning, as well as integration within Glue. FindMatches is specifically designed to address this type of deduplication problem within AWS Glue, offering a simpler and more optimized solution.

In summary, AWS Glue's FindMatches transform is the ideal solution due to its managed nature, specialized functionality for fuzzy matching, and reduced operational overhead. It leverages ML for more accurate duplicate detection and simplifies the ETL process.

Relevant AWS Documentation:

AWS Glue FindMatches: https://docs.aws.amazon.com/glue/latest/dg/find-matches.html
AWS Glue: https://aws.amazon.com/glue/

Question: 31 CertyIQ

A company is building an analytics solution. The solution uses Amazon S3 for data lake storage and Amazon Redshift for a data warehouse. The company wants to use Amazon Redshift Spectrum to query the data that is in Amazon S3.
Which actions will provide the FASTEST queries? (Choose two.)

A.Use gzip compression to compress individual files to sizes that are between 1 GB and 5 GB.

B.Use a columnar storage file format.

C.Partition the data based on the most common query predicates.

D.Split the data into files that are less than 10 KB.

E.Use file formats that are not splittable.

Answer: BC

Explanation:

The correct answer is **BC**. Here's a detailed justification:

B. Use a columnar storage file format: Columnar storage formats like Parquet or ORC are highly optimized for analytical queries. Instead of storing data row by row (like CSV or JSON), these formats store data column by column. This is beneficial because Redshift Spectrum only needs to read the columns relevant to the query, significantly reducing I/O and improving query performance. When only specific columns are selected, only those columns are scanned from S3, making queries faster.

C. Partition the data based on the most common query predicates: Partitioning data in S3 involves organizing files into folders based on the values of one or more columns, such as date or region. When a query includes a WHERE clause that filters on one of these partitioned columns, Redshift Spectrum can use partition pruning. This means Spectrum only scans the folders (partitions) that contain the relevant data, skipping the rest, dramatically reducing the amount of data processed and improving query speed. Partitioning is one of the most effective optimizations for data lakes.

Why the other options are less optimal or incorrect:

A. Use gzip compression to compress individual files to sizes that are between 1 GB and 5 GB: While compression is generally good for reducing storage costs and network transfer times, specifically targeting 1-5 GB for gzip is not the primary factor for fastest Spectrum queries. Other compression algorithms such as Snappy or Zstandard can offer similar compression ratios with better performance for Spectrum. While compression is important, columnar format and partitioning have larger impacts.

D. Split the data into files that are less than 10 KB: Having many small files can lead to increased overhead due to the number of S3 requests required to read the data. This overhead can negatively impact query performance. Redshift Spectrum performs best with larger files.

E. Use file formats that are not splittable: Splittable file formats allow Redshift Spectrum to parallelize the reading of data, utilizing multiple nodes to process the data simultaneously. Non-splittable file formats prevent this parallelism, limiting the query's potential speed.

In Summary:

Columnar storage minimizes data I/O by only reading relevant columns, and partitioning enables Spectrum to skip irrelevant data partitions entirely. These two optimizations work synergistically to significantly improve query performance in Redshift Spectrum when querying data in S3.

Authoritative Links:

Amazon Redshift Spectrum Best Practices: https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift-spectrum/
Optimizing Amazon S3 Performance: https://docs.aws.amazon.com/AmazonS3/latest/userguide/optimizing-performance.html

Question: 32 CertyIQ

A company uses Amazon RDS to store transactional data. The company runs an RDS DB instance in a private subnet. A developer wrote an AWS Lambda function with default settings to insert, update, or delete data in the DB instance.
The developer needs to give the Lambda function the ability to connect to the DB instance privately without using the public internet.
Which combination of steps will meet this requirement with the LEAST operational overhead? (Choose two.)

A.Turn on the public access setting for the DB instance.

B.Update the security group of the DB instance to allow only Lambda function invocations on the database port.

C.Configure the Lambda function to run in the same subnet that the DB instance uses.

D.Attach the same security group to the Lambda function and the DB instance. Include a self-referencing rule that allows access through the database port.

E.Update the network ACL of the private subnet to include a self-referencing rule that allows access through the database port.

Answer: CD

Explanation:

Here's a detailed justification for why options C and D are the correct choices to enable a Lambda function to privately connect to an RDS DB instance in the same VPC, with minimal operational overhead:

Option C: Configure the Lambda function to run in the same subnet that the DB instance uses.

This is crucial for private connectivity. By placing the Lambda function within the same private subnet as the RDS instance, the Lambda function can access the RDS instance using its private IP address, eliminating the need for public internet access. AWS Lambda inherently allows you to configure VPC access. This is a best practice when the function needs to interact with resources inside your VPC, such as databases. Using the VPC means the Lambda function operates within the private network you defined, enhancing security and potentially reducing latency. Subnets must have enough available IP addresses for the Lambda function to scale.

https://docs.aws.amazon.com/lambda/latest/dg/configuration-vpc.html

Option D: Attach the same security group to the Lambda function and the DB instance. Include a self-referencing rule that allows access through the database port.

Security groups act as virtual firewalls, controlling traffic in and out of your resources. Attaching the same security group to both the Lambda function and the RDS instance simplifies security management. The self-referencing rule (allowing traffic within the security group) enables the Lambda function to initiate a connection to the RDS instance on the database port (e.g., 3306 for MySQL, 5432 for PostgreSQL). A self-referencing rule is less restrictive than opening up access to all traffic on the port from any source. Using a single security group is more maintainable than managing separate rules for each resource.

https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

Why other options are incorrect:

A. Turn on the public access setting for the DB instance: This defeats the purpose of private connectivity and exposes the database to the internet, increasing security risks.
B. Update the security group of the DB instance to allow only Lambda function invocations on the database port: While not inherently wrong, this is less efficient than using the same security group and a self-referencing rule as it requires specific knowledge and management of the Lambda function's network interface. Also, how would the security group identify traffic originating specifically from Lambda function invocations without tying the resources together?
E. Update the network ACL of the private subnet to include a self-referencing rule that allows access through the database port: While technically functional, Network ACLs (NACLs) are stateless and operate at the subnet level. Security groups are stateful and operate at the instance level providing a more granular and resource-specific control and are easier to manage in this scenario. You'd also have to manage inbound and outbound rules to make sure the packets can come back to the function.

In summary, by placing the Lambda function in the same subnet as the RDS instance and using a shared security group with a self-referencing rule, you achieve secure and private connectivity with the least amount

of administrative overhead.

Question: 33 CertyIQ

A company has a frontend ReactJS website that uses Amazon API Gateway to invoke REST APIs. The APIs perform the functionality of the website. A data engineer needs to write a Python script that can be occasionally invoked through API Gateway. The code must return results to API Gateway.
Which solution will meet these requirements with the LEAST operational overhead?

A.Deploy a custom Python script on an Amazon Elastic Container Service (Amazon ECS) cluster.

B.Create an AWS Lambda Python function with provisioned concurrency.

C.Deploy a custom Python script that can integrate with API Gateway on Amazon Elastic Kubernetes Service (Amazon EKS).

D.Create an AWS Lambda function. Ensure that the function is warm by scheduling an Amazon EventBridge rule to invoke the Lambda function every 5 minutes by using mock events.

Answer: B

Explanation:

The most suitable solution with the least operational overhead is **B. Create an AWS Lambda Python function with provisioned concurrency.**

Here's why:

AWS Lambda is designed for event-driven, serverless execution. It's perfect for running Python scripts invoked via API Gateway without managing servers or containers. This significantly reduces operational overhead compared to managing ECS or EKS clusters. (Source: https://aws.amazon.com/lambda/)
Provisioned Concurrency: The requirement states the script is occasionally invoked. By using provisioned concurrency, you ensure that Lambda function instances are pre-initialized and ready to respond to requests, minimizing cold starts and improving latency. (Source: https://aws.amazon.com/lambda/provisioned-concurrency/)
Direct API Gateway Integration: Lambda has a direct integration with API Gateway, making it simple to invoke Lambda functions as backends for API endpoints. This streamlined integration simplifies the overall architecture. (Source: https://docs.aws.amazon.com/apigateway/latest/developerguide/services-lambda-integration.html)
Option A (ECS) and C (EKS) are overkill: ECS and EKS involve managing container orchestration, which introduces significant operational complexity and cost for a simple occasional script execution. These options are more appropriate for complex applications that require finer-grained control over the execution environment.
Option D (Lambda with EventBridge warming) is less efficient: While keeping a Lambda function "warm" through scheduled invocations can reduce cold starts, it is not as efficient or cost-effective as using provisioned concurrency. EventBridge invocations consume resources even when the script is not actively needed. Provisioned concurrency dedicates resources and keeps them ready without unnecessary triggers.
Cost Efficiency: Lambda's pay-per-execution model makes it cost-effective for occasional script executions. ECS/EKS require running infrastructure continuously, even when the script is idle. Using Provisioned Concurrency does incur a cost, but it is targetted, manageable, and more efficient than constant EventBridge triggers if minimizing latency is important.

In summary, Lambda with provisioned concurrency provides the best balance of performance, simplicity, and cost-effectiveness for this specific use case. It eliminates the operational burden of managing containers while ensuring responsive execution through pre-initialized function instances.

Question: 34 CertyIQ

A company has a production AWS account that runs company workloads. The company's security team created a security AWS account to store and analyze security logs from the production AWS account. The security logs in the production AWS account are stored in Amazon CloudWatch Logs.
The company needs to use Amazon Kinesis Data Streams to deliver the security logs to the security AWS account.
Which solution will meet these requirements?

A.Create a destination data stream in the production AWS account. In the security AWS account, create an IAM role that has cross-account permissions to Kinesis Data Streams in the production AWS account.

B.Create a destination data stream in the security AWS account. Create an IAM role and a trust policy to grant CloudWatch Logs the permission to put data into the stream. Create a subscription filter in the security AWS account.

C.Create a destination data stream in the production AWS account. In the production AWS account, create an IAM role that has cross-account permissions to Kinesis Data Streams in the security AWS account.

D.Create a destination data stream in the security AWS account. Create an IAM role and a trust policy to grant CloudWatch Logs the permission to put data into the stream. Create a subscription filter in the production AWS account.

Answer: D

Explanation:

Here's a detailed justification for why option D is the correct solution for streaming CloudWatch Logs from a production AWS account to Kinesis Data Streams in a security AWS account, along with supporting concepts and links:

The core requirement is to get security logs from CloudWatch Logs (in the production account) into Kinesis Data Streams (in the security account). CloudWatch Logs subscription filters are the primary mechanism for streaming log data to other AWS services. For cross-account delivery, a specific configuration is necessary involving IAM roles and trust policies.

Option D correctly places the Kinesis Data Stream in the security account, which aligns with the goal of storing and analyzing security logs in that account. It also correctly identifies the need for an IAM role in the security account that CloudWatch Logs assumes. The IAM role's trust policy is crucial; it explicitly grants CloudWatch Logs (running in the production account) permission to assume the role. This trust relationship is fundamental for cross-account access. The subscription filter is created in the production account, where the logs originate. This filter, when configured correctly, will invoke the IAM role and deliver the logs to the Kinesis Data Stream in the security account.

Option A is incorrect because the destination data stream should reside in the security account, not the production account.

Option B is incorrect because the subscription filter must be created in the production account to access the CloudWatch logs in the production account.

Option C is incorrect because the IAM role needs to be created in the security account and assumed by the CloudWatch logs service in the production account.

In summary, option D sets up the necessary cross-account IAM permissions and configures the CloudWatch Logs subscription filter correctly to achieve the desired data flow.

Relevant links for further research:

CloudWatch Logs Subscription Filters:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html
Cross-Account Access with IAM Roles: https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-

account-with-roles.html
Kinesis Data Streams: https://aws.amazon.com/kinesis/data-streams/

Question: 35 CertyIQ

A company uses Amazon S3 to store semi-structured data in a transactional data lake. Some of the data files are small, but other data files are tens of terabytes.
A data engineer must perform a change data capture (CDC) operation to identify changed data from the data source. The data source sends a full snapshot as a JSON file every day and ingests the changed data into the data lake.
Which solution will capture the changed data MOST cost-effectively?

A.Create an AWS Lambda function to identify the changes between the previous data and the current data. Configure the Lambda function to ingest the changes into the data lake.

B.Ingest the data into Amazon RDS for MySQL. Use AWS Database Migration Service (AWS DMS) to write the changed data to the data lake.

C.Use an open source data lake format to merge the data source with the S3 data lake to insert the new data and update the existing data.

D.Ingest the data into an Amazon Aurora MySQL DB instance that runs Aurora Serverless. Use AWS Database Migration Service (AWS DMS) to write the changed data to the data lake.

Answer: C

Explanation:

The correct answer is C because it offers the most cost-effective and efficient solution for CDC in a data lake environment compared to the other options.

Option A is not ideal. While Lambda can compare data, processing large (tens of terabytes) JSON files daily using Lambda would be computationally expensive and likely exceed Lambda's execution time limits. Moreover, managing state and handling potential errors during a full scan comparison becomes complex.

Options B and D are less efficient and unnecessarily involve relational databases. Ingesting full snapshots into RDS or Aurora solely for CDC introduces significant overhead. Setting up and maintaining database instances and DMS adds to the operational complexity and cost. Furthermore, the JSON format isn't naturally suited to relational database structures, requiring transformations that further increase processing time and cost. DMS, while good for database migrations, is overkill for this CDC use case where the source is simply a snapshot file.

Option C leverages open-source data lake formats like Apache Iceberg, Delta Lake, or Apache Hudi. These formats provide built-in support for ACID transactions, schema evolution, and efficient merging of data. They allow for direct processing of data in S3 without the need for intermediate databases. The merge operation updates existing data and inserts new data based on keys, efficiently identifying and applying changes from the daily snapshots. This is the most scalable and cost-effective approach since the data lake format handles the CDC logic directly within the storage layer, using S3 as the processing engine. Services like AWS Glue (with Spark) can be used to perform these merge operations, optimized for data lake workloads. This solution eliminates the need for a database for transient data storage, reducing cost and complexity.

Further Research:

Apache Iceberg: https://iceberg.apache.org/
Delta Lake: https://delta.io/
Apache Hudi: https://hudi.apache.org/
AWS Glue: https://aws.amazon.com/glue/

Question: 36 CertyIQ

A data engineer runs Amazon Athena queries on data that is in an Amazon S3 bucket. The Athena queries use AWS Glue Data Catalog as a metadata table.
The data engineer notices that the Athena query plans are experiencing a performance bottleneck. The data engineer determines that the cause of the performance bottleneck is the large number of partitions that are in the S3 bucket. The data engineer must resolve the performance bottleneck and reduce Athena query planning time.
Which solutions will meet these requirements? (Choose two.)

A.Create an AWS Glue partition index. Enable partition filtering.

B.Bucket the data based on a column that the data have in common in a WHERE clause of the user query.

C.Use Athena partition projection based on the S3 bucket prefix.

D.Transform the data that is in the S3 bucket to Apache Parquet format.

E.Use the Amazon EMR S3DistCP utility to combine smaller objects in the S3 bucket into larger objects.

Answer: AC

Explanation:

The problem is slow Athena query planning time due to a large number of partitions in S3 and the corresponding metadata in the Glue Data Catalog. This significantly increases the time Athena takes to identify and select the relevant partitions for a query.

Option A: Create an AWS Glue partition index. Enable partition filtering. is a correct solution. Glue partition indexes are designed to speed up partition discovery in Athena. By creating an index, Athena can quickly locate the partitions that match the query's filter criteria, drastically reducing planning time. Partition filtering helps to apply the filter conditions early in the query planning process, further minimizing the number of partitions that Athena needs to consider. https://docs.aws.amazon.com/glue/latest/dg/partition-indexes.html

Option C: Use Athena partition projection based on the S3 bucket prefix. is also a valid solution. Partition projection allows Athena to infer partition values directly from the S3 bucket structure, eliminating the need to store partition metadata in the Glue Data Catalog. If the partitions are organized in S3 in a predictable manner (e.g., s3://bucket/year=2023/month=12/), Athena can dynamically determine the partitions based on the bucket paths. This avoids reading a potentially large partition list from the Glue Data Catalog, significantly improving planning time. https://docs.aws.amazon.com/athena/latest/ug/partition-projection.html

Option B is relevant to query performance, but doesn't directly address query planning time. Bucketing improves data locality and helps Athena read only the necessary data during query execution, not during planning.

Option D, while beneficial for query performance due to Parquet's columnar storage and compression, doesn't solve the partition discovery problem directly. It addresses data retrieval efficiency, not the initial planning bottleneck caused by the large number of partitions.

Option E focuses on optimizing the size of individual S3 objects, which might indirectly improve performance by reducing the number of files Athena needs to access. However, it does not directly reduce the number of partitions or the overhead associated with Glue Data Catalog lookups during query planning. It's more about optimizing data storage rather than metadata management.

Question: 37 CertyIQ

A data engineer must manage the ingestion of real-time streaming data into AWS. The data engineer wants to perform real-time analytics on the incoming streaming data by using time-based aggregations over a window of up to 30 minutes. The data engineer needs a solution that is highly fault tolerant.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use an AWS Lambda function that includes both the business and the analytics logic to perform time-based aggregations over a window of up to 30 minutes for the data in Amazon Kinesis Data Streams.

B.Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to analyze the data that might occasionally contain duplicates by using multiple types of aggregations.

C.Use an AWS Lambda function that includes both the business and the analytics logic to perform aggregations for a tumbling window of up to 30 minutes, based on the event timestamp.

D.Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to analyze the data by using multiple types of aggregations to perform time-based analytics over a window of up to 30 minutes.

Answer: D

Explanation:

The correct answer is D because Amazon Managed Service for Apache Flink is specifically designed for real-time analytics on streaming data with minimal operational overhead. It provides built-in support for time-based windowing aggregations, allowing the data engineer to perform analytics over a 30-minute window. Flink handles fault tolerance automatically through checkpointing and state management, ensuring data consistency and availability even in the event of failures.

Option A and C involve using AWS Lambda, which while capable of processing streaming data, would require the data engineer to implement and manage the time-based aggregation logic and fault tolerance mechanisms manually. This significantly increases operational overhead. Lambda functions are also typically better suited for shorter processing times and might not be as efficient or cost-effective for continuous, long-running aggregations over 30-minute windows. Furthermore, Lambda doesn't inherently provide the fault tolerance required without additional complex configuration.

Option B mentions the possibility of duplicate data. Flink handles duplicates through exactly-once processing capabilities, ensuring accurate aggregation results. While the question doesn't explicitly state there are duplicates, choosing Flink provides robustness.

In summary, Flink's native support for windowed aggregations, fault tolerance, and exactly-once processing makes it the most suitable solution for real-time analytics on streaming data with minimal operational overhead. It simplifies the development and deployment process, allowing the data engineer to focus on the analytics logic rather than infrastructure management.

Relevant documentation:

Amazon Managed Service for Apache Flink: https://aws.amazon.com/flink/
Flink Windowing: https://nightlies.apache.org/flink/flink-docs-stable/docs/dev/datastream/operators/windows/
Flink Fault Tolerance: https://nightlies.apache.org/flink/flink-docs-stable/docs/learn-flink/fault_tolerance/

Question: 38 CertyIQ

A company is planning to upgrade its Amazon Elastic Block Store (Amazon EBS) General Purpose SSD storage from gp2 to gp3. The company wants to prevent any interruptions in its Amazon EC2 instances that will cause data loss during the migration to the upgraded storage.
Which solution will meet these requirements with the LEAST operational overhead?

A.Create snapshots of the gp2 volumes. Create new gp3 volumes from the snapshots. Attach the new gp3 volumes to the EC2 instances.

B.Create new gp3 volumes. Gradually transfer the data to the new gp3 volumes. When the transfer is complete, mount the new gp3 volumes to the EC2 instances to replace the gp2 volumes.

C.Change the volume type of the existing gp2 volumes to gp3. Enter new values for volume size, IOPS, and throughput.

D.Use AWS DataSync to create new gp3 volumes. Transfer the data from the original gp2 volumes to the new gp3 volumes.

Answer: C

Explanation:

Here's a detailed justification for why option C is the best solution for upgrading from gp2 to gp3 EBS volumes with minimal downtime and operational overhead:

The core requirement is to upgrade from gp2 to gp3 without interrupting the EC2 instances and causing data loss, all while minimizing operational overhead.

Option C, "Change the volume type of the existing gp2 volumes to gp3. Enter new values for volume size, IOPS, and throughput," is the most efficient approach. EBS volume type modification is an in-place upgrade. You directly modify the existing gp2 volume to become a gp3 volume. This avoids the need to create new volumes, copy data, and remount them, which inherently introduce downtime.

AWS EBS supports modifying the volume type on the fly without detaching the volume or stopping the instance. You can change the volume type, size, IOPS, and throughput using the AWS Management Console, AWS CLI, or AWS SDKs.

Options A, B, and D all involve creating new volumes and transferring data. Option A uses snapshots which will necessitate stopping the instance to ensure data consistency. Option B, gradually transfer data means application-level or OS level data sync tool is needed, which adds complexity and potential risk. Option D introduces AWS DataSync, which is powerful but overkill for a simple EBS volume upgrade within the same availability zone. DataSync is suitable when migrating data across regions or between on-premises and AWS.

Changing the volume type in place (Option C) is significantly faster and less disruptive than creating new volumes and copying data. Furthermore, it requires the least amount of manual intervention and monitoring. This makes it the solution with the LEAST operational overhead while satisfying the critical requirement of preventing data loss and minimizing interruptions.

Therefore, option C is the best solution because it directly addresses the requirement with the lowest operational overhead and minimal downtime.

Relevant AWS documentation:

Modifying EBS Volumes: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-modify-volume.html
EBS Volume Types: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html

Question: 39 CertyIQ

A company is migrating its database servers from Amazon EC2 instances that run Microsoft SQL Server to Amazon RDS for Microsoft SQL Server DB instances. The company's analytics team must export large data elements every day until the migration is complete. The data elements are the result of SQL joins across multiple tables. The data must be in Apache Parquet format. The analytics team must store the data in Amazon S3.
Which solution will meet these requirements in the MOST operationally efficient way?

A.Create a view in the EC2 instance-based SQL Server databases that contains the required data elements. Create an AWS Glue job that selects the data directly from the view and transfers the data in Parquet format to an S3 bucket. Schedule the AWS Glue job to run every day.

B.Schedule SQL Server Agent to run a daily SQL query that selects the desired data elements from the EC2 instance-based SQL Server databases. Configure the query to direct the output .csv objects to an S3 bucket. Create an S3 event that invokes an AWS Lambda function to transform the output format from .csv to Parquet.

C.Use a SQL query to create a view in the EC2 instance-based SQL Server databases that contains the required data elements. Create and run an AWS Glue crawler to read the view. Create an AWS Glue job that retrieves the

data and transfers the data in Parquet format to an S3 bucket. Schedule the AWS Glue job to run every day.

D.Create an AWS Lambda function that queries the EC2 instance-based databases by using Java Database Connectivity (JDBC). Configure the Lambda function to retrieve the required data, transform the data into Parquet format, and transfer the data into an S3 bucket. Use Amazon EventBridge to schedule the Lambda function to run every day.

Answer: C

Explanation:

The best solution for exporting large SQL Server data elements, transforming them to Parquet format, and storing them in S3 with operational efficiency is option C.

Here's why:

AWS Glue is purpose-built for ETL: Glue is specifically designed for Extract, Transform, and Load (ETL) operations. It simplifies the process of data extraction from various sources, transformation, and loading into data stores like S3.

Views simplify data selection: Creating a view in the SQL Server databases allows the Glue job to query a simplified, pre-defined dataset containing the joined data elements, rather than complex SQL joins within the Glue job itself. This improves maintainability and reduces the load on the SQL Server databases.

Glue Crawler for Schema Discovery: A Glue crawler can automatically infer the schema of the view. This reduces the manual effort involved in defining the data schema within the Glue job.

Parquet Conversion in Glue: Glue has built-in support for converting data into Parquet format, which is a columnar storage format optimized for analytical workloads.

Scheduled Glue Jobs for Automation: Scheduling the Glue job automates the daily export and transformation process.

Here's why other options are less optimal:

Option A: While similar, it skips the crawler and assumes you know the schema. In a dynamic environment, a crawler is best practice to ensure schema changes are reflected automatically.

Option B: Using SQL Server Agent to export to CSV and then transforming it with Lambda is less efficient and more complex than using Glue for the entire process. CSV is not an efficient format for large data volumes. Transforming CSV to Parquet via Lambda adds operational overhead.

Option D: Using Lambda with JDBC for large data transfers from SQL Server is not recommended due to potential performance bottlenecks and scaling limitations of Lambda functions. It also increases complexity compared to using AWS Glue. Also Lambda might timeout since its execution is limited to 15 minutes.

Therefore, Option C provides the most operationally efficient and scalable approach for the given requirements by leveraging the capabilities of AWS Glue for ETL and Parquet conversion, with the added benefit of automated schema discovery using Glue crawlers.

Relevant Links:

AWS Glue Documentation
Apache Parquet
AWS Glue Crawlers

Question: 40 CertyIQ

A data engineering team is using an Amazon Redshift data warehouse for operational reporting. The team wants to prevent performance issues that might result from long- running queries. A data engineer must choose a system table in Amazon Redshift to record anomalies when a query optimizer identifies conditions that might indicate performance issues.
Which table views should the data engineer use to meet this requirement?

A.STL_USAGE_CONTROL

B.STL_ALERT_EVENT_LOG

C.STL_QUERY_METRICS

D.STL_PLAN_INFO

Answer: B

Explanation:

The correct answer is B: STL_ALERT_EVENT_LOG. This system table is specifically designed to capture alerts generated by the Amazon Redshift query optimizer when it detects conditions that could lead to performance problems. These alerts flag potential issues, such as missing statistics, suboptimal join orders, or the use of uncompressed data. By monitoring STL_ALERT_EVENT_LOG, the data engineering team can proactively identify and address these issues, preventing long-running queries and maintaining the overall performance of the Amazon Redshift data warehouse.

STL_USAGE_CONTROL (option A) is used for managing concurrency scaling usage and is not related to query performance anomalies. STL_QUERY_METRICS (option C) provides detailed performance metrics about executed queries but doesn't specifically highlight potential issues identified by the optimizer before or during query execution. STL_PLAN_INFO (option D) contains information about the query plan, but it does not directly provide alerts or anomaly detection information based on the optimizer's assessment. Therefore, STL_ALERT_EVENT_LOG is the most appropriate table for identifying query optimizer alerts that indicate potential performance issues in Amazon Redshift.

Amazon Redshift System Tables Reference

Question: 41 CertyIQ

A data engineer must ingest a source of structured data that is in .csv format into an Amazon S3 data lake. The .csv files contain 15 columns. Data analysts need to run Amazon Athena queries on one or two columns of the dataset. The data analysts rarely query the entire file.
Which solution will meet these requirements MOST cost-effectively?

A.Use an AWS Glue PySpark job to ingest the source data into the data lake in .csv format.

B.Create an AWS Glue extract, transform, and load (ETL) job to read from the .csv structured data source. Configure the job to ingest the data into the data lake in JSON format.

C.Use an AWS Glue PySpark job to ingest the source data into the data lake in Apache Avro format.

D.Create an AWS Glue extract, transform, and load (ETL) job to read from the .csv structured data source. Configure the job to write the data into the data lake in Apache Parquet format.

Answer: D

Explanation:

The most cost-effective solution is to use AWS Glue to transform the CSV data into Apache Parquet format before storing it in the S3 data lake.

Here's why:

Columnar Storage: Parquet is a columnar storage format. This means that data for each column is stored contiguously on disk. When Athena queries only one or two columns, it only needs to read those specific columns, drastically reducing the amount of data scanned and, consequently, query costs. This contrasts with row-based formats like CSV or JSON, where the entire row must be read even if only a few columns are needed.
Athena Cost Optimization: Athena charges based on the amount of data scanned. By reducing the data scanned with Parquet, query costs are significantly lowered.
Data Compression: Parquet supports efficient data compression. This further reduces storage costs in S3 and decreases the amount of data that Athena needs to process, leading to faster query performance and lower costs.
Glue ETL Capabilities: AWS Glue is a fully managed ETL service that can read CSV data, transform it, and write it to S3 in Parquet format. Glue provides the ability to define schema and handle data type conversions easily.
Avro: While Avro is a row-based format and supports schema evolution, it does not offer the same cost benefits as Parquet for analytical queries that only need a subset of columns.
CSV and JSON: Storing the data as CSV or JSON would be the least cost-effective option as Athena would have to scan the entire file for each query, regardless of the number of columns needed. This significantly increases query costs.

Therefore, using Glue to convert the CSV data to Parquet optimizes both storage and Athena query costs by leveraging columnar storage and compression, directly addressing the requirement of cost-effectiveness when analysts frequently query only a few columns.

Here are some authoritative links for further research:

Apache Parquet: Official Apache Parquet website.
Amazon Athena Pricing: Details on how Athena is priced.
AWS Glue: AWS Glue product page.
Top 5 Performance Tuning Tips for Amazon Athena: AWS blog on Athena performance tuning.

Question: 42 CertyIQ

A company has five offices in different AWS Regions. Each office has its own human resources (HR) department that uses a unique IAM role. The company stores employee records in a data lake that is based on Amazon S3 storage.
A data engineering team needs to limit access to the records. Each HR department should be able to access records for only employees who are within the HR department's Region.
Which combination of steps should the data engineering team take to meet this requirement with the LEAST operational overhead? (Choose two.)

A.Use data filters for each Region to register the S3 paths as data locations.

B.Register the S3 path as an AWS Lake Formation location.

C.Modify the IAM roles of the HR departments to add a data filter for each department's Region.

D.Enable fine-grained access control in AWS Lake Formation. Add a data filter for each Region.

E.Create a separate S3 bucket for each Region. Configure an IAM policy to allow S3 access. Restrict access based on Region.

Answer: BD

Explanation:

The best approach for implementing regional access control on the employee records in the data lake while minimizing operational overhead is using AWS Lake Formation with fine-grained access control and data filters.

B. Register the S3 path as an AWS Lake Formation location: This is the foundation of the solution. Lake Formation acts as a central governance service for the data lake. Registering the S3 path with Lake Formation allows you to apply granular access control and data filters. Without this, you can't leverage Lake Formation's capabilities. https://aws.amazon.com/lake-formation/

D. Enable fine-grained access control in AWS Lake Formation. Add a data filter for each Region: This is how you implement the required access restriction. Fine-grained access control in Lake Formation enables you to define which users (in this case, IAM roles assumed by the HR departments) can access which data, based on criteria. Data filters allow you to restrict access to specific rows or columns based on conditions, in this instance, the Region. By defining a data filter for each Region, you ensure that each HR department can only see data for employees in their respective Region. This avoids the need to create multiple buckets or modify existing IAM roles. This approach allows for centralized management of security policies for the data lake. Furthermore, Lake Formation integrates with AWS Glue for data cataloging and also provides audit logs via CloudTrail. https://docs.aws.amazon.com/lake-formation/latest/dg/access-control-data-filtering.html

Why other options are incorrect:

A. Use data filters for each Region to register the S3 paths as data locations: This is not a standalone operation that secures the data. It requires using a governance service like Lake Formation.
C. Modify the IAM roles of the HR departments to add a data filter for each department's Region: While technically possible, modifying IAM roles can become complex and difficult to manage as the number of departments and Regions increases. This increases operational overhead significantly compared to using Lake Formation's centralized control. IAM policies become very large and complex.
E. Create a separate S3 bucket for each Region. Configure an IAM policy to allow S3 access. Restrict access based on Region: This creates unnecessary operational complexity due to managing multiple S3 buckets. The amount of S3 buckets that need to be managed increase as the company expands. It duplicates storage and potentially complicates data processing and analytics. Using data filters within Lake Formation on a single S3 bucket is more efficient.

Question: 43 CertyIQ

A company uses AWS Step Functions to orchestrate a data pipeline. The pipeline consists of Amazon EMR jobs that ingest data from data sources and store the data in an Amazon S3 bucket. The pipeline also includes EMR jobs that load the data to Amazon Redshift.
The company's cloud infrastructure team manually built a Step Functions state machine. The cloud infrastructure team launched an EMR cluster into a VPC to support the EMR jobs. However, the deployed Step Functions state machine is not able to run the EMR jobs.
Which combination of steps should the company take to identify the reason the Step Functions state machine is not able to run the EMR jobs? (Choose two.)

A.Use AWS CloudFormation to automate the Step Functions state machine deployment. Create a step to pause the state machine during the EMR jobs that fail. Configure the step to wait for a human user to send approval through an email message. Include details of the EMR task in the email message for further analysis.

B.Verify that the Step Functions state machine code has all IAM permissions that are necessary to create and run the EMR jobs. Verify that the Step Functions state machine code also includes IAM permissions to access the Amazon S3 buckets that the EMR jobs use. Use Access Analyzer for S3 to check the S3 access properties.

C.Check for entries in Amazon CloudWatch for the newly created EMR cluster. Change the AWS Step Functions state machine code to use Amazon EMR on EKS. Change the IAM access policies and the security group configuration for the Step Functions state machine code to reflect inclusion of Amazon Elastic Kubernetes Service (Amazon EKS).

D.Query the flow logs for the VPC. Determine whether the traffic that originates from the EMR cluster can successfully reach the data providers. Determine whether any security group that might be attached to the Amazon EMR cluster allows connections to the data source servers on the informed ports.

E.Check the retry scenarios that the company configured for the EMR jobs. Increase the number of seconds in the interval between each EMR task. Validate that each fallback state has the appropriate catch for each decision state. Configure an Amazon Simple Notification Service (Amazon SNS) topic to store the error

messages.

Answer: BD

Explanation:

The correct answer is BD. Here's why:

B - Verify IAM Permissions: Step Functions needs proper IAM permissions to interact with other AWS services like EMR and S3. If the state machine lacks the necessary permissions to create/run EMR jobs or access S3 buckets, it will fail. Access Analyzer for S3 can help identify if any S3 bucket policies are overly permissive or have unintended access. This is fundamental to AWS security and service integration. (Reference: https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-iam-role.html, https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-analyzer.html)
D - Query VPC Flow Logs: If the EMR cluster is in a VPC and unable to reach data sources or other necessary endpoints, VPC Flow Logs can pinpoint connectivity issues. They can reveal if traffic is being blocked by network ACLs, security groups, or routing configurations. This is critical for diagnosing network-related failures. (Reference: https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html)

Here's why the other options are less suitable:

A - CloudFormation Automation (Not the Immediate Problem): While automation is beneficial, it doesn't directly address the cause of the failure. Debugging needs to precede automation. The suggested "pause" and "email" are workaround actions rather than diagnostic steps.
C - EMR on EKS (Changing Technology): Switching to EMR on EKS would require substantial changes to the pipeline architecture and is not a necessary step for identifying the root cause of the initial problem with EMR on EC2. Checking CloudWatch for the existing cluster is important, but not the main focus.
E - Retry Scenarios: While retry mechanisms are important for resilience, they don't solve the underlying issue causing the EMR jobs to fail in the first place. This is focused on a symptom not the cause.

Therefore, the most effective initial steps are to examine IAM permissions and VPC connectivity because these are common causes for Step Functions failures when interacting with services within a VPC.

Question: 44 CertyIQ

A company is developing an application that runs on Amazon EC2 instances. Currently, the data that the application generates is temporary. However, the company needs to persist the data, even if the EC2 instances are terminated.
A data engineer must launch new EC2 instances from an Amazon Machine Image (AMI) and configure the instances to preserve the data.
Which solution will meet this requirement?

A.Launch new EC2 instances by using an AMI that is backed by an EC2 instance store volume that contains the application data. Apply the default settings to the EC2 instances.

B.Launch new EC2 instances by using an AMI that is backed by a root Amazon Elastic Block Store (Amazon EBS) volume that contains the application data. Apply the default settings to the EC2 instances.

C.Launch new EC2 instances by using an AMI that is backed by an EC2 instance store volume. Attach an Amazon Elastic Block Store (Amazon EBS) volume to contain the application data. Apply the default settings to the EC2 instances.

D.Launch new EC2 instances by using an AMI that is backed by an Amazon Elastic Block Store (Amazon EBS) volume. Attach an additional EC2 instance store volume to contain the application data. Apply the default settings to the EC2 instances.

Answer: C

Explanation:

The correct answer is C. Here's why:

The key requirement is persisting data even when EC2 instances are terminated. Instance store volumes are ephemeral, meaning their data is lost when the instance is stopped, terminated, or fails. This eliminates options A and D because they rely on instance store volumes for data persistence.

While an EBS-backed AMI (option B) can persist the root volume's data, it's not the most suitable approach for persisting application data generated during runtime. Modifying the root volume's AMI for every data change is inefficient and doesn't align with best practices. Also, it might be against security policies since we should not be modifying the core image with runtime application data.

Attaching a separate EBS volume (option C) is the ideal solution. EBS volumes are persistent block storage that can be detached from one instance and attached to another. This allows the application data to survive instance terminations. You can launch new EC2 instances from any suitable base AMI (doesn't need to pre-contain the data) and then simply attach the existing EBS volume containing the application data. This offers flexibility, scalability, and data durability.

In summary, using a separate EBS volume provides the persistence needed, decouples the data from the EC2 instance lifecycle, and aligns with cloud storage best practices.

Relevant Links:

Amazon EBS: https://aws.amazon.com/ebs/
Amazon EC2 Instance Store: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html
Amazon Machine Images (AMIs): https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

Question: 45 CertyIQ

A company uses Amazon Athena to run SQL queries for extract, transform, and load (ETL) tasks by using Create Table As Select (CTAS). The company must use Apache Spark instead of SQL to generate analytics.
Which solution will give the company the ability to use Spark to access Athena?

A.Athena query settings
B.Athena workgroup
C.Athena data source
D.Athena query editor

Answer: B

Explanation:

The correct answer is **B. Athena workgroup**. Here's why:

The core requirement is to enable Apache Spark to access data managed by Amazon Athena. Athena's primary function is to execute SQL queries against data stored in Amazon S3. To integrate Spark with Athena, we need a mechanism for Spark to interact with Athena's data catalog and query engine.

Athena Workgroups: Workgroups are a key feature in Athena for isolating queries and managing resources. They also control access to Athena data and configurations. Creating a workgroup allows you to configure settings specific to Spark's interaction with Athena, particularly in terms of query execution and resource utilization. By using a workgroup, you can define settings for the Spark application to connect to Athena.

Why other options are incorrect:

Athena Query Settings: Query settings are associated with individual queries. They don't provide a centralized and reusable mechanism for Spark to consistently interact with Athena.
Athena Data Source: While a data source configuration is necessary for Athena to access the underlying data in S3, it doesn't directly facilitate Spark's interaction with Athena. Spark needs to communicate with Athena's query engine, not just the underlying S3 data directly (although that's possible, it bypasses Athena's metadata management and query optimization).
Athena Query Editor: The query editor is a web-based interface for manually running queries. It doesn't provide programmatic access for Spark.

Using a JDBC or ODBC driver, a Spark application can connect to an Athena workgroup. The driver uses the configured settings of the workgroup to correctly interact with Athena.

In summary, Athena workgroups provides a central and reusable mechanism for Spark to connect and use Athena.

For further research, check these links:

https://docs.aws.amazon.com/athena/latest/ug/workgroups-concept.html
https://aws.amazon.com/blogs/big-data/using-apache-spark-with-amazon-athena-to-query-data-in-place/

Question: 46 CertyIQ

A company needs to partition the Amazon S3 storage that the company uses for a data lake. The partitioning will use a path of the S3 object keys in the following format: s3://bucket/prefix/year=2023/month=01/day=01.
A data engineer must ensure that the AWS Glue Data Catalog synchronizes with the S3 storage when the company adds new partitions to the bucket.
Which solution will meet these requirements with the LEAST latency?

A.Schedule an AWS Glue crawler to run every morning.
B.Manually run the AWS Glue CreatePartition API twice each day.
C.Use code that writes data to Amazon S3 to invoke the Boto3 AWS Glue create_partition API call.
D.Run the MSCK REPAIR TABLE command from the AWS Glue console.

Answer: C

Explanation:

The correct answer is C because it provides the lowest latency and most automated approach to synchronizing the AWS Glue Data Catalog with new partitions added to the S3 data lake.

Here's why:

Option C: (Correct) Proactive Partition Creation: Invoking the CreatePartition API using Boto3 (AWS SDK for Python) immediately after writing data to S3 ensures that the Glue Data Catalog is updated in real-time (or very close to it). This reduces the time window where the Data Catalog is out of sync with the actual data in S3. The code writing the data can directly notify Glue about the new partition, leading to minimal latency. This approach avoids the need for scheduled scans or manual interventions.

Option A: Scheduled Crawler (Higher Latency): Scheduling a Glue crawler is a valid approach for discovering and registering partitions. However, it introduces latency because the crawler only runs periodically (in this case, every morning). Any partitions added after the crawler runs and before the next scheduled run won't be immediately reflected in the Data Catalog. This delay can hinder real-time or near real-time query performance.

Option B: Manual CreatePartition (Highest Latency, Error-Prone): Manually running the CreatePartition API is

the least desirable option. It requires human intervention, which is prone to errors and delays. This solution is not scalable or maintainable, especially as the number of partitions grows.

Option D: MSCK REPAIR TABLE (Post-Facto, Potentially Inefficient): MSCK REPAIR TABLE is a Hive metastore command that scans the S3 path for partitions and updates the metastore (in this case, the Glue Data Catalog). While it works, it's a reactive approach. It detects and fixes inconsistencies after they occur. Moreover, scanning the entire S3 path can be resource-intensive and slow, especially for large datasets. It also doesn't scale well because each execution requires scanning the entire bucket. It is best used for recovering from unexpected state corruption.

Therefore, using the CreatePartition API directly within the data writing process offers the most immediate and automated synchronization, ensuring that the Data Catalog reflects the current state of the S3 data lake with the least latency.

Supporting Links:

AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
AWS Glue CreatePartition API: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-catalog-partitions.html#aws-glue-api-catalog-partitions-CreatePartition
Boto3 Glue Client: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/glue.html
MSCK REPAIR TABLE: https://docs.aws.amazon.com/athena/latest/ug/msck-repair-table.html

Question: 47 CertyIQ

A media company uses software as a service (SaaS) applications to gather data by using third-party tools. The company needs to store the data in an Amazon S3 bucket. The company will use Amazon Redshift to perform analytics based on the data.
Which AWS service or feature will meet these requirements with the LEAST operational overhead?

A.Amazon Managed Streaming for Apache Kafka (Amazon MSK)
B.Amazon AppFlow
C.AWS Glue Data Catalog
D.Amazon Kinesis

Answer: B

Explanation:

Here's a detailed justification for why Amazon AppFlow is the best choice for this scenario, along with links for further reading:

The media company wants to transfer data from SaaS applications into an S3 bucket for analysis in Redshift, while minimizing operational overhead. Several AWS services could potentially move data, but AppFlow is specifically designed for this type of data transfer.

Why AppFlow is the best fit: Amazon AppFlow is a fully managed integration service that enables you to securely transfer data between SaaS applications and AWS services like S3. It requires minimal coding and has built-in connectors for common SaaS applications (e.g., Salesforce, Marketo, Google Analytics). The "least operational overhead" requirement is precisely what AppFlow addresses by automating the data transfer process.

Why other options are less suitable:

Amazon MSK: Amazon MSK is for streaming data, suitable for continuous data flow, which isn't explicitly required in this case. Setting up and managing a Kafka cluster involves significant operational overhead.

AWS Glue Data Catalog: Glue Data Catalog is a metadata repository, useful for discovering and understanding data, but it doesn't transfer data. It would be useful after the data is in S3, for Redshift to access the data, but it isn't the solution for data ingestion itself.
Amazon Kinesis: Similar to Amazon MSK, Kinesis is focused on real-time streaming data. It might be overkill and introduce unnecessary complexity if the data transfer from SaaS applications can be handled in batches or on a schedule.

AppFlow allows configuring data transformations during the transfer, if necessary, and automates the data transfer process, including error handling and monitoring. Using AppFlow significantly reduces the need for custom code and simplifies the overall data integration pipeline, directly addressing the "least operational overhead" requirement.

Supporting Links:

Amazon AppFlow: https://aws.amazon.com/appflow/
AWS Glue Data Catalog: https://aws.amazon.com/glue/
Amazon MSK: https://aws.amazon.com/msk/
Amazon Kinesis: https://aws.amazon.com/kinesis/

Question: 48 CertyIQ

A data engineer is using Amazon Athena to analyze sales data that is in Amazon S3. The data engineer writes a query to retrieve sales amounts for 2023 for several products from a table named sales_data. However, the query does not return results for all of the products that are in the sales_data table. The data engineer needs to troubleshoot the query to resolve the issue.
The data engineer's original query is as follows:
SELECT product_name, sum(sales_amount)

FROM sales_data -

WHERE year = 2023 -

GROUP BY product_name -
How should the data engineer modify the Athena query to meet these requirements?

A.Replace sum(sales_amount) with count(*) for the aggregation.
B.Change WHERE year = 2023 to WHERE extract(year FROM sales_data) = 2023.
C.Add HAVING sum(sales_amount) > 0 after the GROUP BY clause.
D.Remove the GROUP BY clause.

Answer: B

Explanation:

The original query fails to return all expected results for 2023 sales data, indicating an issue with the WHERE clause filtering. The most probable cause is that the 'year' column isn't a readily extractable year integer, potentially being embedded within a date or timestamp format in the 'sales_data' table.

Option B, changing WHERE year = 2023 to WHERE extract(year FROM sales_data) = 2023, directly addresses this issue. The extract(year FROM sales_data) function extracts the year from the sales_data column (assuming it represents a date or timestamp), allowing the query to correctly filter for sales records specifically from the year 2023. This ensures that all sales figures for 2023 are included in the results, resolving the initial problem of missing data. Athena supports SQL standard functions like extract, making this a viable and efficient solution. https://prestodb.io/docs/current/functions/datetime.html

Option A, replacing sum(sales_amount) with count(*), changes the query's purpose from calculating the total

sales amount to counting the number of sales records. While this might provide insights, it doesn't address the issue of filtering for the correct year, and would not resolve the problem of missing sales data.

Option C, adding HAVING sum(sales_amount) > 0 after the GROUP BY clause, filters out product groups with total sales less than or equal to zero. While useful in some scenarios, it doesn't resolve the problem of the query not retrieving all data for 2023; it merely filters the existing results.

Option D, removing the GROUP BY clause, will result in an error because sales_amount is not part of group by clause and not aggregated in aggregate function.

Therefore, option B is the correct solution because it focuses on correctly filtering the sales data for the specific year, thus ensuring that the query retrieves all relevant records from the sales_data table.

Question: 49 CertyIQ

A data engineer has a one-time task to read data from objects that are in Apache Parquet format in an Amazon S3 bucket. The data engineer needs to query only one column of the data.
Which solution will meet these requirements with the LEAST operational overhead?

A.Configure an AWS Lambda function to load data from the S3 bucket into a pandas dataframe. Write a SQL SELECT statement on the dataframe to query the required column.

B.Use S3 Select to write a SQL SELECT statement to retrieve the required column from the S3 objects.

C.Prepare an AWS Glue DataBrew project to consume the S3 objects and to query the required column.

D.Run an AWS Glue crawler on the S3 objects. Use a SQL SELECT statement in Amazon Athena to query the required column.

Answer: B

Explanation:

The best solution is **B. Use S3 Select to write a SQL SELECT statement to retrieve the required column from the S3 objects.**

Here's why:

Operational Overhead: The core requirement is least operational overhead. S3 Select stands out because it directly queries data within S3, avoiding the need to move or transform the entire dataset. It requires minimal setup.

S3 Select Functionality: S3 Select uses simple SQL expressions to retrieve a subset of data from an object, such as a single column. Since Parquet is a supported format, S3 Select can efficiently parse and query the data.

Cost Efficiency: By only retrieving the necessary column, S3 Select reduces data transfer costs. You only pay for the data scanned and returned.

Alternatives' Drawbacks:

Option A (Lambda + Pandas): This approach involves loading the entire Parquet file into memory within a Lambda function, even though only one column is needed. Pandas provides querying functionality but it's not optimized for large datasets directly in S3. This increases both memory usage and Lambda execution time, leading to higher costs and potential memory limitations. It requires code development and deployment.
Option C (Glue DataBrew): Glue DataBrew is suitable for data cleaning and transformation but it's overkill for a simple query, adds operational complexity because you are not transforming anything and you are working with very big data sets. It requires project setup and configuration.

Option D (Glue Crawler + Athena): This is an acceptable method, however there is too much overhead. This involves crawling the S3 bucket to create metadata in the Glue Data Catalog and then using Athena to query. While Athena is also a good option for querying data in S3, the Glue Crawler adds unnecessary operational overhead for a one-time, single-column retrieval.

Direct Querying: S3 Select aligns perfectly with the task's requirement of directly querying data in S3 without requiring additional services for data loading or transformation, ensuring the least operational overhead.

In summary, S3 Select is the simplest, fastest, and most cost-effective way to query a single column from Parquet files in S3 for a one-time task.

Authoritative Links:

Amazon S3 Select: https://docs.aws.amazon.com/AmazonS3/latest/userguide/selecting-content-from-objects.html

Question: 50 CertyIQ

A company uses Amazon Redshift for its data warehouse. The company must automate refresh schedules for Amazon Redshift materialized views.
Which solution will meet this requirement with the LEAST effort?

A. Use Apache Airflow to refresh the materialized views.
B. Use an AWS Lambda user-defined function (UDF) within Amazon Redshift to refresh the materialized views.
C. Use the query editor v2 in Amazon Redshift to refresh the materialized views.
D. Use an AWS Glue workflow to refresh the materialized views.

Answer: C

Explanation:

The correct answer is C, using the query editor v2 in Amazon Redshift to refresh the materialized views. Here's why:

The primary goal is to automate refresh schedules for Redshift materialized views with the least effort. Query editor v2 offers a straightforward interface for scheduling queries, including REFRESH MATERIALIZED VIEW. This requires minimal setup and is directly integrated within the Redshift console. You can simply create a scheduled query to execute the REFRESH MATERIALIZED VIEW command at desired intervals.

Option A, Apache Airflow, introduces significant overhead. Airflow is a powerful orchestration tool, but setting up and managing an Airflow cluster specifically for this single task is an overkill and requires considerably more effort than necessary. This involves defining DAGs, managing dependencies, and maintaining the Airflow infrastructure.

Option B, Lambda UDF, is also less efficient. While Lambda can execute code, using it within a Redshift UDF to refresh materialized views is complex. It would require setting up proper IAM roles and managing network configurations to allow Redshift to invoke the Lambda function. Furthermore, UDFs have limitations on execution time, which could be problematic for large materialized views.

Option D, AWS Glue workflow, is designed for ETL (Extract, Transform, Load) processes and not specifically for refreshing materialized views. While Glue could be used to achieve this, it would again involve unnecessary complexity of defining crawlers, transformations, and jobs when a simple scheduled query would suffice. Glue is better suited for more complex data processing tasks, rather than just refreshing a MV.

Therefore, leveraging the built-in scheduling capabilities of Redshift's query editor v2 provides the simplest

and most direct approach to automating materialized view refreshes. It requires the least amount of effort, configuration, and management overhead.

Refer to the official Amazon Redshift documentation for more details on scheduled queries:

Amazon Redshift Scheduled Queries
REFRESH MATERIALIZED VIEW

Question: 51 CertyIQ

A data engineer must orchestrate a data pipeline that consists of one AWS Lambda function and one AWS Glue job. The solution must integrate with AWS services.
Which solution will meet these requirements with the LEAST management overhead?

A.Use an AWS Step Functions workflow that includes a state machine. Configure the state machine to run the Lambda function and then the AWS Glue job.

B.Use an Apache Airflow workflow that is deployed on an Amazon EC2 instance. Define a directed acyclic graph (DAG) in which the first task is to call the Lambda function and the second task is to call the AWS Glue job.

C.Use an AWS Glue workflow to run the Lambda function and then the AWS Glue job.

D.Use an Apache Airflow workflow that is deployed on Amazon Elastic Kubernetes Service (Amazon EKS). Define a directed acyclic graph (DAG) in which the first task is to call the Lambda function and the second task is to call the AWS Glue job.

Answer: A

Explanation:

The correct answer is A: Use an AWS Step Functions workflow that includes a state machine. Configure the state machine to run the Lambda function and then the AWS Glue job.

Here's a detailed justification:

AWS Step Functions is a fully managed, serverless orchestration service that allows you to coordinate multiple AWS services into serverless workflows. It's designed for building resilient, auditable, and scalable state machines without managing servers. This aligns perfectly with the requirement of least management overhead.

Option A directly leverages the strengths of Step Functions:

Orchestration: Step Functions can easily orchestrate the execution of the Lambda function and the Glue job in a defined order (Lambda first, then Glue).
Integration: Step Functions integrates seamlessly with AWS Lambda and AWS Glue, making it straightforward to call these services from within the state machine definition.
Serverless: Step Functions is serverless, meaning you don't need to provision or manage any infrastructure. This minimizes operational overhead.
State Management: Step Functions manages the state of the workflow, including error handling and retries. This simplifies the overall pipeline design and improves resilience.

Options B and D involve using Apache Airflow, which, while a powerful orchestration tool, requires more management overhead. You have to manage the Airflow environment itself, whether deployed on EC2 or EKS. This includes maintaining the underlying infrastructure, scaling resources, and handling upgrades. Airflow is a more complex solution than needed for simply orchestrating two tasks.

Option C, using AWS Glue workflows, is specifically designed for orchestrating Glue jobs and crawlers. While Glue workflows can trigger external actions through triggers, they are primarily focused on Glue's ecosystem and aren't the most efficient way to integrate and orchestrate arbitrary Lambda functions before a Glue job.

Step Functions offers more general-purpose orchestration capabilities.

Therefore, Step Functions (Option A) provides the least management overhead because it is a fully managed, serverless orchestration service that integrates directly with the required AWS services (Lambda and Glue). It simplifies the coordination of the pipeline without requiring you to manage any underlying infrastructure.

Relevant Links:

AWS Step Functions: https://aws.amazon.com/step-functions/
AWS Lambda: https://aws.amazon.com/lambda/
AWS Glue: https://aws.amazon.com/glue/

Question: 52 CertyIQ

A company needs to set up a data catalog and metadata management for data sources that run in the AWS Cloud. The company will use the data catalog to maintain the metadata of all the objects that are in a set of data stores. The data stores include structured sources such as Amazon RDS and Amazon Redshift. The data stores also include semistructured sources such as JSON files and .xml files that are stored in Amazon S3.
The company needs a solution that will update the data catalog on a regular basis. The solution also must detect changes to the source metadata.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon Aurora as the data catalog. Create AWS Lambda functions that will connect to the data catalog. Configure the Lambda functions to gather the metadata information from multiple sources and to update the Aurora data catalog. Schedule the Lambda functions to run periodically.

B.Use the AWS Glue Data Catalog as the central metadata repository. Use AWS Glue crawlers to connect to multiple data stores and to update the Data Catalog with metadata changes. Schedule the crawlers to run periodically to update the metadata catalog.

C.Use Amazon DynamoDB as the data catalog. Create AWS Lambda functions that will connect to the data catalog. Configure the Lambda functions to gather the metadata information from multiple sources and to update the DynamoDB data catalog. Schedule the Lambda functions to run periodically.

D.Use the AWS Glue Data Catalog as the central metadata repository. Extract the schema for Amazon RDS and Amazon Redshift sources, and build the Data Catalog. Use AWS Glue crawlers for data that is in Amazon S3 to infer the schema and to automatically update the Data Catalog.

Answer: B

Explanation:

The correct answer is **B. Use the AWS Glue Data Catalog as the central metadata repository. Use AWS Glue crawlers to connect to multiple data stores and to update the Data Catalog with metadata changes. Schedule the crawlers to run periodically to update the metadata catalog.**

Here's a detailed justification:

AWS Glue is specifically designed as a fully managed ETL (Extract, Transform, Load) service and a data catalog. Its Data Catalog serves as a central repository for storing metadata about data assets. AWS Glue crawlers are automated programs that connect to data stores (like RDS, Redshift, and S3) to infer the schema, data types, and other metadata information. Crawlers automatically detect changes to the source metadata and update the Glue Data Catalog accordingly, minimizing operational overhead. Scheduling these crawlers allows for regular updates to the catalog, ensuring that the metadata is current.

Option A is less suitable because managing an Aurora database solely as a data catalog adds unnecessary operational overhead. It requires manually building and maintaining the metadata schema, connection logic, and update mechanisms using Lambda functions, which increases complexity.

Option C, using DynamoDB, suffers from similar issues as Option A. While DynamoDB is a powerful NoSQL

database, it's not designed specifically for metadata management like Glue's Data Catalog. Building the metadata structure and update processes manually would be complex and time-consuming.

Option D is partially correct in using AWS Glue, but it incorrectly suggests manually extracting schema for RDS and Redshift. AWS Glue crawlers can automatically infer the schema for these data sources as well, simplifying the process. The added step to extract the schema manually creates more operational overhead. The crawlers also handle detecting changes to the source metadata automatically.

AWS Glue directly addresses the requirements of a data catalog, metadata management, automated discovery of metadata, and change detection with minimal operational overhead. It is purpose-built for this type of task.

Authoritative Links:

AWS Glue Documentation: https://aws.amazon.com/glue/
AWS Glue Data Catalog: https://docs.aws.amazon.com/glue/latest/dg/components-datacatalog.html
AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html

Question: 53 CertyIQ

A company stores data from an application in an Amazon DynamoDB table that operates in provisioned capacity mode. The workloads of the application have predictable throughput load on a regular schedule. Every Monday, there is an immediate increase in activity early in the morning. The application has very low usage during weekends.
The company must ensure that the application performs consistently during peak usage times.
Which solution will meet these requirements in the MOST cost-effective way?

A.Increase the provisioned capacity to the maximum capacity that is currently present during peak load times.

B.Divide the table into two tables. Provision each table with half of the provisioned capacity of the original table. Spread queries evenly across both tables.

C.Use AWS Application Auto Scaling to schedule higher provisioned capacity for peak usage times. Schedule lower capacity during off-peak times.

D.Change the capacity mode from provisioned to on-demand. Configure the table to scale up and scale down based on the load on the table.

Answer: C

Explanation:

The most cost-effective solution for handling predictable, fluctuating workloads on a DynamoDB table using provisioned capacity mode is **C. Use AWS Application Auto Scaling to schedule higher provisioned capacity for peak usage times. Schedule lower capacity during off-peak times.**

Here's why:

Predictable Workload: The problem explicitly states the workload has predictable throughput load on a regular schedule (Monday morning peaks, weekend lulls). This makes scheduled capacity adjustments ideal.

Cost Efficiency: Application Auto Scaling allows you to define scaling schedules. You can automatically increase read and write capacity units (RCUs/WCUs) before the Monday morning peak and decrease them during weekends when usage is low. This ensures you only pay for the capacity you need, minimizing costs.

Consistent Performance: By pre-emptively scaling up capacity before the peak, the application maintains consistent performance without throttling. Scaling down during off-peak times prevents over-provisioning and reduces unnecessary costs.

Why not A? Simply increasing provisioned capacity to the maximum peak requirement and leaving it there

would be the least cost-effective. You would be paying for unused capacity during off-peak times.

Why not B? Dividing the table would add unnecessary complexity in managing and querying data. It's also not guaranteed to provide better performance without proper sharding and load balancing strategies, which would add more complexity. Spreading queries evenly doesn't address the fundamental problem of insufficient total capacity during the peak.

Why not D? Switching to on-demand capacity mode would incur a different cost structure. While convenient for unpredictable workloads, it's generally more expensive for predictable workloads where you can accurately provision capacity. Furthermore, the question emphasizes cost-effectiveness, and scheduled scaling in provisioned mode directly addresses that. Auto scaling within provisioned mode can be implemented but would not be as effective as scheduled scaling because the response to the load would take some time.

Supporting Concepts:

Provisioned Capacity Mode: DynamoDB's provisioned capacity mode allows you to specify the expected read and write throughput for your table. You are charged based on these provisioned units.
AWS Application Auto Scaling: This service enables you to automatically scale your DynamoDB table capacity based on utilization metrics or, in this case, a schedule.
Scheduled Scaling: A feature of Application Auto Scaling that allows you to set up scaling actions to occur at specific times.

Authoritative Links:

DynamoDB Provisioned Capacity Mode
AWS Application Auto Scaling
Scaling DynamoDB Tables Automatically with AWS Application Auto Scaling

Question: 54 CertyIQ

A company is planning to migrate on-premises Apache Hadoop clusters to Amazon EMR. The company also needs to migrate a data catalog into a persistent storage solution.
The company currently stores the data catalog in an on-premises Apache Hive metastore on the Hadoop clusters.
The company requires a serverless solution to migrate the data catalog.
Which solution will meet these requirements MOST cost-effectively?

A.Use AWS Database Migration Service (AWS DMS) to migrate the Hive metastore into Amazon S3. Configure AWS Glue Data Catalog to scan Amazon S3 to produce the data catalog.

B.Configure a Hive metastore in Amazon EMR. Migrate the existing on-premises Hive metastore into Amazon EMR. Use AWS Glue Data Catalog to store the company's data catalog as an external data catalog.

C.Configure an external Hive metastore in Amazon EMR. Migrate the existing on-premises Hive metastore into Amazon EMR. Use Amazon Aurora MySQL to store the company's data catalog.

D.Configure a new Hive metastore in Amazon EMR. Migrate the existing on-premises Hive metastore into Amazon EMR. Use the new metastore as the company's data catalog.

Answer: B

Explanation:

The correct answer is B because it offers a cost-effective and serverless approach to migrating the Hive metastore. Here's why:

Serverless Solution: AWS Glue Data Catalog is a fully managed, serverless service. This eliminates the need to manage infrastructure for the data catalog, reducing operational overhead and cost.
Metastore Migration: Configuring a Hive metastore in Amazon EMR provides a temporary environment for the

on-premises metastore to be migrated. This is a necessary step to transition the data catalog to the cloud.
External Data Catalog: Configuring the Hive metastore in EMR to use AWS Glue Data Catalog as an external metastore makes the Glue Data Catalog the persistent, central repository for the data catalog after migration. This also avoids the need to retain the EMR metastore.
Cost-Effectiveness: This approach is cost-effective because AWS Glue's pricing is based on usage (storage and metadata operations), which scales with the company's needs. Avoids the cost of more complex solutions like DMS or persistent database infrastructure for Aurora.

Here's why the other options are less ideal:

A: Migrating directly to S3 using DMS and then scanning with Glue is unusual and likely more complex than setting up a temporary Hive metastore.
C: Using Amazon Aurora MySQL is viable, but over-engineered and more costly than AWS Glue Data Catalog, which is specifically designed for this purpose. Also using the term external is misleading as Aurora would be the main metastore in this scenario.
D: Using the EMR Hive metastore directly is not a persistent, serverless solution, as it relies on the EMR cluster's lifecycle. You would have to keep your EMR cluster running indefinitely.

Authoritative Links:

AWS Glue Data Catalog: https://aws.amazon.com/glue/
Amazon EMR: https://aws.amazon.com/emr/

Question: 55 CertyIQ

A company uses an Amazon Redshift provisioned cluster as its database. The Redshift cluster has five reserved ra3.4xlarge nodes and uses key distribution.
A data engineer notices that one of the nodes frequently has a CPU load over 90%. SQL Queries that run on the node are queued. The other four nodes usually have a CPU load under 15% during daily operations.
The data engineer wants to maintain the current number of compute nodes. The data engineer also wants to balance the load more evenly across all five compute nodes.
Which solution will meet these requirements?

A.Change the sort key to be the data column that is most often used in a WHERE clause of the SQL SELECT statement.

B.Change the distribution key to the table column that has the largest dimension.

C.Upgrade the reserved node from ra3.4xlarge to ra3.16xlarge.

D.Change the primary key to be the data column that is most often used in a WHERE clause of the SQL SELECT statement.

Answer: B

Explanation:

The best solution is **B. Change the distribution key to the table column that has the largest dimension.**

Here's a detailed justification:

The problem describes a situation where one node in a Redshift cluster is heavily overloaded while the others are relatively idle. This strongly indicates data skewness – an uneven distribution of data across the compute nodes. When data is skewed, queries involving the skewed data will primarily execute on the node holding the majority of that data, leading to high CPU utilization on that node and query queuing.

The distribution key determines how data is distributed across the nodes. If the distribution key is poorly chosen, it can lead to skew. Choosing a column with a large dimension (high cardinality, many unique values) as the distribution key generally promotes a more even distribution of data. Redshift will attempt to distribute

rows as evenly as possible based on the distribution key's values. When the column has a wide range of values, this tends to result in a better spread across the nodes. This strategy is effective in balancing the workload without altering the cluster size.

Option A (changing the sort key) addresses query performance by influencing the order in which data is stored within each node. It does not directly impact how data is distributed across nodes and therefore won't solve the load imbalance issue. While a proper sort key improves query execution speed within a node, it doesn't address the root cause of one node being overloaded.

Option C (upgrading the node size) could temporarily alleviate the problem by providing more CPU resources to the overloaded node. However, it doesn't address the underlying data skewness issue. The problem will likely reappear as data volumes grow. Upgrading is also a more expensive solution than simply adjusting the distribution key. Furthermore, the question specifically asks for a solution that maintains the current number of compute nodes.

Option D (changing the primary key) is incorrect because Redshift does not enforce primary keys as constraints. While defining a primary key can aid query optimization and documentation, it has no impact on data distribution or query execution from a data skew perspective. Redshift's query optimizer can leverage primary key information, but changing the definition does not change the distribution of data across nodes.

Therefore, changing the distribution key to a high-cardinality column is the most effective way to redistribute the data more evenly across the nodes, balancing the workload and reducing the CPU load on the overloaded node without increasing the cluster size. This improves overall query performance and cluster efficiency.

Further reading:

Amazon Redshift documentation on distribution keys
AWS Big Data Blog: Choosing the right distribution style for Amazon Redshift

Question: 56 CertyIQ

A security company stores IoT data that is in JSON format in an Amazon S3 bucket. The data structure can change when the company upgrades the IoT devices. The company wants to create a data catalog that includes the IoT data. The company's analytics department will use the data catalog to index the data.
Which solution will meet these requirements MOST cost-effectively?

A.Create an AWS Glue Data Catalog. Configure an AWS Glue Schema Registry. Create a new AWS Glue workload to orchestrate the ingestion of the data that the analytics department will use into Amazon Redshift Serverless.

B.Create an Amazon Redshift provisioned cluster. Create an Amazon Redshift Spectrum database for the analytics department to explore the data that is in Amazon S3. Create Redshift stored procedures to load the data into Amazon Redshift.

C.Create an Amazon Athena workgroup. Explore the data that is in Amazon S3 by using Apache Spark through Athena. Provide the Athena workgroup schema and tables to the analytics department.

D.Create an AWS Glue Data Catalog. Configure an AWS Glue Schema Registry. Create AWS Lambda user defined functions (UDFs) by using the Amazon Redshift Data API. Create an AWS Step Functions job to orchestrate the ingestion of the data that the analytics department will use into Amazon Redshift Serverless.

Answer: A

Explanation:

The best solution is A because it leverages AWS Glue for cost-effective data cataloging and schema management while providing a modern serverless data warehousing option. AWS Glue Data Catalog provides a central metadata repository for data sources, making data discoverable and accessible. The AWS Glue Schema Registry tackles the evolving data structure challenge by automatically detecting and managing

schema changes, which is crucial for handling data from upgraded IoT devices. Amazon Redshift Serverless allows running analytics queries without managing infrastructure, providing a cost-effective solution compared to provisioned clusters (option B).

Option B involves creating a provisioned Redshift cluster and Redshift Spectrum, which is more expensive than a serverless solution, especially when resource utilization is sporadic or unpredictable. It also necessitates manual maintenance of the Redshift cluster.

Option C relies on Amazon Athena for data exploration but doesn't offer a robust data ingestion and transformation pipeline into a dedicated data warehouse. Athena's primary use case is direct querying of data in S3, not building an optimized analytics platform.

Option D, while using Glue Data Catalog and Schema Registry, employs Lambda UDFs and Step Functions for data ingestion into Redshift Serverless. This introduces unnecessary complexity and operational overhead compared to directly using AWS Glue's ETL capabilities. Glue's ETL can handle data ingestion, transformation, and loading into Redshift Serverless seamlessly.

Therefore, option A offers the most cost-effective and efficient solution by combining the data cataloging capabilities of AWS Glue with the serverless data warehousing power of Redshift Serverless, effectively addressing the requirements while minimizing cost and operational complexity.

Supporting Links:

AWS Glue Data Catalog: https://aws.amazon.com/glue/features/
AWS Glue Schema Registry: https://aws.amazon.com/blogs/big-data/using-the-aws-glue-schema-registry-to-centrally-manage-schemas-and-improve-data-quality/
Amazon Redshift Serverless: https://aws.amazon.com/redshift/serverless/

Question: 57 CertyIQ

A company stores details about transactions in an Amazon S3 bucket. The company wants to log all writes to the S3 bucket into another S3 bucket that is in the same AWS Region.
Which solution will meet this requirement with the LEAST operational effort?

A.Configure an S3 Event Notifications rule for all activities on the transactions S3 bucket to invoke an AWS Lambda function. Program the Lambda function to write the event to Amazon Kinesis Data Firehose. Configure Kinesis Data Firehose to write the event to the logs S3 bucket.

B.Create a trail of management events in AWS CloudTraiL. Configure the trail to receive data from the transactions S3 bucket. Specify an empty prefix and write-only events. Specify the logs S3 bucket as the destination bucket.

C.Configure an S3 Event Notifications rule for all activities on the transactions S3 bucket to invoke an AWS Lambda function. Program the Lambda function to write the events to the logs S3 bucket.

D.Create a trail of data events in AWS CloudTraiL. Configure the trail to receive data from the transactions S3 bucket. Specify an empty prefix and write-only events. Specify the logs S3 bucket as the destination bucket.

Answer: D

Explanation:

Here's a detailed justification for why option D is the best solution and why the other options are not as suitable for logging S3 write operations with the least operational effort:

Why Option D is Correct: Create a CloudTrail data event trail

CloudTrail Data Events: CloudTrail allows you to log data events, which specifically track object-level API activity on S3 buckets, including PutObject (writes), GetObject (reads), and DeleteObject operations. This is

exactly what the company needs – a record of writes to the S3 bucket.
Data Event Selection: By configuring a trail to track data events for the transactions S3 bucket, you directly capture the required information without needing to create custom logic.
Minimal Configuration: Specifying an empty prefix ensures that all objects within the bucket are monitored. Selecting "write-only" events focuses the logging on the specific operations of interest, reducing unnecessary data.
Direct Delivery: CloudTrail directly delivers the logs to the specified logs S3 bucket, eliminating the need for intermediate services or custom code.
Least Operational Effort: CloudTrail is a managed service designed for logging AWS API calls. This makes it much easier to set up and maintain than alternatives involving Lambda functions and Kinesis.

Why Other Options are Incorrect:

Option A (S3 Event Notifications, Lambda, Kinesis Data Firehose): This is an overly complex solution. S3 Event Notifications can trigger Lambda, but directing those events through Kinesis Data Firehose to another S3 bucket adds unnecessary overhead. It requires configuring and managing multiple services, writing and maintaining Lambda code, and dealing with potential Kinesis buffering issues.
Option B (CloudTrail Management Events): CloudTrail management events track operations performed on AWS resources themselves (e.g., creating a bucket, modifying IAM roles). They do not track object-level API activity such as writing data to an S3 bucket. Therefore, they are unsuitable for this use case.
Option C (S3 Event Notifications, Lambda): While simpler than Option A, this still involves writing and maintaining a Lambda function to handle the S3 events. It's also less efficient than CloudTrail's built-in logging, especially for capturing a comprehensive and auditable history of all writes.

In Summary:

Option D, leveraging CloudTrail data events, offers the most straightforward, efficient, and least operationally intensive way to log all writes to an S3 bucket into another S3 bucket. It utilizes a managed service specifically designed for logging API activity, minimizing the need for custom code and complex configurations.

Authoritative Links:

AWS CloudTrail: https://aws.amazon.com/cloudtrail/
CloudTrail Data Events: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/logging-data-events-with-cloudtrail.html
S3 Event Notifications: https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html

Question: 58 CertyIQ

A data engineer needs to maintain a central metadata repository that users access through Amazon EMR and Amazon Athena queries. The repository needs to provide the schema and properties of many tables. Some of the metadata is stored in Apache Hive. The data engineer needs to import the metadata from Hive into the central metadata repository.
Which solution will meet these requirements with the LEAST development effort?

A.Use Amazon EMR and Apache Ranger.
B.Use a Hive metastore on an EMR cluster.
C.Use the AWS Glue Data Catalog.
D.Use a metastore on an Amazon RDS for MySQL DB instance.

Answer: C

Explanation:

The correct answer is **C. Use the AWS Glue Data Catalog.**

Here's why this is the most suitable solution:

AWS Glue Data Catalog serves as a centralized metadata repository specifically designed for AWS data lakes and analytics services. It provides a persistent metastore to store table definitions, schema information, data lineage, and other metadata. Critically, it's integrated seamlessly with both Amazon EMR and Amazon Athena.

Importing metadata from Hive into the Glue Data Catalog can be achieved with minimal development effort. Glue provides built-in crawlers that can automatically scan data sources (including Hive metastores) and infer schema, creating table definitions in the Glue Data Catalog. This eliminates the need for manual schema definition and management.

Option A (Amazon EMR and Apache Ranger) is not ideal because Apache Ranger primarily focuses on security and access control. While Ranger can integrate with Hive, it doesn't provide a central metadata repository as effectively as Glue Data Catalog. It requires more configuration and management for metadata consolidation.

Option B (Hive metastore on an EMR cluster) creates a Hive-centric solution tied to an EMR cluster's lifecycle. This isn't a central, persistent repository accessible independently by other services like Athena. Maintaining high availability for the EMR cluster would also add unnecessary complexity. It's also less scalable.

Option D (Metastore on an Amazon RDS for MySQL DB instance) is viable but requires more manual configuration and management. While you could host a metastore on RDS, AWS Glue Data Catalog abstracts away the complexity of managing the underlying database, schema, and scaling. The Glue crawler is also a significant advantage for automatically discovering and importing metadata, which is lacking in a standalone RDS metastore setup.

Therefore, AWS Glue Data Catalog offers the most integrated, managed, and least-effort approach for maintaining a central metadata repository accessible by both Amazon EMR and Amazon Athena, especially when the initial metadata is housed in Hive.

Supporting Links:

AWS Glue Data Catalog: https://aws.amazon.com/glue/features/
AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html

Question: 59 CertyIQ

A company needs to build a data lake in AWS. The company must provide row-level data access and column-level data access to specific teams. The teams will access the data by using Amazon Athena, Amazon Redshift Spectrum, and Apache Hive from Amazon EMR.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon S3 for data lake storage. Use S3 access policies to restrict data access by rows and columns. Provide data access through Amazon S3.

B.Use Amazon S3 for data lake storage. Use Apache Ranger through Amazon EMR to restrict data access by rows and columns. Provide data access by using Apache Pig.

C.Use Amazon Redshift for data lake storage. Use Redshift security policies to restrict data access by rows and columns. Provide data access by using Apache Spark and Amazon Athena federated queries.

D.Use Amazon S3 for data lake storage. Use AWS Lake Formation to restrict data access by rows and columns. Provide data access through AWS Lake Formation.

Answer: D

Explanation:

The correct answer is **D. Use Amazon S3 for data lake storage. Use AWS Lake Formation to restrict data**

access by rows and columns. Provide data access through AWS Lake Formation.

Here's why:

S3 for Data Lake Storage: Amazon S3 is the ideal choice for data lake storage due to its scalability, durability, and cost-effectiveness. It can store vast amounts of structured, semi-structured, and unstructured data in its native formats.

AWS Lake Formation for Granular Access Control: AWS Lake Formation provides fine-grained data access control at the row and column levels. It centralizes security management for data in the data lake, simplifying the process of granting and revoking permissions. It also integrates seamlessly with Athena, Redshift Spectrum, and EMR (Hive), addressing the question's specific access requirements.

Least Operational Overhead: Lake Formation simplifies the data access control process, reducing the operational overhead compared to managing security policies directly in S3 or implementing complex solutions with Apache Ranger. It provides a central place to define and manage data access policies.

Let's examine why other options are less suitable:

A. S3 Access Policies: While S3 access policies can restrict access, they become complex and difficult to manage at row and column levels, especially across different services like Athena, Redshift Spectrum, and Hive. It lacks the central management capabilities offered by Lake Formation.

B. Apache Ranger: Apache Ranger, deployed through EMR, can provide row and column-level access control. However, setting it up and maintaining it introduces significant operational overhead, especially when integrating with services outside of the EMR ecosystem like Athena and Redshift Spectrum. This approach necessitates managing another service (Ranger) and its configurations. Moreover, Apache Pig is not used to provide data access by the specific teams, per the use-case.

C. Amazon Redshift: Amazon Redshift is a data warehouse, not a data lake. While it supports security policies, it's not designed for storing the large, diverse datasets typically found in a data lake. Also, it is not optimized for storing the data in its native raw format. In addition, forcing all data access through Redshift, particularly by Spark and Athena federated queries, can add complexity and latency.

In summary, AWS Lake Formation is designed to address the specific requirements of the question (row and column-level access control for Athena, Redshift Spectrum, and Hive with minimal overhead), making it the optimal solution.

Supporting Links:

AWS Lake Formation: https://aws.amazon.com/lake-formation/
Amazon S3: https://aws.amazon.com/s3/

Question: 60 CertyIQ

An airline company is collecting metrics about flight activities for analytics. The company is conducting a proof of concept (POC) test to show how analytics can provide insights that the company can use to increase on-time departures.
The POC test uses objects in Amazon S3 that contain the metrics in .csv format. The POC test uses Amazon Athena to query the data. The data is partitioned in the S3 bucket by date.
As the amount of data increases, the company wants to optimize the storage solution to improve query performance.
Which combination of solutions will meet these requirements? (Choose two.)

A.Add a randomized string to the beginning of the keys in Amazon S3 to get more throughput across partitions.

B.Use an S3 bucket that is in the same account that uses Athena to query the data.

C.Use an S3 bucket that is in the same AWS Region where the company runs Athena queries.

D.Preprocess the .csv data to JSON format by fetching only the document keys that the query requires.

E.Preprocess the .csv data to Apache Parquet format by fetching only the data blocks that are needed for predicates.

Answer: CE

Explanation:

Let's break down why options C and E are the correct solutions for optimizing the airline company's data storage and query performance in this scenario.

Option C: Use an S3 bucket that is in the same AWS Region where the company runs Athena queries.

Athena's performance is significantly impacted by data locality. When the S3 bucket and Athena reside in the same AWS Region, data transfer latency is minimized. This is because the data doesn't need to travel across regions, resulting in faster query execution. AWS prioritizes data transfer within a region over cross-region transfers, leading to lower costs and better performance. This is a fundamental best practice in AWS for any services that interact with S3 for data processing or querying.https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/

Option E: Preprocess the .csv data to Apache Parquet format by fetching only the data blocks that are needed for predicates.

Apache Parquet is a columnar storage format optimized for analytical queries. Unlike row-based formats like CSV, Parquet stores data by column. This means Athena only reads the specific columns required by a query, reducing I/O and improving performance. Furthermore, Parquet supports predicate pushdown, allowing Athena to filter data based on WHERE clause conditions before reading the data from S3. This significantly reduces the amount of data that needs to be scanned, leading to substantial performance gains. Parquet also supports compression, reducing storage costs.https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/

Why other options are incorrect:

Option A: Adding a randomized string to the S3 key is relevant for write throughput when dealing with S3's request rate limits and doesn't directly optimize Athena query performance once the data is stored. It's primarily about distributing write requests across different partitions to avoid throttling, not read performance.

Option B: Using an S3 bucket in the same account has no significant direct effect on Athena query performance. IAM permissions might be slightly simpler to manage within the same account, but the performance boost is negligible compared to regional locality and columnar storage.

Option D: While converting to JSON might offer some benefits over CSV, JSON is still a row-based format. It doesn't provide the columnar advantages and predicate pushdown offered by Parquet for analytical queries. Therefore, Parquet is the significantly better option for query performance.

Question: 61 CertyIQ

A company uses Amazon RDS for MySQL as the database for a critical application. The database workload is mostly writes, with a small number of reads.
A data engineer notices that the CPU utilization of the DB instance is very high. The high CPU utilization is slowing down the application. The data engineer must reduce the CPU utilization of the DB Instance.
Which actions should the data engineer take to meet this requirement? (Choose two.)

A.Use the Performance Insights feature of Amazon RDS to identify queries that have high CPU utilization. Optimize the problematic queries.

B.Modify the database schema to include additional tables and indexes.

C.Reboot the RDS DB instance once each week.

D.Upgrade to a larger instance size.

E.Implement caching to reduce the database query load.

Answer: AD

Explanation:

The correct answer is AD. Here's a detailed justification:

A. Use the Performance Insights feature of Amazon RDS to identify queries that have high CPU utilization. Optimize the problematic queries.

Performance Insights is an RDS feature specifically designed to diagnose database performance issues. It allows you to identify the queries that are consuming the most CPU resources. These queries are often inefficient or poorly written. By identifying and optimizing these "top" queries (e.g., through query rewriting, adding indexes, or optimizing table structures), you can significantly reduce the CPU load on the database server. This is a direct approach to addressing the root cause of the high CPU utilization.

Authoritative Link: Amazon RDS Performance Insights

D. Upgrade to a larger instance size.

Upgrading to a larger instance size (e.g., from db.m5.large to db.m5.xlarge) provides the database with more CPU cores and memory. This increased capacity allows the database to handle the workload with less strain on the existing CPU resources. While this doesn't solve the underlying problem of inefficient queries, it can provide immediate relief from the high CPU utilization and improve application performance. It's akin to upgrading the engine in a car to handle the same load more easily. However, this is often a more expensive solution than query optimization and should ideally be pursued in tandem with option A.

Why other options are not ideal:

B. Modify the database schema to include additional tables and indexes: While adding indexes can improve query performance for read-heavy workloads, the question specifies a write-heavy workload. Adding indexes to a write-heavy database can actually increase CPU utilization due to the overhead of maintaining the indexes on every write operation. Splitting tables without a clear performance benefit is unlikely to solve the CPU problem and may complicate the application. Therefore, this is generally an incorrect approach without more information about specific queries.

C. Reboot the RDS DB instance once each week: Rebooting the instance only provides temporary relief. It clears the cache and restarts processes, but the same high CPU utilization will return as soon as the workload resumes. This is not a sustainable or effective solution to the problem.

E. Implement caching to reduce the database query load. Caching is beneficial primarily for read-heavy workloads. The scenario specifically mentions the workload being mostly write, so caching has less of an impact on reducing the CPU utilization due to writes.

In summary, identifying and optimizing CPU-intensive queries (A) and scaling up the instance size (D) are the most appropriate actions to reduce CPU utilization in a write-heavy RDS for MySQL database.

Question: 62 CertyIQ

A company has used an Amazon Redshift table that is named Orders for 6 months. The company performs weekly updates and deletes on the table. The table has an interleaved sort key on a column that contains AWS Regions. The company wants to reclaim disk space so that the company will not run out of storage space. The company also wants to analyze the sort key column.

Which Amazon Redshift command will meet these requirements?

A.VACUUM FULL Orders
B.VACUUM DELETE ONLY Orders
C.VACUUM REINDEX Orders
D.VACUUM SORT ONLY Orders

Answer: C

Explanation:

Here's a detailed justification for why option C, VACUUM REINDEX Orders, is the most appropriate answer, and why the other options are less suitable, given the scenario.

The company's primary concerns are reclaiming disk space due to weekly updates and deletes, and analyzing the sort key column. Amazon Redshift's VACUUM command family helps with these tasks, but each variation addresses a different aspect of table maintenance.

VACUUM FULL Orders: This option performs a full vacuum, which involves both sorting and merging deleted rows. While it reclaims space and sorts, it is the most resource-intensive and time-consuming vacuum operation. Since the table already uses an interleaved sort key, repeatedly performing VACUUM FULL might be overkill if only deleted rows are the immediate concern.

VACUUM DELETE ONLY Orders: This option removes rows marked for deletion without re-sorting the table. This is a good solution for reclaiming space quickly, but it does not analyze or reorganize the interleaved sort key, which the company wants to do.

VACUUM SORT ONLY Orders: This option sorts the table without removing deleted rows. While this helps maintain sort order, it doesn't address the primary concern of reclaiming disk space consumed by deleted rows.

VACUUM REINDEX Orders: This option is specifically designed to rebuild the indexes on the sort key columns. In the context of an interleaved sort key, VACUUM REINDEX not only reclusters the data according to the sort key (which improves query performance based on the sort key column), but also removes deleted rows in the process of re-indexing. This accomplishes both goals of reclaiming space and optimizing the interleaved sort key, which is based on the AWS Regions column. Because interleaved sort keys are designed to optimize queries where multiple columns are used in WHERE clauses, re-indexing ensures that the blocks of data are optimally arranged based on all interleaved key columns. It achieves the required disk space reclamation because rows marked for deletion will not be included in the rebuilt indexes.

Therefore, VACUUM REINDEX Orders is the most suitable command. It reclaims disk space by removing deleted rows during index rebuilding and analyzes and optimizes the interleaved sort key column for improved query performance based on AWS Regions and possibly other columns involved in the interleaved index. This is a more targeted approach than VACUUM FULL while also addressing the need to optimize the interleaved sort key, which other options omit.

Authoritative Links:

Amazon Redshift VACUUM command: https://docs.aws.amazon.com/redshift/latest/dg/r_VACUUM.html
Working with interleaved sort keys: https://docs.aws.amazon.com/redshift/latest/dg/tutorial-sort-interleaved.html

Question: 63

A manufacturing company wants to collect data from sensors. A data engineer needs to implement a solution that ingests sensor data in near real time.
The solution must store the data to a persistent data store. The solution must store the data in nested JSON format. The company must have the ability to query from the data store with a latency of less than 10 milliseconds.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use a self-hosted Apache Kafka cluster to capture the sensor data. Store the data in Amazon S3 for querying.

B.Use AWS Lambda to process the sensor data. Store the data in Amazon S3 for querying.

C.Use Amazon Kinesis Data Streams to capture the sensor data. Store the data in Amazon DynamoDB for querying.

D.Use Amazon Simple Queue Service (Amazon SQS) to buffer incoming sensor data. Use AWS Glue to store the data in Amazon RDS for querying.

Answer: C

Explanation:

The correct answer is C because it provides the best balance of near real-time ingestion, persistent storage of nested JSON data, low-latency querying, and minimal operational overhead.

Let's analyze why the other options are less suitable:

A: Apache Kafka + Amazon S3: While Kafka is excellent for high-throughput streaming, setting up and managing a self-hosted Kafka cluster introduces significant operational overhead, including managing brokers, zookeepers, and ensuring high availability. S3, being an object store, is not designed for low-latency (sub-10ms) queries. Although services like Athena can query S3 data, the latency would be much higher than 10ms, especially for complex JSON structures.
B: AWS Lambda + Amazon S3: Lambda can process data, but it's not a dedicated streaming ingestion service. Using Lambda as a continuous data ingestion mechanism might lead to invocation limits and cold start issues if the data flow is continuous. Again, S3 doesn't provide the required low-latency querying.
D: Amazon SQS + AWS Glue + Amazon RDS: SQS acts as a message queue, suitable for buffering, but not optimized for continuous high-velocity streams. AWS Glue is an ETL service used for data preparation and transformation, but not ideal for real-time data ingestion. Relational databases (RDS) can provide low latency but are not inherently suited for storing nested JSON. While JSON datatypes are supported, querying complex nested structures often requires more complex SQL and doesn't scale as well as a NoSQL database.

Justification for C: Kinesis Data Streams + DynamoDB:

1. **Near Real-time Ingestion:** Kinesis Data Streams is specifically designed for high-throughput, continuous data ingestion in near real-time. It can handle sensor data streams efficiently. https://aws.amazon.com/kinesis/data-streams/
2. **Persistent Storage:** DynamoDB, a NoSQL database, provides persistent storage and supports storing data in JSON format (including nested structures) natively using its document model. https://aws.amazon.com/dynamodb/
3. **Low-Latency Queries:** DynamoDB is a key-value and document database designed for extremely low-latency reads and writes. It can easily meet the requirement of queries with a latency of less than 10 milliseconds, especially if the data model and access patterns are optimized.
4. **Least Operational Overhead:** Kinesis Data Streams and DynamoDB are fully managed services. AWS handles scaling, patching, and availability, minimizing the operational overhead for the data engineer. You don't need to manage servers or infrastructure components as you would with Kafka.
5. **Data Format Flexibility:** DynamoDB's schema-less nature allows it to easily handle nested JSON structures.
6. **Scalability:** Both Kinesis and DynamoDB are highly scalable, allowing the manufacturing company to increase data ingestion and storage without significant architectural changes.

Therefore, Kinesis Data Streams and DynamoDB offer the most suitable solution for the manufacturing

company's needs by combining near real-time ingestion, nested JSON storage, low-latency querying, and minimal operational overhead.

Question: 64 CertyIQ

A company stores data in a data lake that is in Amazon S3. Some data that the company stores in the data lake contains personally identifiable information (PII). Multiple user groups need to access the raw data. The company must ensure that user groups can access only the PII that they require.
Which solution will meet these requirements with the LEAST effort?

A.Use Amazon Athena to query the data. Set up AWS Lake Formation and create data filters to establish levels of access for the company's IAM roles. Assign each user to the IAM role that matches the user's PII access requirements.

B.Use Amazon QuickSight to access the data. Use column-level security features in QuickSight to limit the PII that users can retrieve from Amazon S3 by using Amazon Athena. Define QuickSight access levels based on the PII access requirements of the users.

C.Build a custom query builder UI that will run Athena queries in the background to access the data. Create user groups in Amazon Cognito. Assign access levels to the user groups based on the PII access requirements of the users.

D.Create IAM roles that have different levels of granular access. Assign the IAM roles to IAM user groups. Use an identity-based policy to assign access levels to user groups at the column level.

Answer: A

Explanation:

The correct answer is A. Here's a detailed justification:

The requirement is to provide different user groups with access to PII data in an S3 data lake, granting them access only to the specific PII they need, while minimizing effort.

Option A utilizes AWS Lake Formation and Athena. Lake Formation provides centralized governance over data lakes, making it easier to define, secure, and manage access to data. By setting up data filters in Lake Formation, you can define row and column-level security policies. These filters are then applied when users query the data using Athena. IAM roles are used to control which users can access the data and which data filters apply to them. This ensures that users only see the data they are authorized to see. This leverages a managed service designed for this purpose, resulting in the least operational overhead.

Option B, using QuickSight, is not ideal. QuickSight's column-level security works on visualizations, not directly on the underlying data source (S3 via Athena). This means the data is still accessible in Athena, and the security is enforced in the QuickSight layer, which could be circumvented.

Option C, building a custom query builder, introduces significant complexity and operational overhead. It requires development, maintenance, and potentially security vulnerabilities introduced through custom code. While Cognito handles user authentication, it doesn't directly integrate with Athena for fine-grained access control like Lake Formation does.

Option D involves managing granular IAM policies directly, which quickly becomes complex and difficult to maintain, especially with multiple user groups and varying PII access requirements. IAM policies for data access are best managed through a service like Lake Formation for simplification and clarity.

Therefore, using Athena with Lake Formation offers the most efficient and secure solution for managing PII access in a data lake, meeting the requirements with the least amount of effort. Lake Formation simplifies security management by centralizing access policies and integrating seamlessly with Athena.

Supporting links:

AWS Lake Formation: https://aws.amazon.com/lake-formation/
Amazon Athena: https://aws.amazon.com/athena/
AWS IAM: https://aws.amazon.com/iam/

Question: 65 CertyIQ

A data engineer must build an extract, transform, and load (ETL) pipeline to process and load data from 10 source systems into 10 tables that are in an Amazon Redshift database. All the source systems generate .csv, JSON, or Apache Parquet files every 15 minutes. The source systems all deliver files into one Amazon S3 bucket. The file sizes range from 10 MB to 20 GB. The ETL pipeline must function correctly despite changes to the data schema. Which data pipeline solutions will meet these requirements? (Choose two.)

A.Use an Amazon EventBridge rule to run an AWS Glue job every 15 minutes. Configure the AWS Glue job to process and load the data into the Amazon Redshift tables.

B.Use an Amazon EventBridge rule to invoke an AWS Glue workflow job every 15 minutes. Configure the AWS Glue workflow to have an on-demand trigger that runs an AWS Glue crawler and then runs an AWS Glue job when the crawler finishes running successfully. Configure the AWS Glue job to process and load the data into the Amazon Redshift tables.

C.Configure an AWS Lambda function to invoke an AWS Glue crawler when a file is loaded into the S3 bucket. Configure an AWS Glue job to process and load the data into the Amazon Redshift tables. Create a second Lambda function to run the AWS Glue job. Create an Amazon EventBridge rule to invoke the second Lambda function when the AWS Glue crawler finishes running successfully.

D.Configure an AWS Lambda function to invoke an AWS Glue workflow when a file is loaded into the S3 bucket. Configure the AWS Glue workflow to have an on-demand trigger that runs an AWS Glue crawler and then runs an AWS Glue job when the crawler finishes running successfully. Configure the AWS Glue job to process and load the data into the Amazon Redshift tables.

E.Configure an AWS Lambda function to invoke an AWS Glue job when a file is loaded into the S3 bucket. Configure the AWS Glue job to read the files from the S3 bucket into an Apache Spark DataFrame. Configure the AWS Glue job to also put smaller partitions of the DataFrame into an Amazon Kinesis Data Firehose delivery stream. Configure the delivery stream to load data into the Amazon Redshift tables.

Answer: BD

Explanation:

The correct answers are **B** and **D**. Here's a detailed justification:

Option B is correct because:

EventBridge Scheduling: EventBridge rules provide a reliable and scalable way to trigger ETL pipelines on a schedule (every 15 minutes, as required).
AWS Glue Workflow: Glue workflows orchestrate complex ETL processes. They allow chaining multiple jobs and crawlers.
On-demand Glue Crawler: The crawler automatically detects schema changes in the source data, addressing the requirement that the pipeline must function despite schema evolution. Crawlers infer the schema and register the tables in the AWS Glue Data Catalog.
Glue Job for Transformation and Loading: After the crawler updates the schema, the Glue job transforms the data according to the new schema and loads it into Amazon Redshift.
Handles Variety of File Types: Glue can handle .csv, JSON, and Apache Parquet files.

Option D is correct because:

Lambda Triggering: A Lambda function triggered by S3 events provides a mechanism to initiate the workflow whenever a new file is loaded into the bucket.
AWS Glue Workflow with Crawler: This component is the same as in option B, and it handles schema changes.
Glue Job for Transformation and Loading: Similar to option B, the Glue job executes the transformation and

loading logic into Amazon Redshift, after the schema is updated by the Glue Crawler.
Event-Driven Architecture: D uses an event-driven setup using an AWS Lambda function that triggers a Glue Workflow, which enables quick processing of files as they arrive in the S3 bucket.

Why other options are incorrect:

Option A: A direct Glue job without a crawler is less robust to schema changes. The job could fail if the schema changes.
Option C: This option creates a complex interaction between two Lambda functions with an event-driven architecture that is not needed, making it less optimal than the others.
Option E: While Lambda can trigger Glue jobs, using Kinesis Data Firehose for loading into Redshift is less efficient and appropriate for large file sizes. Firehose is better suited for streaming data. Glue can natively load data into Redshift with higher throughput. This architecture would be harder to manage and configure.

Authoritative Links:

Amazon EventBridge: https://aws.amazon.com/eventbridge/
AWS Glue: https://aws.amazon.com/glue/
AWS Glue Workflows: https://docs.aws.amazon.com/glue/latest/dg/workflows-using.html
AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
AWS Lambda: https://aws.amazon.com/lambda/
Amazon Redshift: https://aws.amazon.com/redshift/

Question: 66 CertyIQ

A financial company wants to use Amazon Athena to run on-demand SQL queries on a petabyte-scale dataset to support a business intelligence (BI) application. An AWS Glue job that runs during non-business hours updates the dataset once every day. The BI application has a standard data refresh frequency of 1 hour to comply with company policies.
A data engineer wants to cost optimize the company's use of Amazon Athena without adding any additional infrastructure costs.
Which solution will meet these requirements with the LEAST operational overhead?

A.Configure an Amazon S3 Lifecycle policy to move data to the S3 Glacier Deep Archive storage class after 1 day.

B.Use the query result reuse feature of Amazon Athena for the SQL queries.

C.Add an Amazon ElastiCache cluster between the BI application and Athena.

D.Change the format of the files that are in the dataset to Apache Parquet.

Answer: B

Explanation:

The most cost-effective and least operationally burdensome solution is **B. Use the query result reuse feature of Amazon Athena for the SQL queries.**

Here's a detailed justification:

Athena's query result reuse is a built-in feature that automatically caches and reuses query results. If a query is run again within a specified timeframe (configurable), and the underlying data hasn't changed, Athena retrieves the results from the cache instead of re-scanning the data in S3. Since the BI application refreshes every hour and the dataset is updated only daily, the majority of queries during that hourly window will be identical and thus be served from the cache. This significantly reduces data scanned by Athena, translating directly to cost savings because Athena's pricing is primarily based on data scanned per query. This method avoids adding any infrastructure overhead, which is vital considering the constraint.

Option A (S3 Lifecycle to Glacier Deep Archive) would dramatically increase query latency and thus negatively impact the BI application's responsiveness. S3 Glacier Deep Archive is designed for long-term archival, not frequent retrieval.

Option C (ElastiCache) would introduce infrastructure management overhead. ElastiCache requires provisioning, configuration, scaling, and monitoring, adding unnecessary complexity and potential costs, which directly contradicts the requirement for minimal operational overhead and cost optimization. While it could theoretically cache query results, Athena's built-in feature already fulfills this purpose.

Option D (Parquet format) is beneficial for cost optimization in general by reducing data size and improving query performance. However, given the immediate requirement for cost optimization without adding infrastructure costs and with minimal operational overhead, and the already existing dataset and Glue job, changing the data format would introduce substantial initial effort (data conversion) and potentially disrupt existing ETL processes. Also, it may not provide immediate and noticeable impact within the defined timeframe as query result reuse. The query result reuse feature offers immediate cost reduction with zero setup.

Therefore, Athena's query result reuse feature directly addresses the problem by leveraging existing capabilities and achieving cost optimization with minimal effort and no additional infrastructure.

Authoritative links:

Amazon Athena Pricing: https://aws.amazon.com/athena/pricing/ (Shows cost based on data scanned.)
Using Query Result Reuse: https://docs.aws.amazon.com/athena/latest/ug/querying-with-reuse.html (Explains the feature and its configuration.)

Question: 67 CertyIQ

A company's data engineer needs to optimize the performance of table SQL queries. The company stores data in an Amazon Redshift cluster. The data engineer cannot increase the size of the cluster because of budget constraints.
The company stores the data in multiple tables and loads the data by using the EVEN distribution style. Some tables are hundreds of gigabytes in size. Other tables are less than 10 MB in size.
Which solution will meet these requirements?

A.Keep using the EVEN distribution style for all tables. Specify primary and foreign keys for all tables.
B.Use the ALL distribution style for large tables. Specify primary and foreign keys for all tables.
C.Use the ALL distribution style for rarely updated small tables. Specify primary and foreign keys for all tables.
D.Specify a combination of distribution, sort, and partition keys for all tables.

Answer: C

Explanation:

The correct answer is C. Let's break down why this solution works and why the others don't.

Why C is correct:

ALL Distribution for Small, Rarely Updated Tables: The ALL distribution style replicates the entire table to every node in the Amazon Redshift cluster. For small tables (less than 10 MB), the overhead of replication is minimal. Since the tables are rarely updated, the cost of propagating changes across all nodes is also low. This allows each node to perform joins locally without needing to transfer data across the network, significantly improving query performance, especially when joining these small tables with larger distributed tables. This is a key optimization technique in data warehousing.
Primary and Foreign Keys (Informational): While not directly related to data distribution, specifying primary

and foreign keys provides valuable metadata to the Redshift query optimizer. It can assist in generating more efficient query plans, though the actual impact is less pronounced than the distribution strategy. The query optimizer knows about the data relationships allowing it to make better choices.

Why the other options are incorrect:

A. Keep using the EVEN distribution style for all tables: The EVEN distribution style distributes rows evenly across all nodes in the cluster. While simple, it doesn't consider data relationships, leading to significant data redistribution during joins. This data movement across the network becomes a bottleneck, especially for large tables. This distribution style does not optimize the location of the data for querying.
B. Use the ALL distribution style for large tables: Replicating large tables (hundreds of gigabytes) to every node is highly inefficient. It consumes excessive storage on each node, increases the time for data loading and updates dramatically, and may even lead to performance degradation due to resource contention (memory, disk I/O) on each node.
D. Specify a combination of distribution, sort, and partition keys for all tables: While crucial in general Redshift optimization, this isn't the most targeted solution for the described scenario. While distribution keys would help, the prompt clearly identified that some tables are small and this is where a different strategy is needed. Distribution, sort, and partition keys is a general recommendation that does not optimize this particular problem.

In summary: ALL distribution is a good strategy for small, relatively static tables in Amazon Redshift, enabling faster local joins. Specifying keys helps the query optimizer find efficient ways to perform queries.

Authoritative Links:

Amazon Redshift Data Distribution Styles:
https://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
Amazon Redshift Key Concepts https://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-sort-dist.html

Question: 68 CertyIQ

A company receives .csv files that contain physical address data. The data is in columns that have the following names: Door_No, Street_Name, City, and Zip_Code. The company wants to create a single column to store these values in the following format:


```
{
        "Door_No": "24",
        "Street_Name": "AAA street",
        "City": "BBB",
        "Zip_Code": "111111"
}
```


Which solution will meet this requirement with the LEAST coding effort?

A.Use AWS Glue DataBrew to read the files. Use the NEST_TO_ARRAY transformation to create the new column.

B.Use AWS Glue DataBrew to read the files. Use the NEST_TO_MAP transformation to create the new column.

C.Use AWS Glue DataBrew to read the files. Use the PIVOT transformation to create the new column.

D.Write a Lambda function in Python to read the files. Use the Python data dictionary type to create the new column.

Answer: B

Explanation:

Use AWS Glue DataBrew to read the files. Use the NEST_TO_MAP transformation to create the new column.

Question: 69 CertyIQ

A company receives call logs as Amazon S3 objects that contain sensitive customer information. The company must protect the S3 objects by using encryption. The company must also use encryption keys that only specific employees can access.
Which solution will meet these requirements with the LEAST effort?

A.Use an AWS CloudHSM cluster to store the encryption keys. Configure the process that writes to Amazon S3 to make calls to CloudHSM to encrypt and decrypt the objects. Deploy an IAM policy that restricts access to the CloudHSM cluster.

B.Use server-side encryption with customer-provided keys (SSE-C) to encrypt the objects that contain customer information. Restrict access to the keys that encrypt the objects.

C.Use server-side encryption with AWS KMS keys (SSE-KMS) to encrypt the objects that contain customer information. Configure an IAM policy that restricts access to the KMS keys that encrypt the objects.

D.Use server-side encryption with Amazon S3 managed keys (SSE-S3) to encrypt the objects that contain customer information. Configure an IAM policy that restricts access to the Amazon S3 managed keys that encrypt the objects.

Answer: C

Explanation:

The correct answer is C: "Use server-side encryption with AWS KMS keys (SSE-KMS) to encrypt the objects that contain customer information. Configure an IAM policy that restricts access to the KMS keys that encrypt the objects."

Here's why: The question emphasizes ease of implementation and controlled access to encryption keys. SSE-KMS is the most straightforward approach to achieving both. SSE-KMS integrates seamlessly with S3 and provides fine-grained control over who can use the encryption keys through IAM policies. This means you can easily grant access to the keys only to specific employees as required. When data is uploaded, S3 uses the KMS key to encrypt the object server-side. No code changes are required in the application writing to S3.

Option A, using CloudHSM, involves significantly more complexity. CloudHSM requires managing a dedicated hardware security module cluster. This includes tasks such as provisioning, patching, and key management within the HSM. The application writing to S3 needs to be modified to directly interact with CloudHSM for encryption, increasing development and maintenance overhead.

Option B, SSE-C, places the responsibility of key management entirely on the user. While it allows restricting access to keys, it adds the burden of securely storing, rotating, and providing these keys with every request. S3 doesn't manage the keys at all.

Option D, SSE-S3, offers the least control. Amazon S3 manages the encryption keys, and while IAM can restrict access to the S3 bucket itself, you cannot restrict access to the encryption keys within the service in the same granular way you can with KMS. You cannot specify that only specific employees have access to the encryption keys.

In summary, SSE-KMS offers the best balance of security, control, and ease of implementation for encrypting S3 objects with restricted key access. It allows you to easily control which employees can use the keys without needing to manage the key infrastructure, making it the least effort solution.

Relevant Links:

AWS KMS Encryption:
Protecting Data Using Server-Side Encryption:
SSE-KMS:
IAM Policies:

Question: 70 CertyIQ

A company stores petabytes of data in thousands of Amazon S3 buckets in the S3 Standard storage class. The data supports analytics workloads that have unpredictable and variable data access patterns.
The company does not access some data for months. However, the company must be able to retrieve all data within milliseconds. The company needs to optimize S3 storage costs.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use S3 Storage Lens standard metrics to determine when to move objects to more cost-optimized storage classes. Create S3 Lifecycle policies for the S3 buckets to move objects to cost-optimized storage classes. Continue to refine the S3 Lifecycle policies in the future to optimize storage costs.

B.Use S3 Storage Lens activity metrics to identify S3 buckets that the company accesses infrequently. Configure S3 Lifecycle rules to move objects from S3 Standard to the S3 Standard-Infrequent Access (S3 Standard-IA) and S3 Glacier storage classes based on the age of the data.

C.Use S3 Intelligent-Tiering. Activate the Deep Archive Access tier.

D.Use S3 Intelligent-Tiering. Use the default access tier.

Answer: D

Explanation:

The correct answer is D: Use S3 Intelligent-Tiering. Use the default access tier. Here's why:

Requirement Match: The problem states the company needs millisecond retrieval times and has variable, unpredictable access patterns with some data being infrequently accessed. S3 Intelligent-Tiering is designed precisely for this scenario. It automatically moves data between frequent, infrequent, and archive access tiers based on access patterns, without any operational overhead.

Cost Optimization: S3 Intelligent-Tiering optimizes storage costs by automatically moving infrequently accessed data to lower-cost tiers like the Infrequent Access tier and the Archive Access tier. This helps reduce overall storage expenses compared to keeping all data in S3 Standard.

Low Operational Overhead: S3 Intelligent-Tiering requires minimal configuration. Once enabled on a bucket or object, it manages tiering automatically. This eliminates the need for manually creating and refining S3 Lifecycle policies, as suggested in option A.

Millisecond Retrieval: Regardless of the tier an object is in within S3 Intelligent-Tiering, the retrieval time remains the same (milliseconds). This satisfies the stringent retrieval time requirement.

Why other options are incorrect:

Option A: While using S3 Storage Lens and S3 Lifecycle policies can optimize costs, it introduces significant operational overhead. Continuously monitoring metrics and refining policies is time-consuming and error-prone. It doesn't directly leverage the automated intelligence of S3 Intelligent-Tiering. Also, using S3 Glacier, even through lifecycle policies, would violate the millisecond retrieval requirement.

Option B: Similar to option A, this involves manual configuration of S3 Lifecycle rules and does not leverage the automated advantages of Intelligent-Tiering. Also, using S3 Glacier violates the millisecond retrieval time requirement.
Option C: Activating the Deep Archive Access tier would introduce longer retrieval times (hours), violating the millisecond retrieval requirement.

Therefore, S3 Intelligent-Tiering with the default access tier offers the best balance of cost optimization, millisecond retrieval times, and minimal operational overhead for the given scenario.

Authoritative Links:

S3 Intelligent-Tiering: https://aws.amazon.com/s3/storage-classes/intelligent-tiering/
S3 Storage Classes: https://aws.amazon.com/s3/storage-classes/

Question: 71 CertyIQ

During a security review, a company identified a vulnerability in an AWS Glue job. The company discovered that credentials to access an Amazon Redshift cluster were hard coded in the job script.
A data engineer must remediate the security vulnerability in the AWS Glue job. The solution must securely store the credentials.
Which combination of steps should the data engineer take to meet these requirements? (Choose two.)

A.Store the credentials in the AWS Glue job parameters.
B.Store the credentials in a configuration file that is in an Amazon S3 bucket.
C.Access the credentials from a configuration file that is in an Amazon S3 bucket by using the AWS Glue job.
D.Store the credentials in AWS Secrets Manager.
E.Grant the AWS Glue job IAM role access to the stored credentials.

Answer: DE

Explanation:

The vulnerability lies in storing database credentials directly within the AWS Glue job script. This is a major security risk as it exposes sensitive information. Options A, B and C are not secure practices. Option A stores credentials in Glue Job parameters, which are often accessible and not designed for secure storage. Option B and C store credentials in an S3 bucket, which, while potentially more obfuscated than hardcoding, are still vulnerable if the bucket permissions are misconfigured or if the file is inadvertently exposed.

Option D, storing the credentials in AWS Secrets Manager, is a best practice for securely managing secrets. Secrets Manager is a dedicated service designed to store, rotate, and manage sensitive information like database credentials, API keys, and passwords.

Option E, granting the AWS Glue job IAM role access to the stored credentials in Secrets Manager, is crucial for enabling the Glue job to retrieve the credentials securely. The IAM role provides the Glue job with the necessary permissions to call the Secrets Manager API and access the specific secret containing the credentials. Without this IAM permission, the Glue job would be unable to retrieve the credentials, even if they are stored securely in Secrets Manager.

Therefore, storing the credentials in AWS Secrets Manager (Option D) and granting the AWS Glue job IAM role access to those credentials (Option E) provides a secure and auditable way to manage and access sensitive information, mitigating the original vulnerability of hardcoding credentials. This approach aligns with the principle of least privilege and reduces the risk of credential compromise.

Relevant Links:

AWS Secrets Manager: https://aws.amazon.com/secrets-manager/
IAM Roles: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
AWS Glue Security: https://docs.aws.amazon.com/glue/latest/dg/security-iam.html

Question: 72 CertyIQ

A data engineer uses Amazon Redshift to run resource-intensive analytics processes once every month. Every month, the data engineer creates a new Redshift provisioned cluster. The data engineer deletes the Redshift provisioned cluster after the analytics processes are complete every month. Before the data engineer deletes the cluster each month, the data engineer unloads backup data from the cluster to an Amazon S3 bucket.
The data engineer needs a solution to run the monthly analytics processes that does not require the data engineer to manage the infrastructure manually.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon Step Functions to pause the Redshift cluster when the analytics processes are complete and to resume the cluster to run new processes every month.

B.Use Amazon Redshift Serverless to automatically process the analytics workload.

C.Use the AWS CLI to automatically process the analytics workload.

D.Use AWS CloudFormation templates to automatically process the analytics workload.

Answer: B

Explanation:

The correct answer is B: Use Amazon Redshift Serverless to automatically process the analytics workload.

Here's why:

Redshift Serverless Benefits: Redshift Serverless eliminates the need for manual infrastructure management. It automatically provisions and scales compute and data warehouse capacity to deliver fast performance for demanding analytics workloads. This aligns perfectly with the requirement of minimizing operational overhead. https://aws.amazon.com/redshift/serverless/

Eliminates Cluster Management: Because Redshift Serverless is, well, serverless, the data engineer doesn't have to worry about creating, deleting, pausing, or resuming clusters. This simplifies the monthly process significantly, reducing manual intervention.

Cost Optimization: Redshift Serverless automatically scales resources based on demand, so you only pay for what you use. The described workload only requires monthly processing, so Redshift Serverless would be highly efficient and cost-effective, as it would only consume resources during that period.

Why other options are less suitable:

A (Step Functions with Provisioned Redshift): While Step Functions can automate the process of pausing and resuming a Redshift cluster, it still requires the data engineer to provision and manage the cluster initially, which increases operational overhead compared to Redshift Serverless.

C (AWS CLI): The AWS CLI is a powerful tool for managing AWS resources, but it does not, by itself, provide a serverless solution for running analytics. It would require scripting and orchestration to create, manage, and shut down a provisioned Redshift cluster.

D (CloudFormation): CloudFormation templates can automate infrastructure provisioning, but they still require the data engineer to define and manage the resources. It does not inherently eliminate infrastructure management in the same way as a serverless service.

In summary, Redshift Serverless provides the simplest and most automated solution for running monthly analytics workloads without requiring manual infrastructure management, thus meeting the requirement of

least operational overhead.

Question: 73 CertyIQ

A company receives a daily file that contains customer data in .xls format. The company stores the file in Amazon S3. The daily file is approximately 2 GB in size.
A data engineer concatenates the column in the file that contains customer first names and the column that contains customer last names. The data engineer needs to determine the number of distinct customers in the file.
Which solution will meet this requirement with the LEAST operational effort?

A.Create and run an Apache Spark job in an AWS Glue notebook. Configure the job to read the S3 file and calculate the number of distinct customers.

B.Create an AWS Glue crawler to create an AWS Glue Data Catalog of the S3 file. Run SQL queries from Amazon Athena to calculate the number of distinct customers.

C.Create and run an Apache Spark job in Amazon EMR Serverless to calculate the number of distinct customers.

D.Use AWS Glue DataBrew to create a recipe that uses the COUNT_DISTINCT aggregate function to calculate the number of distinct customers.

Answer: D

Explanation:

The correct answer is D: Use AWS Glue DataBrew to create a recipe that uses the COUNT_DISTINCT aggregate function to calculate the number of distinct customers. Here's a detailed justification:

The primary goal is to determine the number of distinct customers from a daily 2GB .xls file stored in S3 with minimal operational overhead. Several options are available, but their efficiency and complexity vary.

Option A, using AWS Glue notebook with Apache Spark, involves writing and managing Spark code for reading the .xls file, concatenating columns, and counting distinct values. While feasible, it requires more development and operational effort for scripting, job execution, and potential error handling. https://aws.amazon.com/glue/features/

Option B, utilizing AWS Glue crawler and Amazon Athena, also presents some complexity. AWS Glue crawler needs to be configured to infer the schema of the .xls file (which might be challenging), and Athena will then query this inferred schema. While SQL-based, it might require knowledge of complex queries and optimal performance tuning for large files. https://aws.amazon.com/athena/

Option C, employing Amazon EMR Serverless with Apache Spark, is similar to option A. EMR Serverless simplifies the management of the EMR cluster. The core challenge of writing and maintaining Spark code for the .xls processing and distinct count calculation remains. https://aws.amazon.com/emr/serverless/

Option D, using AWS Glue DataBrew, is the most streamlined approach. AWS Glue DataBrew is a visual data preparation tool that requires minimal coding. It directly provides functionalities to read data from S3, transform the data using built-in recipes (concatenation in this case), and apply aggregate functions like COUNT_DISTINCT to determine the unique customer count. The visual interface and pre-built transformations reduce development time and operational complexity significantly. https://aws.amazon.com/glue/databrew/

Because the .xls format can be read using DataBrew and DataBrew has built-in functions for both concatenating the columns and providing a count of distinct values, this option involves the least development and operational effort. It provides a quick, efficient, and easily manageable solution for the given task.

Question: 74 CertyIQ

A healthcare company uses Amazon Kinesis Data Streams to stream real-time health data from wearable devices, hospital equipment, and patient records.
A data engineer needs to find a solution to process the streaming data. The data engineer needs to store the data in an Amazon Redshift Serverless warehouse. The solution must support near real-time analytics of the streaming data and the previous day's data.
Which solution will meet these requirements with the LEAST operational overhead?

A.Load data into Amazon Kinesis Data Firehose. Load the data into Amazon Redshift.

B.Use the streaming ingestion feature of Amazon Redshift.

C.Load the data into Amazon S3. Use the COPY command to load the data into Amazon Redshift.

D.Use the Amazon Aurora zero-ETL integration with Amazon Redshift.

Answer: B

Explanation:

The correct answer is B: Use the streaming ingestion feature of Amazon Redshift.

Here's why:

Amazon Redshift's streaming ingestion directly addresses the need for near real-time analytics of streaming data. It allows data to be ingested directly from Kinesis Data Streams into Redshift tables. This minimizes latency because it bypasses intermediary storage like S3. Since Redshift Serverless is used, operational overhead is already minimized as resource management is handled by AWS, and this solution further simplifies the data pipeline. The streaming ingestion feature builds on this by reducing the need for separate ETL pipelines, reducing management complexity.

Option A, using Kinesis Data Firehose and loading into Redshift, introduces an unnecessary intermediary step. While Firehose can load data into Redshift, it's generally used for batch-oriented or micro-batch scenarios, adding latency compared to Redshift's native streaming ingestion.

Option C, loading data into Amazon S3 first and then using the COPY command, is also a batch-oriented approach. COPY command requires data to be staged in S3 before it can be loaded into Redshift. This adds significant latency and is not suitable for near real-time analytics. This is much more applicable to a batch processing or traditional ETL pipeline.

Option D, the Aurora zero-ETL integration with Redshift, is not applicable here. This option is specifically designed to replicate data from an Aurora database to Redshift, and not from a Kinesis Data Stream. The use case is database replication, not streaming data processing.

By directly ingesting data from Kinesis into Redshift using the native streaming ingestion feature, the healthcare company achieves the lowest latency, supports near real-time analytics, leverages the managed nature of Redshift Serverless to minimize operational overhead, and simplifies the overall data pipeline compared to the alternatives.

For further research, refer to the official AWS documentation:

Amazon Redshift Streaming Ingestion: https://aws.amazon.com/redshift/features/streaming-ingestion/
Amazon Redshift Serverless: https://aws.amazon.com/redshift/serverless/

Question: 75 CertyIQ

A data engineer needs to use an Amazon QuickSight dashboard that is based on Amazon Athena queries on data that is stored in an Amazon S3 bucket. When the data engineer connects to the QuickSight dashboard, the data engineer receives an error message that indicates insufficient permissions.

Which factors could cause to the permissions-related errors? (Choose two.)

A.There is no connection between QuickSight and Athena.
B.The Athena tables are not cataloged.
C.QuickSight does not have access to the S3 bucket.
D.QuickSight does not have access to decrypt S3 data.
E.There is no IAM role assigned to QuickSight.

Answer: CD

Explanation:

The correct answer is C and D. Let's break down why:

C: QuickSight does not have access to the S3 bucket.

QuickSight needs explicit permission to read the data residing in the S3 bucket. Athena queries might run successfully because Athena itself might have the necessary permissions to access S3. However, QuickSight directly accesses the data to visualize it.
If QuickSight's IAM role or policy doesn't grant s3:GetObject permission on the S3 bucket and the specific objects (data files), QuickSight will fail to retrieve the data and display an error. This is a fundamental aspect of AWS security – resources only have access if explicitly granted.
Without this access, QuickSight cannot visualize the data returned by Athena. This is a common cause of permissions-related errors when using QuickSight with S3-backed data.

D: QuickSight does not have access to decrypt S3 data.

If the data in the S3 bucket is encrypted (either using server-side encryption with S3 managed keys (SSE-S3), server-side encryption with KMS managed keys (SSE-KMS), or client-side encryption), QuickSight needs permission to decrypt the data.
For SSE-KMS, QuickSight's IAM role must have permission to use the KMS key (kms:Decrypt).
If encryption is in place and QuickSight lacks the necessary decryption permissions, it will fail to read the data, leading to permissions-related errors. This is crucial for data security and ensuring only authorized services can access sensitive information.
Server-side encryption with customer-provided keys (SSE-C) is also a potential scenario, where QuickSight would need to be configured with those keys.

Why other options are incorrect:

A: There is no connection between QuickSight and Athena. A connection needs to be established between QuickSight and Athena as a data source, but this not being set would cause a different error.
B: The Athena tables are not cataloged. If Athena tables are not cataloged, Athena itself would have problems querying the data, before QuickSight could even access it.
E: There is no IAM role assigned to QuickSight. QuickSight does have an IAM role that gets created when setting it up.

Supporting Documentation:

Connecting to Amazon S3: https://docs.aws.amazon.com/quicksight/latest/user/supported-data-sources.html
IAM Role for QuickSight: https://docs.aws.amazon.com/quicksight/latest/user/security-iam-service-role-permissions.html
Amazon S3 Encryption: https://docs.aws.amazon.com/AmazonS3/latest/userguide/serv-side-encryption.html

Question: 76 CertyIQ

A company stores datasets in JSON format and .csv format in an Amazon S3 bucket. The company has Amazon RDS for Microsoft SQL Server databases, Amazon DynamoDB tables that are in provisioned capacity mode, and an Amazon Redshift cluster. A data engineering team must develop a solution that will give data scientists the ability to query all data sources by using syntax similar to SQL.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use AWS Glue to crawl the data sources. Store metadata in the AWS Glue Data Catalog. Use Amazon Athena to query the data. Use SQL for structured data sources. Use PartiQL for data that is stored in JSON format.

B.Use AWS Glue to crawl the data sources. Store metadata in the AWS Glue Data Catalog. Use Redshift Spectrum to query the data. Use SQL for structured data sources. Use PartiQL for data that is stored in JSON format.

C.Use AWS Glue to crawl the data sources. Store metadata in the AWS Glue Data Catalog. Use AWS Glue jobs to transform data that is in JSON format to Apache Parquet or .csv format. Store the transformed data in an S3 bucket. Use Amazon Athena to query the original and transformed data from the S3 bucket.

D.Use AWS Lake Formation to create a data lake. Use Lake Formation jobs to transform the data from all data sources to Apache Parquet format. Store the transformed data in an S3 bucket. Use Amazon Athena or Redshift Spectrum to query the data.

Answer: A

Explanation:

The best solution is A because it leverages Amazon Athena, which is designed for querying data in place in S3 using SQL-like syntax, with the added benefit of PartiQL support for querying JSON data directly. This avoids unnecessary data movement or transformation, minimizing operational overhead. AWS Glue handles the crawling and metadata management, providing a unified catalog.

Here's a detailed breakdown:

Athena's Suitability: Athena's serverless nature means no infrastructure management is required. It's ideal for ad-hoc querying by data scientists. Its pay-per-query pricing model aligns with on-demand analytical needs.

PartiQL for JSON: PartiQL provides a SQL-compatible query language to efficiently access and manipulate data in various formats, including JSON, simplifying the interaction with semi-structured data stored in S3.

AWS Glue Integration: Glue simplifies the process of discovering data schemas and registering tables in the Glue Data Catalog, which Athena uses. This eliminates the manual effort of defining table structures.

Why other options are less optimal:

B (Redshift Spectrum): Redshift Spectrum, while powerful for querying S3 data, is better suited for larger data volumes and complex analytical workloads where Redshift's compute power is justified. It is generally overkill for the described scenario and increases operational complexity.

C (Glue transformation to Parquet): Transforming all data to Parquet introduces ETL processes, adding to operational overhead. The requirement mentions querying "original and transformed data," implying a need for both.

D (Lake Formation): Lake Formation simplifies creating and managing a data lake. However, transforming all data to Parquet using Lake Formation jobs before querying is unnecessarily complex when a serverless query engine can directly query both the JSON and CSV data.

Authoritative Links: * Amazon Athena * AWS Glue * PartiQL * Amazon Redshift Spectrum * AWS Lake Formation

Question: 77 CertyIQ

A data engineer is configuring Amazon SageMaker Studio to use AWS Glue interactive sessions to prepare data for machine learning (ML) models.
The data engineer receives an access denied error when the data engineer tries to prepare the data by using SageMaker Studio.
Which change should the engineer make to gain access to SageMaker Studio?

A.Add the AWSGlueServiceRole managed policy to the data engineer's IAM user.

B.Add a policy to the data engineer's IAM user that includes the sts:AssumeRole action for the AWS Glue and SageMaker service principals in the trust policy.

C.Add the AmazonSageMakerFullAccess managed policy to the data engineer's IAM user.

D.Add a policy to the data engineer's IAM user that allows the sts:AddAssociation action for the AWS Glue and SageMaker service principals in the trust policy.

Answer: B

Explanation:

The correct answer is **B. Add a policy to the data engineer's IAM user that includes the sts:AssumeRole action for the AWS Glue and SageMaker service principals in the trust policy.**

Here's why:

The problem indicates an access denied error when the data engineer tries to use SageMaker Studio with AWS Glue interactive sessions. This strongly suggests a permissions issue related to assuming roles. SageMaker Studio, when interacting with other AWS services like Glue, often needs to assume a role to gain the necessary permissions. Glue interactive sessions need to access the data the data engineer is trying to prepare. Therefore, the data engineer's IAM user needs explicit permission to assume the roles that SageMaker and Glue use for these interactions.

sts:AssumeRole is the IAM action that grants the ability to assume a role. The policy needs to specify that the data engineer's user can assume roles that have a trust policy allowing the SageMaker and Glue service principals (sagemaker.amazonaws.com and glue.amazonaws.com) to assume them. The trust policy on the role assumed by SageMaker/Glue defines who can assume it, whereas the user's policy defines which roles the user is allowed to assume. Without this, the data engineer's user is effectively blocked from authorizing SageMaker and Glue to do work on their behalf.

Option A, adding AWSGlueServiceRole, might seem relevant, but this role is for Glue, not for the user. While it's required for Glue itself to function, granting it to the engineer wouldn't give them permission to use Glue through SageMaker. Option C, AmazonSageMakerFullAccess, is overly permissive. It grants a wide range of SageMaker permissions that aren't necessarily needed and violates the principle of least privilege. Option D, sts:AddAssociation, is not a standard STS action and wouldn't be relevant to role assumption. The operation sts:AddAssociation is not related to role assumption or granting permissions for using AWS Glue interactive sessions through Amazon SageMaker Studio. The correct approach involves enabling the data engineer's IAM user to assume the roles that are used by the SageMaker Studio and AWS Glue services.

In summary, allowing the data engineer's IAM user to AssumeRole for SageMaker and Glue service principals ensures the data engineer's IAM user can authorize those services to access the data.

Refer to these resources for further information:

IAM Roles
Granting Permissions to Assume a Role
AWS STS AssumeRole

Question: 78 CertyIQ

A company extracts approximately 1 TB of data every day from data sources such as SAP HANA, Microsoft SQL Server, MongoDB, Apache Kafka, and Amazon DynamoDB. Some of the data sources have undefined data schemas or data schemas that change.
A data engineer must implement a solution that can detect the schema for these data sources. The solution must extract, transform, and load the data to an Amazon S3 bucket. The company has a service level agreement (SLA) to load the data into the S3 bucket within 15 minutes of data creation.
Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon EMR to detect the schema and to extract, transform, and load the data into the S3 bucket. Create a pipeline in Apache Spark.

B.Use AWS Glue to detect the schema and to extract, transform, and load the data into the S3 bucket. Create a pipeline in Apache Spark.

C.Create a PySpark program in AWS Lambda to extract, transform, and load the data into the S3 bucket.

D.Create a stored procedure in Amazon Redshift to detect the schema and to extract, transform, and load the data into a Redshift Spectrum table. Access the table from Amazon S3.

Answer: B

Explanation:

The correct answer is B: Use AWS Glue to detect the schema and to extract, transform, and load the data into the S3 bucket. Create a pipeline in Apache Spark.

Here's a detailed justification:

Schema Detection: AWS Glue provides automatic schema discovery capabilities through its Glue Data Catalog and Crawlers. This is crucial because the data sources have undefined or changing schemas. Glue Crawlers can infer schemas from various data sources, including those mentioned in the scenario (SAP HANA, SQL Server, MongoDB, Kafka, DynamoDB).
ETL Capability: AWS Glue is specifically designed for ETL (Extract, Transform, Load) operations. It offers a managed Apache Spark environment that simplifies data transformation tasks.
Integration with S3: Glue seamlessly integrates with Amazon S3 as both a source and destination for data.
Operational Overhead: AWS Glue is a serverless service, minimizing operational overhead. You don't have to manage the underlying infrastructure. Glue manages the Spark environment, scaling, and maintenance, reducing manual effort.
Scalability and Performance: Glue can handle large datasets and complex transformations due to its managed Spark environment, meeting the requirement of processing 1 TB of data daily.
SLA Compliance: Glue's scalability and built-in ETL features allow for timely data processing, enabling the company to meet its 15-minute SLA for loading data into S3.
Comparison with Alternatives:

A (Amazon EMR): While EMR can perform the same tasks, it requires more operational overhead. Managing an EMR cluster, including cluster sizing, configuration, and monitoring, is more complex than using Glue.
C (AWS Lambda): Lambda functions have execution time limits and memory constraints, making them unsuitable for processing 1 TB of data. Also, writing a robust PySpark program within Lambda for schema detection and ETL would be complex and impractical.
D (Amazon Redshift): Redshift is primarily a data warehouse and not ideal for detecting schemas of diverse, rapidly changing data sources directly. Redshift Spectrum relies on pre-defined schemas, defeating the purpose of dynamic schema detection. Also, loading 1 TB into a Redshift Spectrum table and then accessing from S3 would involve unnecessary data transfer and computational cost.

In summary, AWS Glue offers a managed, serverless ETL service that is ideally suited for schema detection and data transformation from diverse sources to S3, fulfilling all the given requirements with the least

operational overhead.

Authoritative Links:

AWS Glue Documentation: https://docs.aws.amazon.com/glue/index.html
AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
Amazon S3 Documentation: https://docs.aws.amazon.com/s3/index.html

Question: 79 CertyIQ

A company has multiple applications that use datasets that are stored in an Amazon S3 bucket. The company has an ecommerce application that generates a dataset that contains personally identifiable information (PII). The company has an internal analytics application that does not require access to the PII.
To comply with regulations, the company must not share PII unnecessarily. A data engineer needs to implement a solution that with redact PII dynamically, based on the needs of each application that accesses the dataset.
Which solution will meet the requirements with the LEAST operational overhead?

A.Create an S3 bucket policy to limit the access each application has. Create multiple copies of the dataset. Give each dataset copy the appropriate level of redaction for the needs of the application that accesses the copy.

B.Create an S3 Object Lambda endpoint. Use the S3 Object Lambda endpoint to read data from the S3 bucket. Implement redaction logic within an S3 Object Lambda function to dynamically redact PII based on the needs of each application that accesses the data.

C.Use AWS Glue to transform the data for each application. Create multiple copies of the dataset. Give each dataset copy the appropriate level of redaction for the needs of the application that accesses the copy.

D.Create an API Gateway endpoint that has custom authorizers. Use the API Gateway endpoint to read data from the S3 bucket. Initiate a REST API call to dynamically redact PII based on the needs of each application that accesses the data.

Answer: B

Explanation:

The correct answer is **B: Create an S3 Object Lambda endpoint. Use the S3 Object Lambda endpoint to read data from the S3 bucket. Implement redaction logic within an S3 Object Lambda function to dynamically redact PII based on the needs of each application that accesses the data.**

Here's a detailed justification:

S3 Object Lambda is designed for transforming data retrieved from S3 on-the-fly, before it's returned to the application. This makes it ideal for dynamic PII redaction. With S3 Object Lambda, a Lambda function intercepts requests for objects stored in S3. This function can then modify the data (in this case, redact PII) based on application-specific logic before returning it to the requesting application. The key advantage is that only one copy of the dataset is maintained in S3, reducing storage and management overhead.

Option A, creating multiple copies of the dataset with varying levels of redaction, introduces significant operational overhead. Maintaining multiple copies requires more storage, increases the risk of inconsistencies between datasets, and complicates data governance. The S3 bucket policy can control access, but it doesn't dynamically redact data based on the specific application needs without creating multiple copies.

Option C, using AWS Glue, is more suitable for batch processing and ETL (Extract, Transform, Load) workloads. While Glue can perform data transformations, it is not the best choice for dynamic, real-time redaction during data retrieval. Creating multiple transformed copies with Glue suffers from the same drawbacks as option A.

Option D, using API Gateway with custom authorizers, adds unnecessary complexity. API Gateway is primarily designed for managing APIs, not for data transformation. While it could theoretically be used to trigger a

redaction process, it's a less efficient and more complex solution compared to S3 Object Lambda, which is specifically built for this type of use case. It will involve another REST API call and adds latency.

S3 Object Lambda offers the least operational overhead because it allows for dynamic, on-demand data transformation without replicating the dataset. This ensures that each application receives the appropriate version of the data with PII redacted as needed, all while managing a single source of truth. It directly integrates with S3 and utilizes serverless compute (Lambda) for the transformation logic, making it a scalable and cost-effective solution.

Here are authoritative links for further research:

AWS S3 Object Lambda: https://aws.amazon.com/s3/object-lambda/
S3 Object Lambda Documentation: https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-object-lambda.html

Question: 80 CertyIQ

A data engineer needs to build an extract, transform, and load (ETL) job. The ETL job will process daily incoming .csv files that users upload to an Amazon S3 bucket. The size of each S3 object is less than 100 MB.
Which solution will meet these requirements MOST cost-effectively?

A.Write a custom Python application. Host the application on an Amazon Elastic Kubernetes Service (Amazon EKS) cluster.

B.Write a PySpark ETL script. Host the script on an Amazon EMR cluster.

C.Write an AWS Glue PySpark job. Use Apache Spark to transform the data.

D.Write an AWS Glue Python shell job. Use pandas to transform the data.

Answer: C

Explanation:

The correct answer is **C. Write an AWS Glue PySpark job. Use Apache Spark to transform the data.**

Here's why:

AWS Glue is purpose-built for ETL: AWS Glue is a fully managed ETL service, designed to discover, prepare, and integrate data for analytics, machine learning, and application development. This eliminates the need for setting up and managing infrastructure specifically for ETL. (https://aws.amazon.com/glue/)

Cost-effectiveness: Glue is cost-effective because it's a pay-as-you-go service. You only pay for the time your ETL job runs. For this use case with relatively small daily files, the cost associated with AWS Glue will likely be lower than maintaining a dedicated EKS cluster or EMR cluster.

Spark for Scalability: Using a Glue PySpark job leverages the distributed processing capabilities of Apache Spark, enabling efficient processing of data even if the file sizes increase in the future. While the individual files are currently small, Spark provides scalability if the data volume grows.

Simplicity: Glue automates many ETL tasks like data discovery, schema inference, and job scheduling. This reduces the development and operational overhead.

Option A (EKS): Using EKS involves managing a Kubernetes cluster, which adds significant operational overhead. The cost of running a Kubernetes cluster can be substantial, even for a small cluster, making it less cost-effective for simple ETL tasks.

Option B (EMR): Amazon EMR is also a good choice for ETL and Big Data applications; however, setting up and maintaining an EMR cluster has a higher overhead and cost than AWS Glue, particularly when you need a

service that scales with demand.

Option D (Glue Python Shell with Pandas): While AWS Glue Python Shell is a cheaper option, it does not provide as much functionality as other solutions. Python Shells are better for simple ETL jobs, but not as effective when dealing with potentially large datasets that would benefit from Apache Spark. Additionally, using pandas might become a bottleneck when file sizes increases.

In Summary, **AWS Glue PySpark** provides the best balance of cost-effectiveness, scalability, and simplicity for this particular ETL use case, making it the most suitable solution.
(https://docs.aws.amazon.com/glue/index.html)

Question: 81 CertyIQ

A data engineer creates an AWS Glue Data Catalog table by using an AWS Glue crawler that is named Orders. The data engineer wants to add the following new partitions:

s3://transactions/orders/order_date=2023-01-01
s3://transactions/orders/order_date=2023-01-02

The data engineer must edit the metadata to include the new partitions in the table without scanning all the folders and files in the location of the table.

Which data definition language (DDL) statement should the data engineer use in Amazon Athena?

A.ALTER TABLE Orders ADD PARTITION(order_date='2023-01-01') LOCATION 's3://transactions/orders/order_date=2023-01-01';
ALTER TABLE Orders ADD PARTITION(order_date='2023-01-02') LOCATION 's3://transactions/orders/order_date=2023-01-02';

B.MSCK REPAIR TABLE Orders;

C.REPAIR TABLE Orders;

D.ALTER TABLE Orders MODIFY PARTITION(order_date='2023-01-01') LOCATION 's3://transactions/orders/2023-01-01';
ALTER TABLE Orders MODIFY PARTITION(order_date='2023-01-02') LOCATION 's3://transactions/orders/2023-01-02';

Answer: A

Explanation:

The correct answer is **A**. Here's why:

The requirement is to add new partitions to an existing AWS Glue Data Catalog table (Orders) without rescanning the entire data location. The most efficient way to achieve this is by directly adding the partition metadata using the ALTER TABLE ADD PARTITION DDL statement in Amazon Athena.

Option A accurately reflects this approach. It uses ALTER TABLE Orders ADD PARTITION to specifically add the new partitions order_date='2023-01-01' and order_date='2023-01-02' to the Orders table. Crucially, it also specifies the LOCATION for each partition, which is the S3 path where the data for that partition resides (s3://transactions/orders/order_date=2023-01-01 and s3://transactions/orders/order_date=2023-01-02). This directly updates the metadata within the Glue Data Catalog without triggering a full scan.

Option B (MSCK REPAIR TABLE Orders) attempts to repair the table metadata, but it requires scanning the S3 location to discover new partitions. This contradicts the requirement of avoiding a full scan. MSCK REPAIR TABLE does not allow for explicit specification of the partition location, it discovers partitions by directory structure.

Option C (REPAIR TABLE Orders) is not a valid Athena DDL statement. It is a command specific to Hive and might not be supported in Athena/Glue contexts for partition management in the same way.

Option D (ALTER TABLE Orders MODIFY PARTITION) is used to modify the location of an existing partition. It does not add new partitions. Since the partitions don't exist yet in the catalog, this command would not function as desired. The partitions need to be added before they can be modified. Furthermore, the provided path s3://transactions/orders/2023-01-01 is not adhering to the standard partition naming convention which is s3://transactions/orders/order_date=2023-01-01.

Therefore, ALTER TABLE ADD PARTITION is the appropriate DDL statement to directly add the partition metadata to the AWS Glue Data Catalog table without performing a full data scan, fulfilling the requirements of the question.

For further reading on Athena partition management, consider the following resource:

AWS Athena - Working with Partitioned Data

Question: 82 CertyIQ

A company stores 10 to 15 TB of uncompressed .csv files in Amazon S3. The company is evaluating Amazon Athena as a one-time query engine.

The company wants to transform the data to optimize query runtime and storage costs.

Which file format and compression solution will meet these requirements for Athena queries?

A..csv format compressed with zip
B.JSON format compressed with bzip2
C.Apache Parquet format compressed with Snappy
D.Apache Avro format compressed with LZO

Answer: C

Explanation:

The correct answer is C: Apache Parquet format compressed with Snappy. Here's why:

Athena benefits significantly from columnar storage formats like Apache Parquet and Apache Avro, as it only reads the columns needed for the query, reducing I/O and improving query performance. Parquet is often favored for its efficient encoding and compression schemes. CSV and JSON are row-oriented, requiring Athena to scan entire rows even if only a few columns are needed, leading to slower query times and increased costs.

Compression is crucial for reducing storage costs and improving I/O performance. Snappy is a fast compression/decompression algorithm that balances compression ratio with speed. Bzip2 and LZO offer higher compression ratios but are generally slower, which can negatively impact query performance. Zip is suitable for archiving but not optimal for querying large datasets in Athena.

Given the requirement for optimizing both query runtime and storage costs, Parquet with Snappy strikes the best balance. Parquet's columnar nature reduces data scanned, and Snappy provides a good trade-off between compression ratio and decompression speed. Avro is also columnar, but Parquet is generally more efficient for analytical workloads. For a one-time query, ease of setup also plays a role, and Parquet on S3 with Snappy is generally straightforward to configure with Athena.

Therefore, Apache Parquet with Snappy is the most suitable choice.

Authoritative Links:

Amazon Athena Best Practices: https://docs.aws.amazon.com/athena/latest/ug/best-practices.html
Apache Parquet: https://parquet.apache.org/
Snappy Compression: https://github.com/google/snappy

Question: 83 CertyIQ

A company uses Apache Airflow to orchestrate the company's current on-premises data pipelines. The company runs SQL data quality check tasks as part of the pipelines. The company wants to migrate the pipelines to AWS and to use AWS managed services.

Which solution will meet these requirements with the LEAST amount of refactoring?

A.Setup AWS Outposts in the AWS Region that is nearest to the location where the company uses Airflow. Migrate the servers into Outposts hosted Amazon EC2 instances. Update the pipelines to interact with the Outposts hosted EC2 instances instead of the on-premises pipelines.

B.Create a custom Amazon Machine Image (AMI) that contains the Airflow application and the code that the company needs to migrate. Use the custom AMI to deploy Amazon EC2 instances. Update the network connections to interact with the newly deployed EC2 instances.

C.Migrate the existing Airflow orchestration configuration into Amazon Managed Workflows for Apache Airflow (Amazon MWAA). Create the data quality checks during the ingestion to validate the data quality by using SQL tasks in Airflow.

D.Convert the pipelines to AWS Step Functions workflows. Recreate the data quality checks in SQL as Python based AWS Lambda functions.

Answer: C

Explanation:

The best solution is C, migrating the existing Airflow configuration to Amazon Managed Workflows for Apache Airflow (Amazon MWAA) and integrating data quality checks using SQL tasks within Airflow. This approach minimizes refactoring because it allows the company to leverage its existing Airflow workflows with minimal changes. Amazon MWAA is a managed service that simplifies the setup, operation, and scaling of Airflow in the cloud, freeing the company from managing the underlying infrastructure. This reduces operational overhead.

Creating data quality checks as SQL tasks in Airflow enables a direct translation of the company's current on-premises SQL checks into the AWS environment. This minimizes code rewriting. Options A and B involve managing EC2 instances or Outposts, which increases management overhead and doesn't fully leverage AWS managed services. Option D requires completely rewriting the SQL data quality checks as Python-based Lambda functions, resulting in significant refactoring effort.

Amazon MWAA provides seamless integration with other AWS services, allowing the company to easily connect to its data sources and destinations. It's also scalable and highly available, ensuring the reliability of the data pipelines. This allows the company to focus on data quality checks. Choosing MWAA retains Airflow, and allows SQL tasks to be re-used.

https://aws.amazon.com/managed-workflows-for-apache-airflow/https://docs.aws.amazon.com/mwaa/latest/userguide/what-is-mwaa.html

Question: 84 CertyIQ

A company uses Amazon EMR as an extract, transform, and load (ETL) pipeline to transform data that comes from

multiple sources. A data engineer must orchestrate the pipeline to maximize performance.

Which AWS service will meet this requirement MOST cost effectively?

A.Amazon EventBridge
B.Amazon Managed Workflows for Apache Airflow (Amazon MWAA)
C.AWS Step Functions
D.AWS Glue Workflows

Answer: C

Explanation:

The correct answer is C, AWS Step Functions. Here's why:

Orchestration Focus: The question explicitly asks for a service to "orchestrate" the ETL pipeline. Orchestration refers to coordinating and managing a complex series of tasks to achieve a larger goal. Step Functions is designed precisely for this purpose, allowing you to define workflows as state machines that manage the execution of individual tasks in a specific order.

Cost-Effectiveness: For orchestration of an ETL pipeline involving EMR jobs, Step Functions often proves more cost-effective than alternatives. Step Functions charges per state transition, meaning you only pay for the actual execution of the steps in your workflow. MWAA, on the other hand, involves running an entire Airflow environment, which can be more expensive if the workflow isn't constantly running.

Integration with AWS Services: Step Functions natively integrates with a wide range of AWS services, including Amazon EMR. This tight integration makes it simple to invoke EMR jobs (e.g., start a Spark job, monitor its status, and move to the next step upon completion) within a Step Functions workflow.

AWS Glue and EventBridge Considerations: While AWS Glue Workflows can orchestrate Glue jobs, the question focuses on orchestrating an EMR pipeline. EventBridge is primarily for event-driven architectures and triggering actions based on events; it's less suitable for defining and managing the sequential steps of a complex ETL pipeline.

MWAA Considerations: Amazon MWAA provides a managed Apache Airflow environment. While Airflow can orchestrate complex workflows, MWAA is generally better suited for more complex data engineering pipelines where you need the flexibility and features that Airflow provides. This often comes with a cost and operational overhead that's not always necessary for simpler ETL pipeline orchestration. Step Functions provides a simpler and often more cost-effective alternative when you specifically need a service for orchestration.

State Management: Step Functions excels at managing the state of a workflow. It keeps track of the current step, inputs, outputs, and any error conditions, making it easier to debug and monitor the ETL pipeline.

Therefore, for orchestrating an EMR-based ETL pipeline while maximizing cost-effectiveness, AWS Step Functions is the most suitable choice.

Further Research:

AWS Step Functions: https://aws.amazon.com/step-functions/
Amazon MWAA: https://aws.amazon.com/managed-workflows-for-apache-airflow/

Question: 85 CertyIQ

An online retail company stores Application Load Balancer (ALB) access logs in an Amazon S3 bucket. The company wants to use Amazon Athena to query the logs to analyze traffic patterns.

A data engineer creates an unpartitioned table in Athena. As the amount of the data gradually increases, the response time for queries also increases. The data engineer wants to improve the query performance in Athena.

Which solution will meet these requirements with the LEAST operational effort?

A.Create an AWS Glue job that determines the schema of all ALB access logs and writes the partition metadata to AWS Glue Data Catalog.

B.Create an AWS Glue crawler that includes a classifier that determines the schema of all ALB access logs and writes the partition metadata to AWS Glue Data Catalog.

C.Create an AWS Lambda function to transform all ALB access logs. Save the results to Amazon S3 in Apache Parquet format. Partition the metadata. Use Athena to query the transformed data.

D.Use Apache Hive to create bucketed tables. Use an AWS Lambda function to transform all ALB access logs.

Answer: B

Explanation:

The correct answer is **B**. Here's why:

The core problem is slow query performance due to the increasing volume of data in the S3 bucket. To address this, we need to partition the data and ensure that the Athena table has awareness of these partitions via the AWS Glue Data Catalog.

Option B is the most efficient way to achieve this with the least operational effort. An AWS Glue crawler automatically infers the schema and creates or updates the table definition in the AWS Glue Data Catalog. Crucially, you can configure the crawler to identify partition keys based on the directory structure of the ALB access logs in S3 (e.g., year/month/day). The built-in classifiers handle common log formats, so no custom coding is needed. Once the crawler is set up, it automatically discovers and adds new partitions as data is added to S3, significantly improving query performance in Athena by only scanning relevant partitions.

Option A is similar to option B, but an AWS Glue job requires manual creation and coding to define the schema detection and partition metadata writing logic, which adds operational overhead compared to the automatic discovery provided by a Glue crawler.

Option C involves transforming the logs to Parquet format and using Lambda, which, while improving query performance with columnar storage and compression, introduces significant operational complexity. It necessitates managing Lambda functions, transformations, and the partition creation process. It also doesn't leverage the Glue Data Catalog as efficiently as option B.

Option D is the most complex solution, involving bucketing with Hive and using Lambda for data transformation. Hive bucketing is a more advanced technique and typically requires greater expertise to implement and maintain. Additionally, using Lambda adds more operational overhead compared to the automated approach of option B.

Therefore, Option B provides the best balance between improving query performance through partitioning and minimizing operational effort by leveraging the automatic schema inference and partition discovery capabilities of the AWS Glue crawler.

Authoritative Links:

AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
AWS Glue Data Catalog: https://docs.aws.amazon.com/glue/latest/dg/datacatalog-overview.html
Amazon Athena Partitioning: https://docs.aws.amazon.com/athena/latest/ug/partitions.html

Question: 86

A company has a business intelligence platform on AWS. The company uses an AWS Storage Gateway Amazon S3 File Gateway to transfer files from the company's on-premises environment to an Amazon S3 bucket.

A data engineer needs to setup a process that will automatically launch an AWS Glue workflow to run a series of AWS Glue jobs when each file transfer finishes successfully.

Which solution will meet these requirements with the LEAST operational overhead?

A.Determine when the file transfers usually finish based on previous successful file transfers. Set up an Amazon EventBridge scheduled event to initiate the AWS Glue jobs at that time of day.

B.Set up an Amazon EventBridge event that initiates the AWS Glue workflow after every successful S3 File Gateway file transfer event.

C.Set up an on-demand AWS Glue workflow so that the data engineer can start the AWS Glue workflow when each file transfer is complete.

D.Set up an AWS Lambda function that will invoke the AWS Glue Workflow. Set up an event for the creation of an S3 object as a trigger for the Lambda function.

Answer: B

Explanation:

The correct answer is B because it leverages event-driven architecture for automated workflow initiation, minimizing operational overhead. Let's break down why:

Event-driven architecture: The core of this solution relies on Amazon EventBridge, a serverless event bus service. S3 File Gateway publishes events upon successful file transfers. EventBridge can then be configured to automatically trigger an AWS Glue workflow in response to these events.

Automation: This approach fully automates the process. The data engineer doesn't need to manually monitor file transfers or initiate the Glue workflow. This reduces the operational burden.

Real-time Processing: Glue workflows are triggered immediately after a successful file transfer.

Now, let's discuss why the other options are not as ideal:

Option A (Scheduled Event): This is suboptimal because file transfer times can vary. A fixed schedule might trigger the Glue workflow before a file transfer is complete or wait unnecessarily long after the transfer is finished. It introduces a time delay which is less effective.

Option C (On-demand Workflow): Requiring manual initiation defeats the purpose of automation. It creates unnecessary operational work for the data engineer.

Option D (Lambda Function): While a Lambda function can indeed invoke the Glue workflow, it adds an extra layer of complexity compared to directly connecting EventBridge to the Glue workflow. The Lambda function's sole purpose would be to relay the event, which EventBridge can handle directly. EventBridge simplifies architecture and requires less maintenance than Lambda.

In conclusion, Option B offers the most efficient and automated approach by directly linking S3 File Gateway events to Glue workflow initiation via EventBridge, ensuring timely and streamlined data processing with minimal operational effort.

Supporting Links:

Amazon EventBridge: https://aws.amazon.com/eventbridge/
AWS Glue Workflows: https://docs.aws.amazon.com/glue/latest/dg/workflows-getting-started.html
Amazon S3 File Gateway: https://aws.amazon.com/storagegateway/file/

Question: 87 CertyIQ

A retail company uses Amazon Aurora PostgreSQL to process and store live transactional data. The company uses an Amazon Redshift cluster for a data warehouse.

An extract, transform, and load (ETL) job runs every morning to update the Redshift cluster with new data from the PostgreSQL database. The company has grown rapidly and needs to cost optimize the Redshift cluster.

A data engineer needs to create a solution to archive historical data. The data engineer must be able to run analytics queries that effectively combine data from live transactional data in PostgreSQL, current data in Redshift, and archived historical data. The solution must keep only the most recent 15 months of data in Amazon Redshift to reduce costs.

Which combination of steps will meet these requirements? (Choose two.)

A.Configure the Amazon Redshift Federated Query feature to query live transactional data that is in the PostgreSQL database.

B.Configure Amazon Redshift Spectrum to query live transactional data that is in the PostgreSQL database.

C.Schedule a monthly job to copy data that is older than 15 months to Amazon S3 by using the UNLOAD command. Delete the old data from the Redshift cluster. Configure Amazon Redshift Spectrum to access historical data in Amazon S3.

D.Schedule a monthly job to copy data that is older than 15 months to Amazon S3 Glacier Flexible Retrieval by using the UNLOAD command. Delete the old data from the Redshift cluster. Configure Redshift Spectrum to access historical data from S3 Glacier Flexible Retrieval.

E.Create a materialized view in Amazon Redshift that combines live, current, and historical data from different sources.

Answer: AC

Explanation:

Let's analyze the proposed solution (AC) and justify why it correctly addresses the requirements.

Option A: Configure the Amazon Redshift Federated Query feature to query live transactional data that is in the PostgreSQL database.

This is a valid approach for directly querying the Aurora PostgreSQL database from Redshift. Federated Query allows Redshift to query data residing in other databases without needing to load it into Redshift. This satisfies the requirement to combine data from the live transactional data in PostgreSQL.

Option C: Schedule a monthly job to copy data that is older than 15 months to Amazon S3 by using the UNLOAD command. Delete the old data from the Redshift cluster. Configure Amazon Redshift Spectrum to access historical data in Amazon S3.

This is also a valid and cost-effective approach for archiving historical data. The UNLOAD command efficiently exports data from Redshift to S3. Deleting the older data after unloading ensures Redshift keeps only the most recent 15 months of data, thus reducing costs. Redshift Spectrum allows querying data directly from S3 without loading it into Redshift, enabling analytics that combine current and historical data.

Why other options are incorrect:

Option B: Redshift Spectrum is for querying data in S3 data lakes, not directly querying PostgreSQL.
Option D: S3 Glacier Flexible Retrieval is optimized for infrequent access and comes with retrieval costs that could be higher if you are running analytics often on the archived data. Redshift Spectrum works with S3 Standard, S3 Standard-IA, and S3 Intelligent-Tiering. Storing archived data in S3 and configuring Redshift Spectrum is a more cost-effective solution.
Option E: Materialized views in Redshift are stored within Redshift itself and will not provide the cost saving for archiving the historical data to S3. While materialized views can speed up query performance, they would duplicate data in Redshift, counteracting the goal of cost optimization.

In summary:

Options A and C together provide a solution that meets all requirements. Federated Query enables querying the live data in PostgreSQL, while UNLOAD and Redshift Spectrum facilitate archiving historical data to S3 and running queries that combine live, current, and archived data. This approach also keeps only the most recent 15 months of data in Redshift, reducing costs.

Supporting Links:

Amazon Redshift Federated Query: https://docs.aws.amazon.com/redshift/latest/dg/federated-query.html
Amazon Redshift Spectrum: https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html
Redshift UNLOAD command: https://docs.aws.amazon.com/redshift/latest/dg/r_UNLOAD.html

Question: 88 CertyIQ

A manufacturing company has many IoT devices in facilities around the world. The company uses Amazon Kinesis Data Streams to collect data from the devices. The data includes device ID, capture date, measurement type, measurement value, and facility ID. The company uses facility ID as the partition key.

The company's operations team recently observed many WriteThroughputExceeded exceptions. The operations team found that some shards were heavily used but other shards were generally idle.

How should the company resolve the issues that the operations team observed?

A.Change the partition key from facility ID to a randomly generated key.
B.Increase the number of shards.
C.Archive the data on the producer's side.
D.Change the partition key from facility ID to capture date.

Answer: A

Explanation:

The issue observed, WriteThroughputExceeded exceptions combined with uneven shard utilization, indicates a "hot partition" problem in Kinesis Data Streams. This happens when a disproportionate amount of data is directed to a small number of shards because of a poorly chosen partition key. Facility ID, while seemingly logical, leads to this problem if some facilities generate significantly more data than others.

Option A, changing the partition key to a randomly generated key, is the most effective solution. A random key distributes data more evenly across all shards, mitigating the "hot partition" effect. By doing this, the write load is spread across all available shards, preventing specific shards from being overwhelmed and triggering WriteThroughputExceeded exceptions. The goal is to achieve even distribution to maximize throughput.

Option B, increasing the number of shards, can provide temporary relief but doesn't address the root cause. While more shards increase overall capacity, the uneven distribution caused by the facility ID partition key remains. The "hot shards" will still be overwhelmed, negating the benefit of the increased shard count.

Option C, archiving data on the producer side, reduces the amount of data being written to Kinesis, but it is not solving the core problem of uneven distribution. Archiving data on the producer side would reduce the overall throughput, which could have a negative impact if the goal is to keep and analyze all the data. It is more appropriate to solve the root cause of the problem (uneven partition) before reducing the amount of data.

Option D, changing the partition key to capture date, is likely to worsen the issue. Capture date may also lead to uneven distribution, especially if data generation is not uniform throughout the day or if specific time periods have significantly higher data volumes. Daily patterns could still lead to certain shards being overloaded.

In summary, a randomly generated key provides the most immediate and effective solution to the hot partition problem by ensuring a more uniform distribution of data across all available Kinesis shards.

Further reading:

AWS Kinesis Data Streams Developer Guide - Choosing a Partition Key:
https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html
AWS Big Data Blog - Tuning Amazon Kinesis Data Streams for optimal performance:
https://aws.amazon.com/blogs/big-data/tuning-amazon-kinesis-data-streams-for-optimal-performance/

Question: 89 CertyIQ

A data engineer wants to improve the performance of SQL queries in Amazon Athena that run against a sales data table.

The data engineer wants to understand the execution plan of a specific SQL statement. The data engineer also wants to see the computational cost of each operation in a SQL query.

Which statement does the data engineer need to run to meet these requirements?

A.EXPLAIN SELECT * FROM sales;
B.EXPLAIN ANALYZE FROM sales;
C.EXPLAIN ANALYZE SELECT * FROM sales;
D.EXPLAIN FROM sales;

Answer: C

Explanation:

The correct answer is C: EXPLAIN ANALYZE SELECT * FROM sales;. Here's why:

The EXPLAIN statement in SQL, including Athena's Presto-based engine, provides information about the query execution plan. It describes the steps the query engine will take to process the query.

Option A, EXPLAIN SELECT * FROM sales;, provides the execution plan, but it doesn't execute the query. Thus, it doesn't provide the computational cost of each operation because it's merely a plan.

Option D, EXPLAIN FROM sales;, is syntactically incorrect. EXPLAIN must be followed by a valid SELECT statement or another query type.

Option B, EXPLAIN ANALYZE FROM sales;, is also syntactically incorrect. EXPLAIN ANALYZE requires a SELECT statement (or other executable query) after it to be valid.

Option C, EXPLAIN ANALYZE SELECT * FROM sales;, is the correct option because it does both: it displays the query execution plan and executes the query. The ANALYZE keyword causes Athena to run the query and collect statistics about each operation (e.g., CPU time, memory usage, data transferred). This allows the data engineer to see the actual computational cost of each step in the query plan, thus fulfilling the requirement to understand both the execution plan and the computational cost of each operation. The output includes the standard execution plan with additional metrics detailing the performance of each stage.

In summary, EXPLAIN ANALYZE gives a data engineer both the execution plan and the actual cost data, which are essential for performance tuning of Athena queries. EXPLAIN alone only provides the plan. This combined information facilitates understanding the query's behavior and identifying areas for optimization (e.g., using appropriate data types, partitioning, or indexing).

Here's an authoritative link for further research:

Amazon Athena documentation on EXPLAIN: https://docs.aws.amazon.com/athena/latest/ug/explain.html (While this specific page doesn't directly discuss EXPLAIN ANALYZE, understanding EXPLAIN is foundational).
Presto documentation on EXPLAIN ANALYZE: As Athena is based on Presto, the Presto documentation gives detail about EXPLAIN ANALYZE: https://prestodb.io/docs/current/sql/explain.html

Question: 90 CertyIQ

A company plans to provision a log delivery stream within a VPC. The company configured the VPC flow logs to publish to Amazon CloudWatch Logs. The company needs to send the flow logs to Splunk in near real time for further analysis.

Which solution will meet these requirements with the LEAST operational overhead?

A.Configure an Amazon Kinesis Data Streams data stream to use Splunk as the destination. Create a CloudWatch Logs subscription filter to send log events to the data stream.

B.Create an Amazon Kinesis Data Firehose delivery stream to use Splunk as the destination. Create a CloudWatch Logs subscription filter to send log events to the delivery stream.

C.Create an Amazon Kinesis Data Firehose delivery stream to use Splunk as the destination. Create an AWS Lambda function to send the flow logs from CloudWatch Logs to the delivery stream.

D.Configure an Amazon Kinesis Data Streams data stream to use Splunk as the destination. Create an AWS Lambda function to send the flow logs from CloudWatch Logs to the data stream.

Answer: B

Explanation:

The correct answer is B: Create an Amazon Kinesis Data Firehose delivery stream to use Splunk as the destination. Create a CloudWatch Logs subscription filter to send log events to the delivery stream.

Here's a detailed justification:

The problem requires a solution that delivers VPC flow logs from CloudWatch Logs to Splunk in near real-time with minimal operational overhead. Both Kinesis Data Streams and Kinesis Data Firehose are suitable for streaming data. However, Kinesis Data Firehose is designed for loading data into data stores and analytics tools like Splunk. Kinesis Data Firehose offers built-in integration with Splunk, simplifying the setup and management of the data ingestion pipeline.

A CloudWatch Logs subscription filter allows you to send log events to other AWS services, including Kinesis Data Firehose. This direct integration removes the need for custom code or intermediary components, further reducing operational overhead. The subscription filter pushes the logs as they arrive, enabling near real-time delivery.

Option A using Kinesis Data Streams, while possible, would require you to manage scaling, partitioning, and potentially data transformation before sending it to Splunk. This increases operational complexity. Splunk is not a direct destination in Kinesis Data Streams without additional configuration.

Options C and D using a Lambda function to forward logs from CloudWatch Logs to either Kinesis Data Firehose or Data Streams introduce unnecessary complexity. While Lambda offers flexibility, it adds operational overhead in terms of function management, error handling, and scaling. The direct integration offered by the CloudWatch Logs subscription filter eliminates the need for this intermediary Lambda function.

Therefore, creating a Kinesis Data Firehose delivery stream with Splunk as the destination and utilizing a CloudWatch Logs subscription filter offers the simplest, most direct, and least operationally intensive solution. It leverages built-in integrations and eliminates the need for custom code or additional services, which aligns with the problem's requirements.

For more information, refer to the following resources:

Amazon Kinesis Data Firehose: https://aws.amazon.com/kinesis/data-firehose/
Amazon CloudWatch Logs Subscription Filters:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html
Kinesis Data Firehose Splunk Integration: https://docs.aws.amazon.com/firehose/latest/dev/splunk.html

Question: 91 CertyIQ

A company has a data lake on AWS. The data lake ingests sources of data from business units. The company uses Amazon Athena for queries. The storage layer is Amazon S3 with an AWS Glue Data Catalog as a metadata repository.

The company wants to make the data available to data scientists and business analysts. However, the company first needs to manage fine-grained, column-level data access for Athena based on the user roles and responsibilities.

Which solution will meet these requirements?

A.Set up AWS Lake Formation. Define security policy-based rules for the users and applications by IAM role in Lake Formation.

B.Define an IAM resource-based policy for AWS Glue tables. Attach the same policy to IAM user groups.

C.Define an IAM identity-based policy for AWS Glue tables. Attach the same policy to IAM roles. Associate the IAM roles with IAM groups that contain the users.

D.Create a resource share in AWS Resource Access Manager (AWS RAM) to grant access to IAM users.

Answer: A

Explanation:

The correct answer is A: Set up AWS Lake Formation. Define security policy-based rules for the users and applications by IAM role in Lake Formation.

Here's why:

AWS Lake Formation is specifically designed to simplify and automate data access management in data lakes. It provides fine-grained access control at the column level, which directly addresses the requirement to manage column-level access for Athena queries based on user roles and responsibilities. Lake Formation uses a centralized permissions model that leverages AWS Identity and Access Management (IAM) roles, allowing administrators to define security policies that govern access to data in the data lake. These policies can specify which users or roles have access to specific columns in tables registered in the AWS Glue Data Catalog. Lake Formation also provides auditing capabilities, making it easier to track data access and ensure compliance with security policies.

Option B is incorrect because while IAM resource-based policies can be used to grant access to AWS Glue tables, managing fine-grained, column-level access becomes cumbersome and difficult to scale across many users and tables. It would require creating and maintaining numerous complex policies, which is not ideal for a centralized data lake access management solution.

Option C is incorrect because IAM identity-based policies are attached to users or roles and control what actions those users or roles can perform. While they can grant access to AWS Glue tables, they don't easily allow for fine-grained, column-level access based on specific data attributes or user responsibilities. You'd still need to manage complex policy combinations.

Option D is incorrect because AWS RAM is primarily for sharing AWS resources across AWS accounts. While useful for cross-account access, it doesn't directly address the need for fine-grained, column-level access

control within a single account. It doesn't provide the centralized, policy-driven access management capabilities of Lake Formation.

In summary, Lake Formation is the most appropriate solution because it is specifically designed for managing fine-grained access control in data lakes, including column-level access, using a centralized, policy-driven approach integrated with IAM and the AWS Glue Data Catalog.

Further Research:

AWS Lake Formation: https://aws.amazon.com/lake-formation/
AWS Lake Formation Permissions Model: https://docs.aws.amazon.com/lake-formation/latest/dg/security-data-access.html
AWS Glue Data Catalog: https://aws.amazon.com/glue/features/

Question: 92 CertyIQ

A company has developed several AWS Glue extract, transform, and load (ETL) jobs to validate and transform data from Amazon S3. The ETL jobs load the data into Amazon RDS for MySQL in batches once every day. The ETL jobs use a DynamicFrame to read the S3 data.

The ETL jobs currently process all the data that is in the S3 bucket. However, the company wants the jobs to process only the daily incremental data.

Which solution will meet this requirement with the LEAST coding effort?

A.Create an ETL job that reads the S3 file status and logs the status in Amazon DynamoDB.

B.Enable job bookmarks for the ETL jobs to update the state after a run to keep track of previously processed data.

C.Enable job metrics for the ETL jobs to help keep track of processed objects in Amazon CloudWatch.

D.Configure the ETL jobs to delete processed objects from Amazon S3 after each run.

Answer: B

Explanation:

The correct answer is **B. Enable job bookmarks for the ETL jobs to update the state after a run to keep track of previously processed data.**

Here's a detailed justification:

The goal is to process only incremental data in S3 using AWS Glue ETL jobs, minimizing coding effort. Job bookmarks in AWS Glue provide a built-in mechanism to track the progress of ETL jobs, remembering the last processed state. When a job runs with bookmarks enabled, Glue stores information about the files it has already processed in the Glue Data Catalog. During subsequent runs, Glue uses this bookmark information to only process new or updated files, effectively achieving incremental data processing.

Option B requires minimal coding because it leverages Glue's existing functionality. Enabling job bookmarks involves a simple configuration change in the Glue job definition. Glue automatically manages the tracking of processed files.

Option A requires custom coding to track file status in DynamoDB, adding complexity. The job needs to manage the DynamoDB table, update it on each run, and query to identify unprocessed files.

Option C, enabling job metrics in CloudWatch, does not directly address the requirement of processing only incremental data. CloudWatch metrics provide insights into job performance but don't inherently track which files have been processed. It would still require additional custom code to achieve the desired functionality.

Option D, deleting processed objects from S3, is generally not recommended. It can lead to data loss if the ETL job fails or if the data is needed for auditing or historical analysis. It also introduces potential risks of data corruption.

Job bookmarks are the most efficient and maintainable solution, reducing coding effort and providing a robust mechanism for incremental data processing in Glue ETL jobs. Glue automatically manages the tracking of processed files, making this the preferred option.

Refer to the official AWS documentation for more information on AWS Glue job bookmarks:

AWS Glue Job Bookmarks

Question: 93 CertyIQ

An online retail company has an application that runs on Amazon EC2 instances that are in a VPC. The company wants to collect flow logs for the VPC and analyze network traffic.

Which solution will meet these requirements MOST cost-effectively?

A.Publish flow logs to Amazon CloudWatch Logs. Use Amazon Athena for analytics.
B.Publish flow logs to Amazon CloudWatch Logs. Use an Amazon OpenSearch Service cluster for analytics.
C.Publish flow logs to Amazon S3 in text format. Use Amazon Athena for analytics.
D.Publish flow logs to Amazon S3 in Apache Parquet format. Use Amazon Athena for analytics.

Answer: D

Explanation:

The correct answer is D, which involves publishing VPC flow logs to Amazon S3 in Apache Parquet format and then using Amazon Athena for analysis. This solution is the most cost-effective for several reasons.

Firstly, publishing flow logs to S3 is generally cheaper than publishing to CloudWatch Logs, especially for high volumes of data. S3 storage costs are lower than CloudWatch Logs ingestion and storage costs.

Secondly, storing flow logs in Apache Parquet format is crucial for cost optimization and query performance. Parquet is a columnar storage format that is highly efficient for analytical queries. Columnar storage means that Athena only reads the columns needed for the query, significantly reducing the amount of data scanned and, consequently, the query cost.

Thirdly, Amazon Athena is a serverless query service that allows you to analyze data directly in S3 using standard SQL. Athena charges only for the queries that you run, making it a cost-effective solution for ad-hoc analysis and exploration of flow logs. No dedicated infrastructure is required, and you avoid the operational overhead of managing a database or data warehouse.

Option A is less optimal because while Athena is a good choice for analytics, CloudWatch Logs are generally more expensive for storing high volumes of flow logs than S3. Option B is incorrect because an OpenSearch Service cluster involves significant operational overhead and ongoing costs, making it overkill for simple flow log analysis. OpenSearch is also generally more expensive than Athena for this purpose. Option C suggests storing flow logs in text format. Text format is less efficient for querying because Athena has to scan the entire file and parse the entire contents, consuming more resources.

In summary, storing flow logs in Parquet format on S3 and analyzing them using Athena provides a cost-effective, scalable, and efficient solution for analyzing VPC flow logs. Parquet format optimizes storage and query costs, while Athena provides a serverless querying experience without requiring extensive infrastructure management.

Relevant links:

VPC Flow Logs: https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html
Amazon S3: https://aws.amazon.com/s3/
Amazon Athena: https://aws.amazon.com/athena/
Apache Parquet: https://parquet.apache.org/

Question: 94 CertyIQ

A retail company stores transactions, store locations, and customer information tables in four reserved ra3.4xlarge Amazon Redshift cluster nodes. All three tables use even table distribution.

The company updates the store location table only once or twice every few years.

A data engineer notices that Redshift queues are slowing down because the whole store location table is constantly being broadcast to all four compute nodes for most queries. The data engineer wants to speed up the query performance by minimizing the broadcasting of the store location table.

Which solution will meet these requirements in the MOST cost-effective way?

A.Change the distribution style of the store location table from EVEN distribution to ALL distribution.

B.Change the distribution style of the store location table to KEY distribution based on the column that has the highest dimension.

C.Add a join column named store_id into the sort key for all the tables.

D.Upgrade the Redshift reserved node to a larger instance size in the same instance family.

Answer: A

Explanation:

The most cost-effective solution to minimize broadcasting of the store location table in Amazon Redshift, given its infrequent updates and impact on query performance, is **A. Change the distribution style of the store location table from EVEN distribution to ALL distribution.**

Here's why:

Understanding the Problem: The EVEN distribution distributes data evenly across all nodes in the cluster. While good for large tables with frequent updates, it necessitates broadcasting small tables like the store location table during joins with other tables. This is because the data needed for the join might reside on different nodes. Broadcasting incurs network overhead and slows down query execution.
Why ALL Distribution is Optimal: ALL distribution replicates the entire store location table on every compute node. Since the table is small and rarely updated, the storage overhead is minimal. This eliminates the need for broadcasting during joins, as each node already possesses the complete table. This significantly speeds up query performance by performing joins locally on each node.
Why other options are less suitable:

B. KEY distribution: KEY distribution distributes rows based on the values in a specified column (the distribution key). Choosing a high-dimension column might not necessarily align the store location data with the other tables based on join criteria, still potentially requiring broadcasting. Furthermore, maintaining an appropriate distribution key requires understanding the query patterns which might change.
C. Adding a join column to the sort key: Sort keys primarily affect the order in which data is stored within each node, improving the efficiency of range-restricted queries. They don't influence how data is distributed across nodes and thus won't prevent broadcasting.
D. Upgrading the Redshift node instance size: This is a more expensive solution. While it might improve overall query performance due to increased resources, it doesn't specifically address the broadcasting issue

of the store location table. The underlying problem of network traffic due to broadcasting remains.

In summary, ALL distribution addresses the root cause of the performance bottleneck, which is broadcasting a small, infrequently updated table. It's a targeted and cost-effective approach compared to the broader, more expensive solution of upgrading the cluster.

Authoritative Links:

Amazon Redshift Distribution Styles:
https://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
Choosing a Distribution Style: https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_a_dist_style.html

Question: 95 CertyIQ

A company has a data warehouse that contains a table that is named Sales. The company stores the table in Amazon Redshift. The table includes a column that is named city_name. The company wants to query the table to find all rows that have a city_name that starts with "San" or "El".

Which SQL query will meet this requirement?

A.Select * from Sales where city_name ~ '$(San|El)*';
B.Select * from Sales where city_name ~ '^(San|El)*';
C.Select * from Sales where city_name ~'$(San&El)*';
D.Select * from Sales where city_name ~ '^(San&El)*';

Answer: B

Explanation:

The correct SQL query to retrieve rows from the Sales table where city_name starts with "San" or "El" is option B: Select *from Sales where city_name ~ '^(SanlEl)'*;. This query utilizes regular expression matching in Amazon Redshift.

Here's a breakdown of why this works and why the other options are incorrect:

~ operator: In Redshift, the ~ operator performs pattern matching using POSIX regular expressions.
^ character: The ^ character within a regular expression signifies the beginning of a string. Therefore, ^(SanlEl) asserts that the string must start with either "San" or "El".
(SanlEl): This is a regular expression construct indicating alternation. It means "either San or El".
character: The character means 'zero or more occurrences' of the preceding character or group. In our case, it means zero or more repetitions of the characters n, a, or S in "San" or l or E in "El". In this case, ^(SanlEl) suffices to find city names that start with "San" or "El" because the city name is by definition more than just those three or two letters.

Let's examine why the other options are incorrect:

Option A: Select *from Sales where city_name ~ '$(SanlEl)'*; The $ symbol means the end of the string. So this query would look for city names that end with 'San' or 'El' or any repetition of their constituting characters.
Option C: Select *from Sales where city_name ~'$(San&El)'*; The & symbol does not have the intended logical OR behavior in regular expressions. It's not a standard regex operator for alternation. This would likely result in no matches or an error.
Option D: Select *from Sales where city_name ~ '^(San&El)'*; Similar to option C, the & is not a valid regex for "or".

In Summary:

Option B correctly uses the ^ anchor to ensure that the match occurs at the beginning of the string, the | operator for alternation (OR), and the ~ operator for regular expression matching in Redshift, making it the only option that fulfills the requirements.

Supporting Documentation:

Amazon Redshift Regular Expression Functions: https://docs.aws.amazon.com/redshift/latest/dg/pattern-matching-conditions-like.html
POSIX Regular Expressions: https://www.postgresql.org/docs/current/functions-matching.html (Redshift's regex engine is based on POSIX)

Question: 96 CertyIQ

A company needs to send customer call data from its on-premises PostgreSQL database to AWS to generate near real-time insights. The solution must capture and load updates from operational data stores that run in the PostgreSQL database. The data changes continuously.

A data engineer configures an AWS Database Migration Service (AWS DMS) ongoing replication task. The task reads changes in near real time from the PostgreSQL source database transaction logs for each table. The task then sends the data to an Amazon Redshift cluster for processing.

The data engineer discovers latency issues during the change data capture (CDC) of the task. The data engineer thinks that the PostgreSQL source database is causing the high latency.

Which solution will confirm that the PostgreSQL database is the source of the high latency?

A.Use Amazon CloudWatch to monitor the DMS task. Examine the CDCIncomingChanges metric to identify delays in the CDC from the source database.

B.Verify that logical replication of the source database is configured in the postgresql.conf configuration file.

C.Enable Amazon CloudWatch Logs for the DMS endpoint of the source database. Check for error messages.

D.Use Amazon CloudWatch to monitor the DMS task. Examine the CDCLatencySource metric to identify delays in the CDC from the source database.

Answer: D

Explanation:

The correct answer is D because it directly addresses the problem of identifying latency at the source database using the appropriate metric within CloudWatch.

Here's a detailed justification:

The problem describes a data pipeline with latency issues during Change Data Capture (CDC) from an on-premises PostgreSQL database to Amazon Redshift using AWS DMS. The objective is to confirm if the PostgreSQL database is the source of the high latency.

Option D, "Use Amazon CloudWatch to monitor the DMS task. Examine the CDCLatencySource metric to identify delays in the CDC from the source database," is the most direct and appropriate solution. AWS DMS emits metrics to CloudWatch that provide insights into the replication process. The CDCLatencySource metric specifically measures the time delay between a change being made in the source database and the moment AWS DMS reads that change from the source's transaction logs. A high CDCLatencySource value would directly confirm that the PostgreSQL database is indeed the source of the latency. This isolates the problem to the source side of the replication process.

Option A, while also using CloudWatch and the DMS task, suggests examining the CDCIncomingChanges metric. This metric represents the number of CDC events received by the DMS task, not the latency at the source. It indicates the volume of changes, not where the delay occurs. A high volume could contribute to

latency elsewhere, but not confirm source latency.

Option B suggests verifying the logical replication configuration in postgresql.conf. While crucial for CDC, simply confirming the configuration doesn't pinpoint latency. Incorrect configuration could cause latency, but the configuration being correct doesn't rule out the PostgreSQL database as a latency source (e.g., resource constraints, lock contention).

Option C proposes enabling CloudWatch Logs for the DMS source endpoint and checking for error messages. While error messages can indicate issues, they aren't specific to measuring latency. The absence of errors does not guarantee the absence of latency issues within the PostgreSQL database (e.g., slow queries, resource limitations). The issue is not necessarily an error but potential slow performance at the source database.Links for further research:

AWS DMS Metrics: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Monitoring.Metrics.html
Monitoring AWS DMS Tasks: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Monitoring.html

Question: 97 CertyIQ

A lab uses IoT sensors to monitor humidity, temperature, and pressure for a project. The sensors send 100 KB of data every 10 seconds. A downstream process will read the data from an Amazon S3 bucket every 30 seconds.

Which solution will deliver the data to the S3 bucket with the LEAST latency?

A.Use Amazon Kinesis Data Streams and Amazon Kinesis Data Firehose to deliver the data to the S3 bucket. Use the default buffer interval for Kinesis Data Firehose.

B.Use Amazon Kinesis Data Streams to deliver the data to the S3 bucket. Configure the stream to use 5 provisioned shards.

C.Use Amazon Kinesis Data Streams and call the Kinesis Client Library to deliver the data to the S3 bucket. Use a 5 second buffer interval from an application.

D.Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) and Amazon Kinesis Data Firehose to deliver the data to the S3 bucket. Use a 5 second buffer interval for Kinesis Data Firehose.

Answer: C

Explanation:

Here's a detailed justification for why option C is the best solution for delivering IoT sensor data to an S3 bucket with the least latency:

The primary goal is to minimize the delay between data generation by the sensors and its availability in S3. Several factors contribute to latency: data buffering, data processing overhead, and network transfer time.

Option A uses Kinesis Data Streams and Kinesis Data Firehose with the default buffer interval. Firehose's default buffer interval is 60 seconds, which is significantly longer than the requirement of accessing data every 30 seconds, creating excessive latency.

Option B only uses Kinesis Data Streams. While Kinesis Data Streams provides real-time data ingestion, it doesn't directly deliver data to S3. A custom application is needed to consume data from the stream and write it to S3. This adds complexity and potential latency from application processing. Simply provisioning shards doesn't directly influence the delivery latency to S3.

Option C utilizes Kinesis Data Streams and Kinesis Client Library (KCL) to stream and deliver the data to the S3 bucket. KCL enables building custom applications to process data from Kinesis Data Streams. By using a 5-second buffer interval in the application, it aggregates data and writes it to S3 much more frequently than Firehose's default interval. This significantly reduces latency while still providing some batching to improve

write efficiency. Since the data is read from the S3 bucket every 30 seconds, a 5-second buffer interval is the best option.

Option D involves Amazon Managed Service for Apache Flink (formerly Kinesis Data Analytics) along with Kinesis Data Firehose. While Flink can process data in real-time, it introduces unnecessary processing overhead for this simple data delivery use case. The overhead of Flink combined with even a 5-second buffer interval in Firehose can still lead to more latency than Option C, which streams data and delivers it with the KCL library.

Therefore, option C offers the best balance between real-time ingestion (Kinesis Data Streams), custom control over the buffering interval (KCL application with a 5-second buffer), and direct data delivery to S3, resulting in the lowest latency. The ability to finely tune the buffer interval in the KCL application provides the necessary control to meet the latency requirements.

Supporting Documentation:

Amazon Kinesis Data Streams: https://aws.amazon.com/kinesis/data-streams/
Amazon Kinesis Data Firehose: https://aws.amazon.com/kinesis/data-firehose/
Kinesis Client Library (KCL): https://docs.aws.amazon.com/streams/latest/dev/developing-consumers-with-kcl.html
Amazon Managed Service for Apache Flink: https://aws.amazon.com/flink/

Question: 98 CertyIQ

A company wants to use machine learning (ML) to perform analytics on data that is in an Amazon S3 data lake. The company has two data transformation requirements that will give consumers within the company the ability to create reports.

The company must perform daily transformations on 300 GB of data that is in a variety format that must arrive in Amazon S3 at a scheduled time. The company must perform one-time transformations of terabytes of archived data that is in the S3 data lake. The company uses Amazon Managed Workflows for Apache Airflow (Amazon MWAA) Directed Acyclic Graphs (DAGs) to orchestrate processing.

Which combination of tasks should the company schedule in the Amazon MWAA DAGs to meet these requirements MOST cost-effectively? (Choose two.)

A.For daily incoming data, use AWS Glue crawlers to scan and identify the schema.
B.For daily incoming data, use Amazon Athena to scan and identify the schema.
C.For daily incoming data, use Amazon Redshift to perform transformations.
D.For daily and archived data, use Amazon EMR to perform data transformations.
E. For archived data, use Amazon SageMaker to perform data transformations.

Answer: AD

Explanation:

The most cost-effective approach combines AWS Glue crawlers for schema discovery and Amazon EMR for data transformations.

A. For daily incoming data, use AWS Glue crawlers to scan and identify the schema: AWS Glue crawlers automatically infer the schema of data stored in S3. This is crucial for the daily 300 GB of data that arrives in various formats. Instead of manually defining schemas, crawlers automate this process, saving time and effort. Glue crawlers are a cost-effective solution for schema discovery, especially when dealing with varying data formats. https://aws.amazon.com/glue/features/

D. For daily and archived data, use Amazon EMR to perform data transformations: Amazon EMR is a

managed cluster platform that lets you run big data frameworks like Apache Spark, Hadoop, and Hive. EMR is well-suited for transforming large datasets, whether daily or archived, because it can scale compute resources as needed. For the daily 300 GB and the terabytes of archived data, EMR provides a flexible and cost-effective way to perform the necessary transformations. The ability to use different instance types in EMR allows for optimizing cost and performance, especially considering the varying sizes of data being processed. EMR also integrates well with S3. https://aws.amazon.com/emr/features/

Why other options are less suitable:

B. Amazon Athena for schema discovery: While Athena can query data with unknown schema using features like dynamic partitioning and schema on read, it's more suited for ad-hoc querying rather than automated, scheduled schema discovery. Glue crawlers are purpose-built for this task.
C. Amazon Redshift for transformations: Redshift is a data warehouse, optimized for analytical queries. While it can perform transformations, using it solely for transforming 300GB of daily data would be an overkill and a less cost-effective solution compared to EMR, especially when dealing with varied data formats.
E. Amazon SageMaker for transformations: SageMaker is primarily designed for building, training, and deploying machine learning models. While it can perform data processing as part of the ML pipeline, it's generally not the most cost-effective or direct tool for simple data transformations, especially at this scale. EMR is more suitable.

Question: 99 CertyIQ

A retail company uses AWS Glue for extract, transform, and load (ETL) operations on a dataset that contains information about customer orders. The company wants to implement specific validation rules to ensure data accuracy and consistency.

Which solution will meet these requirements?

A.Use AWS Glue job bookmarks to track the data for accuracy and consistency.
B.Create custom AWS Glue Data Quality rulesets to define specific data quality checks.
C.Use the built-in AWS Glue Data Quality transforms for standard data quality validations.
D.Use AWS Glue Data Catalog to maintain a centralized data schema and metadata repository.

Answer: B

Explanation:

The best solution for implementing specific validation rules in AWS Glue for data accuracy and consistency is to create custom AWS Glue Data Quality rulesets.

Option B is the correct choice because AWS Glue Data Quality enables users to define custom rulesets tailored to specific data requirements. These rulesets allow you to establish precise data quality checks for your dataset, ensuring that the data meets defined criteria for accuracy, completeness, consistency, and validity. This gives the company granular control over the validation process.

Option A is incorrect because AWS Glue job bookmarks are primarily used to track the progress of ETL jobs and prevent reprocessing of data that has already been processed. Bookmarks do not offer data validation functionality.

Option C, using built-in AWS Glue Data Quality transforms, might provide some standard validations, but it is likely that these pre-defined transforms might not meet the retail company's specific and unique data quality needs, which require custom rules.

Option D, AWS Glue Data Catalog, is a centralized metadata repository but does not provide the necessary functionalities for defining and enforcing data quality rules. While the Data Catalog holds schema information,

it doesn't validate data against custom rulesets.

Therefore, custom rulesets provide the necessary flexibility and control to address unique data quality requirements.

For further research, refer to AWS Glue Data Quality documentation:https://docs.aws.amazon.com/glue/latest/dg/glue_data_quality.html

Question: 100 CertyIQ

An insurance company stores transaction data that the company compressed with gzip.

The company needs to query the transaction data for occasional audits.

Which solution will meet this requirement in the MOST cost-effective way?

A.Store the data in Amazon Glacier Flexible Retrieval. Use Amazon S3 Glacier Select to query the data.
B.Store the data in Amazon S3. Use Amazon S3 Select to query the data.
C.Store the data in Amazon S3. Use Amazon Athena to query the data.
D.Store the data in Amazon Glacier Instant Retrieval. Use Amazon Athena to query the data.

Answer: A

Explanation:

The correct answer is A, storing the gzipped data in Amazon Glacier Flexible Retrieval and using S3 Glacier Select for querying. This is the most cost-effective solution for infrequent audit queries on archived data.

Option A leverages the cost-optimized storage tier of Amazon Glacier Flexible Retrieval, which is designed for data that is rarely accessed but needs to be preserved. S3 Glacier Select allows you to run SQL queries directly on data stored in Glacier without needing to restore the entire object, significantly reducing data retrieval costs and time. Since the audit is occasional, the longer retrieval times associated with Glacier Flexible Retrieval are acceptable.

Option B, storing the data in Amazon S3 and using S3 Select, is generally more expensive than Glacier Flexible Retrieval. S3 is suited for more frequent access, making it less cost-effective for infrequent audits. While S3 Select avoids full object retrieval, the storage cost of S3 is higher than Glacier.

Option C, using Amazon S3 and Athena, involves moving data from S3 to Athena for querying. This can be effective but more expensive than S3 Select for gzipped data, as Athena would need to decompress the data first. Also, the storage costs in S3 are higher than Glacier.

Option D is not recommended because Glacier Instant Retrieval, although it offers quicker data access, is significantly more expensive than Glacier Flexible Retrieval, defeating the purpose of cost-effectiveness when infrequent queries are involved. Furthermore, Athena doesn't natively query Glacier, so using Athena would require restoring data from Glacier to S3, which is not cost-effective.

Therefore, storing the data in Glacier Flexible Retrieval and using S3 Glacier Select minimizes both storage and query costs for the specific use case.

[Amazon S3 Glacier Select: https://aws.amazon.com/glacier/select/][Amazon S3 Storage Classes: https://aws.amazon.com/s3/storage-classes/][Amazon Athena: https://aws.amazon.com/athena/]

Question: 101 CertyIQ

A data engineer finished testing an Amazon Redshift stored procedure that processes and inserts data into a table that is not mission critical. The engineer wants to automatically run the stored procedure on a daily basis.

Which solution will meet this requirement in the MOST cost-effective way?

A.Create an AWS Lambda function to schedule a cron job to run the stored procedure.

B.Schedule and run the stored procedure by using the Amazon Redshift Data API in an Amazon EC2 Spot Instance.

C.Use query editor v2 to run the stored procedure on a schedule.

D.Schedule an AWS Glue Python shell job to run the stored procedure.

Answer: C

Explanation:

The correct answer is C: Use query editor v2 to run the stored procedure on a schedule. Here's why:

Cost-Effectiveness: Query editor v2 is designed to be a cost-effective method for running SQL queries and stored procedures directly against your Amazon Redshift cluster. Its scheduling features can execute the stored procedure daily without needing additional services.
Simplicity: Query editor v2 offers a simple and direct interface for scheduling and managing database tasks.
Integration with Amazon Redshift: It's specifically designed to work with Amazon Redshift, minimizing setup and configuration overhead.
No Additional Infrastructure: Using query editor v2 eliminates the need for provisioning or managing additional compute resources like Lambda functions (Option A), EC2 instances (Option B), or AWS Glue jobs (Option D). This translates to lower operational costs and reduced complexity.

Why other options are not cost-effective:

A. AWS Lambda: While Lambda is serverless, scheduling cron jobs within Lambda can be complex, and running a Lambda function daily to execute a stored procedure may result in costs from Lambda invocations.

B. EC2 Spot Instance with Redshift Data API: Using an EC2 Spot Instance with the Amazon Redshift Data API is a valid approach for running scheduled tasks, but it incurs the cost of the EC2 instance. Even with spot pricing, it's more expensive than using the native scheduling capabilities of Query editor v2.

D. AWS Glue Python Shell Job: Glue Python Shell jobs are designed for ETL (Extract, Transform, Load) operations. Using Glue solely for running a stored procedure is overkill and more expensive than alternatives, as it involves setting up Glue jobs and incurring Glue runtime costs.

Supporting Links:

Amazon Redshift Query Editor v2: https://docs.aws.amazon.com/redshift/latest/mgmt/query-editor.html
AWS Pricing: https://aws.amazon.com/pricing/

In summary, leveraging Query editor v2 provides the most direct, cost-effective, and simple method for scheduling a stored procedure to run daily within Amazon Redshift, eliminating the need for additional infrastructure or complex configurations.

Question: 102 CertyIQ

A marketing company collects clickstream data. The company sends the clickstream data to Amazon Kinesis Data Firehose and stores the clickstream data in Amazon S3. The company wants to build a series of dashboards that

hundreds of users from multiple departments will use.

The company will use Amazon QuickSight to develop the dashboards. The company wants a solution that can scale and provide daily updates about clickstream activity.

Which combination of steps will meet these requirements MOST cost-effectively? (Choose two.)

A.Use Amazon Redshift to store and query the clickstream data.

B.Use Amazon Athena to query the clickstream data

C.Use Amazon S3 analytics to query the clickstream data.

D.Access the query data through a QuickSight direct SQL query.

E.Access the query data through QuickSight SPICE (Super-fast, Parallel, In-memory Calculation Engine). Configure a daily refresh for the dataset.

Answer: BE

Explanation:

The solution requires a scalable, cost-effective data storage and query solution for clickstream data, coupled with daily updates for QuickSight dashboards.

Option B, using Amazon Athena to query the clickstream data in S3, is cost-effective. Athena is a serverless query service, meaning there are no infrastructure costs, and you only pay for the queries you run. This is ideal for ad-hoc queries and analysis of data stored in S3. Since the data is already in S3 via Kinesis Data Firehose, Athena provides direct access without requiring data migration. https://aws.amazon.com/athena/pricing/

Option E, using QuickSight SPICE with daily refreshes, is essential for the dashboard performance and update requirements. SPICE is QuickSight's in-memory engine, providing rapid data retrieval and visualization for interactive dashboards. Daily refreshes ensure the dashboards reflect the most recent clickstream activity. Direct SQL queries (Option D) to Athena each time a user interacts with the dashboard would be significantly slower and more expensive than using SPICE. SPICE imports the data into QuickSight, reducing query latency and load on Athena. https://docs.aws.amazon.com/quicksight/latest/user/spice.html

Option A, Redshift, while a powerful data warehouse, is overkill and more expensive than Athena for this specific use case of relatively simple clickstream analysis and visualization, especially considering the data is already in S3. Redshift involves infrastructure management and higher operational costs. Option C, S3 Analytics, focuses on storage analytics related to object lifecycle and storage usage patterns, not querying the data for dashboard visualization. It does not provide the querying capabilities needed for QuickSight integration.

Question: 103 CertyIQ

A data engineer is building a data orchestration workflow. The data engineer plans to use a hybrid model that includes some on-premises resources and some resources that are in the cloud. The data engineer wants to prioritize portability and open source resources.

Which service should the data engineer use in both the on-premises environment and the cloud-based environment?

A.AWS Data Exchange

B.Amazon Simple Workflow Service (Amazon SWF)

C.Amazon Managed Workflows for Apache Airflow (Amazon MWAA)

D.AWS Glue

Answer: C

Explanation:

The correct answer is C, Amazon Managed Workflows for Apache Airflow (Amazon MWAA). Here's why:

The requirement emphasizes a hybrid model, portability, and open-source resources for data orchestration. Apache Airflow is an open-source workflow management platform that excels in defining, scheduling, and monitoring workflows as Directed Acyclic Graphs (DAGs). It's highly portable because Airflow workflows (DAGs) are defined in Python code and can be executed on different infrastructures if the necessary dependencies are met.

Amazon MWAA is a managed service that makes it easier to set up, operate, and scale Airflow in the AWS cloud. Importantly, you can run the same Airflow DAGs on-premises (if you set up your own Airflow environment) and on Amazon MWAA in the cloud. This directly addresses the hybrid requirement and promotes portability, as your workflow definitions remain consistent across environments.

A. AWS Data Exchange is for finding, subscribing to, and using third-party data in the cloud, not data orchestration.

B. Amazon SWF is a fully managed workflow service for building distributed applications. While it can handle orchestration, it's less focused on data pipelines and is not directly aligned with the open-source focus like Airflow. Also, mirroring SWF functionality on-premises would require significant custom development.

D. AWS Glue is a serverless data integration service for discovering, preparing, and combining data for analytics, machine learning, and application development. While Glue can orchestrate data workflows, it is more tightly integrated with the AWS ecosystem and less portable to on-premises environments compared to Airflow. Glue also is not primarily an open-source tool.

Therefore, Amazon MWAA allows a unified approach using Apache Airflow's open-source nature to manage data orchestration both on-premises and in the cloud, fulfilling all requirements.

Supporting links:

Apache Airflow: https://airflow.apache.org/
Amazon MWAA: https://aws.amazon.com/managed-workflows-for-apache-airflow/

Question: 104 CertyIQ

A gaming company uses a NoSQL database to store customer information. The company is planning to migrate to AWS.

The company needs a fully managed AWS solution that will handle high online transaction processing (OLTP) workload, provide single-digit millisecond performance, and provide high availability around the world.

Which solution will meet these requirements with the LEAST operational overhead?

A.Amazon Keyspaces (for Apache Cassandra)
B.Amazon DocumentDB (with MongoDB compatibility)
C.Amazon DynamoDB
D.Amazon Timestream

Answer: C

Explanation:

The correct answer is **C. Amazon DynamoDB**. Here's why:

DynamoDB is a fully managed, serverless, key-value and document database designed for high-performance, high-availability applications at any scale. It's ideally suited for OLTP workloads needing single-digit millisecond latency. Its distributed architecture ensures global availability and fault tolerance, minimizing operational overhead for the gaming company. DynamoDB's serverless nature means the company doesn't have to manage servers, patching, scaling, or backups, further reducing operational burden.

Amazon Keyspaces (A) offers Cassandra compatibility, which might be useful if the company already uses Cassandra, but DynamoDB's native design on AWS offers tighter integration and typically lower operational overhead. Also, Keyspaces doesn't inherently provide single-digit millisecond performance without careful tuning.

Amazon DocumentDB (B), while compatible with MongoDB, isn't optimized for the same scale and speed as DynamoDB for OLTP workloads. Its focus is more on document storage and complex queries, rather than the consistently low latency required for high-transaction gaming applications.

Amazon Timestream (D) is a time-series database, not suitable for general customer information storage or OLTP workloads. It's optimized for analyzing time-series data, such as sensor readings or financial data.

Therefore, DynamoDB's fully managed nature, consistent single-digit millisecond performance, and high availability make it the best choice for minimizing operational overhead while meeting the gaming company's requirements.

Further reading:

Amazon DynamoDB
Amazon Keyspaces
Amazon DocumentDB
Amazon Timestream

Question: 105 CertyIQ

A data engineer creates an AWS Lambda function that an Amazon EventBridge event will invoke. When the data engineer tries to invoke the Lambda function by using an EventBridge event, an AccessDeniedException message appears.

How should the data engineer resolve the exception?

A.Ensure that the trust policy of the Lambda function execution role allows EventBridge to assume the execution role.

B.Ensure that both the IAM role that EventBridge uses and the Lambda function's resource-based policy have the necessary permissions.

C.Ensure that the subnet where the Lambda function is deployed is configured to be a private subnet.

D.Ensure that EventBridge schemas are valid and that the event mapping configuration is correct.

Answer: B

Explanation:

The AccessDeniedException when invoking a Lambda function via EventBridge signifies a permission issue. Two key areas govern these permissions: the IAM role EventBridge uses and the Lambda function's resource-based policy.

Option B directly addresses this. The IAM role used by EventBridge needs permission to invoke the Lambda function. This is usually done through an IAM policy attached to the EventBridge role that grants lambda:InvokeFunction permission on the specific Lambda function resource or a wildcard.

The Lambda function's resource-based policy also plays a critical role. This policy defines who can invoke the function. EventBridge must be explicitly allowed to invoke the function through this resource-based policy. This ensures that the Lambda function trusts EventBridge as a valid invoker. Without this, even if EventBridge's IAM role has invoke permissions, the Lambda function itself will deny the request.

Option A only addresses one side of the equation: the Lambda function's execution role's trust policy. While the execution role does need a trust policy allowing AWS services like Lambda to assume it, the EventBridge service needs permission to invoke the function in the first place. EventBridge isn't assuming the Lambda execution role, it's invoking the function.

Option C is irrelevant. Whether the Lambda function is in a private or public subnet impacts its network connectivity but doesn't relate to invocation permissions granted by IAM or the Lambda function policy.

Option D focuses on the event structure and schema validation. While schema validation is important for data integrity, it does not directly address the AccessDeniedException. This exception indicates a problem with authorization, not data validation. Even if the event schema is valid, if EventBridge lacks the correct permissions, the invocation will fail with an AccessDeniedException.

Therefore, option B is the most comprehensive and correct approach because it ensures that both EventBridge has permission to invoke the Lambda function and the Lambda function trusts EventBridge's invocation request, resolving the AccessDeniedException.

For further research, refer to the AWS documentation on Lambda permissions and EventBridge integration:

AWS Lambda Permissions: https://docs.aws.amazon.com/lambda/latest/dg/invocation-permissions.html
Using AWS Lambda with Amazon EventBridge: https://docs.aws.amazon.com/eventbridge/latest/userguide/put-events-to-lambda.html

Question: 106 CertyIQ

A company uses a data lake that is based on an Amazon S3 bucket. To comply with regulations, the company must apply two layers of server-side encryption to files that are uploaded to the S3 bucket. The company wants to use an AWS Lambda function to apply the necessary encryption.

Which solution will meet these requirements?

A.Use both server-side encryption with AWS KMS keys (SSE-KMS) and the Amazon S3 Encryption Client.
B.Use dual-layer server-side encryption with AWS KMS keys (DSSE-KMS).
C.Use server-side encryption with customer-provided keys (SSE-C) before files are uploaded.
D.Use server-side encryption with AWS KMS keys (SSE-KMS).

Answer: B

Explanation:

The correct answer is **B. Use dual-layer server-side encryption with AWS KMS keys (DSSE-KMS).**

Here's a detailed justification:

The requirement is to apply two layers of server-side encryption. This implies a need for a mechanism that encrypts the data twice at the server level. Option B, DSSE-KMS, directly addresses this requirement. DSSE-KMS is specifically designed to provide dual-layer encryption using AWS Key Management Service (KMS) keys. It encrypts the data twice with two different keys managed by KMS, enhancing security beyond single-layer encryption.

Option A, while involving KMS encryption, doesn't inherently provide two layers of encryption using a single

AWS mechanism. Using the S3 Encryption Client alongside SSE-KMS would require custom implementation and management, adding complexity and potentially introducing vulnerabilities if not implemented correctly. While the S3 Encryption Client is useful, it's not the most straightforward way to fulfill a dual-layer requirement.

Option C, SSE-C, requires the company to manage the encryption keys. This contradicts the use of an AWS Lambda function for encryption, as the Lambda function would need to be provisioned with the customer-provided keys. This also introduces a significant key management burden on the company. Moreover, it only provides one layer of encryption.

Option D, SSE-KMS, provides only a single layer of encryption. It doesn't satisfy the requirement of two layers of server-side encryption.

Therefore, DSSE-KMS provides a built-in, managed, and efficient way to apply two layers of server-side encryption with KMS, fulfilling the regulatory requirements without added complexity. It centralizes key management with KMS and offers enhanced security.

Further Research:

DSSE-KMS Documentation:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingKMSEncryption.html (Look for the Dual-Layer Server-Side Encryption with KMS keys (DSSE-KMS) section.)
AWS KMS: https://aws.amazon.com/kms/
Amazon S3 Encryption: https://docs.aws.amazon.com/AmazonS3/latest/userguide/serv-side-encryption.html

Question: 107 CertyIQ

A data engineer notices that Amazon Athena queries are held in a queue before the queries run.

How can the data engineer prevent the queries from queueing?

A.Increase the query result limit.
B.Configure provisioned capacity for an existing workgroup.
C.Use federated queries.
D.Allow users who run the Athena queries to an existing workgroup.

Answer: B

Explanation:

The correct answer is **B. Configure provisioned capacity for an existing workgroup.**

Athena queries queue when there are insufficient resources to execute them immediately. This typically happens when the workgroup's capacity to handle concurrent queries is exceeded. Athena uses data source connectors to analyze data, and each query needs resources like compute power to run.

Option B directly addresses this issue. By configuring provisioned capacity for a workgroup, you are reserving a specific amount of Data Processing Units (DPUs) exclusively for that workgroup. This ensures that a defined amount of resources are always available to execute the queries submitted to the workgroup, thus minimizing queuing. Provisioning capacity allows you to control and pay for a predictable amount of processing power.

Option A, increasing the query result limit, only affects the amount of data returned by a query, not the resource availability for executing it. Option C, using federated queries, might introduce new queuing challenges if the federated data sources are overloaded. Option D, allowing more users to an existing workgroup, would likely exacerbate the queuing problem if there aren't enough resources to begin with. More

users submitting queries to a workgroup with limited capacity could lead to longer wait times. Configuring provisioned capacity is the most direct method to reduce queuing by increasing the processing power available for queries within the workgroup. This ensures that Athena can immediately execute incoming requests without delay.

Refer to the official AWS documentation for more details on provisioned capacity and workgroups: https://docs.aws.amazon.com/athena/latest/ug/manage-queries-workgroups-console.html and https://aws.amazon.com/blogs/big-data/optimize-amazon-athena-cost-and-performance-with-provisioned-dpus/

Question: 108 CertyIQ

A data engineer needs to debug an AWS Glue job that reads from Amazon S3 and writes to Amazon Redshift. The data engineer enabled the bookmark feature for the AWS Glue job.
The data engineer has set the maximum concurrency for the AWS Glue job to 1.

The AWS Glue job is successfully writing the output to Amazon Redshift. However, the Amazon S3 files that were loaded during previous runs of the AWS Glue job are being reprocessed by subsequent runs.

What is the likely reason the AWS Glue job is reprocessing the files?

A.The AWS Glue job does not have the s3:GetObjectAcl permission that is required for bookmarks to work correctly.

B.The maximum concurrency for the AWS Glue job is set to 1.

C.The data engineer incorrectly specified an older version of AWS Glue for the Glue job.

D.The AWS Glue job does not have a required commit statement.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most likely reason for the AWS Glue job reprocessing files, focusing on the bookmark feature and its requirements:

Option D: The AWS Glue job does not have a required commit statement.

The core concept here is that AWS Glue bookmarks track processed data for incremental data loading. Bookmarks prevent reprocessing of data, but they rely on the job properly committing its progress. Without a commit, Glue can't reliably record which files were processed during a specific run. Essentially, the bookmark isn't updated correctly, causing subsequent job runs to start from the beginning, reprocessing all files.

Here's a breakdown:

1. **AWS Glue Bookmarks:** Bookmarks enable Glue jobs to process only new or updated data in a source dataset, such as files in S3. This prevents reprocessing of already processed data.
2. **Commit Mechanism:** Bookmarks work by tracking the last processed state. However, merely processing the data isn't enough; the Glue job must explicitly commit its progress to update the bookmark. This involves signaling to Glue that the data has been successfully written to the target (Redshift, in this case) and that the bookmark should be updated to reflect the new state.
3. **Absence of Commit:** If the job doesn't contain the necessary commit logic (often through calls to commit operations within the DynamicFrame or DataFrame write operations), Glue can't save the last processed state. Consequently, on the next run, it assumes no data has been processed and starts over.

4. **Impact on Reprocessing:** This lack of commit is the root cause of the described problem. Even though the job is writing to Redshift, Glue doesn't know which data was successfully written unless a commit updates the bookmark.

5. **Illustrative Example:** Imagine a database transaction. Data is manipulated, but it is only permanently saved after a COMMIT statement. Similarly, in Glue, the data is processed and written, but the 'transaction' is not finalized in bookmark terms without an explicit 'commit'.

6. **Concurrency and Permissions (Options A & B):** While concurrency and permissions can affect job behavior, they are not directly related to the bookmark feature failing to prevent reprocessing. The concurrency being set to 1 eliminates the chance of concurrent tasks interfering with bookmark updates. s3:GetObjectAcl permission, while potentially important in other aspects, is not directly essential for bookmark functionality.

7. **Glue Version (Option C):** Using an older Glue version might present other problems, but it is less directly related to why a bookmark isn't preventing reprocessing if a commit mechanism is missing.

Therefore, the absence of a proper commit mechanism within the Glue job's code is the most probable reason for the observed reprocessing behavior.

Authoritative Links:

AWS Glue Bookmarks: https://docs.aws.amazon.com/glue/latest/dg/monitor-continuations.html
AWS Glue Programming Guide: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming.html

Question: 109 CertyIQ

An ecommerce company wants to use AWS to migrate data pipelines from an on-premises environment into the AWS Cloud. The company currently uses a third-party tool in the on-premises environment to orchestrate data ingestion processes.

The company wants a migration solution that does not require the company to manage servers. The solution must be able to orchestrate Python and Bash scripts. The solution must not require the company to refactor any code.

Which solution will meet these requirements with the LEAST operational overhead?

A.AWS Lambda
B.Amazon Managed Workflows for Apache Airflow (Amazon MVVAA)
C.AWS Step Functions
D.AWS Glue

Answer: B

Explanation:

The correct answer is **B. Amazon Managed Workflows for Apache Airflow (Amazon MWAA)**. Here's why:

Orchestration: The question highlights a requirement for orchestrating data ingestion processes, implying a workflow-driven need. Both Amazon MWAA and AWS Step Functions are workflow orchestration services.
Python and Bash Script Support: MWAA excels at this. Apache Airflow is natively designed to handle Python scripts through its PythonOperator. It can also execute Bash scripts using the BashOperator.
Serverless Nature: MWAA is a managed service, meaning AWS handles the underlying infrastructure, relieving the company of server management overhead, fulfilling a critical requirement.
No Code Refactoring: Since the company already uses a third-party orchestration tool, migrating to Airflow (via MWAA) allows them to largely port existing workflows (DAGs) with minimal code changes, fulfilling the "no refactoring" constraint.

Operational Overhead: MWAA's managed nature drastically reduces operational overhead compared to alternatives that would require server management or significant code adaptations.

Here's why the other options are less suitable:

A. AWS Lambda: Lambda is designed for event-driven, short-running tasks. While it can execute Python, it's not primarily designed for complex workflow orchestration like Airflow. It lacks built-in workflow management features.
C. AWS Step Functions: Step Functions is an orchestration service, but it primarily uses JSON-based state machine definitions. Executing arbitrary Python or Bash scripts directly within Step Functions requires workarounds (like invoking Lambda functions), increasing complexity and operational overhead.
D. AWS Glue: AWS Glue is a serverless data integration service focused on ETL. While it provides some orchestration capabilities, it is primarily optimized for data transformation rather than general-purpose workflow management and script execution. Using Glue for script orchestration can be cumbersome.

Therefore, Amazon MWAA directly addresses all requirements: orchestration, Python/Bash script support, serverless operation, and minimal code refactoring, with the least operational overhead.

Authoritative Links:

Amazon MWAA: https://aws.amazon.com/managed-workflows-for-apache-airflow/
Apache Airflow: https://airflow.apache.org/

Question: 110 CertyIQ

A retail company stores data from a product lifecycle management (PLM) application in an on-premises MySQL database. The PLM application frequently updates the database when transactions occur.

The company wants to gather insights from the PLM application in near real time. The company wants to integrate the insights with other business datasets and to analyze the combined dataset by using an Amazon Redshift data warehouse.

The company has already established an AWS Direct Connect connection between the on-premises infrastructure and AWS.

Which solution will meet these requirements with the LEAST development effort?

A.Run a scheduled AWS Glue extract, transform, and load (ETL) job to get the MySQL database updates by using a Java Database Connectivity (JDBC) connection. Set Amazon Redshift as the destination for the ETL job.

B.Run a full load plus CDC task in AWS Database Migration Service (AWS DMS) to continuously replicate the MySQL database changes. Set Amazon Redshift as the destination for the task.

C.Use the Amazon AppFlow SDK to build a custom connector for the MySQL database to continuously replicate the database changes. Set Amazon Redshift as the destination for the connector.

D.Run scheduled AWS DataSync tasks to synchronize data from the MySQL database. Set Amazon Redshift as the destination for the tasks.

Answer: B

Explanation:

The best solution for near real-time data integration from an on-premises MySQL database to Amazon Redshift with minimal development effort is **B. Run a full load plus CDC task in AWS Database Migration Service (AWS DMS) to continuously replicate the MySQL database changes. Set Amazon Redshift as the destination for the task.**

Here's a breakdown of why this option is superior:

AWS DMS Purpose-Built for Migration & Replication: AWS DMS is explicitly designed for database migration and continuous data replication. It streamlines the process of moving data from various source databases (like MySQL) to target databases (like Amazon Redshift). https://aws.amazon.com/dms/
Change Data Capture (CDC): DMS's CDC feature efficiently captures and applies incremental changes from the source database to the target in near real-time. This avoids full database dumps and reduces latency for data availability in Redshift. https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Introduction.html
Minimal Development Effort: DMS provides a managed service where most configuration is done through the AWS console or API calls. It handles the complexities of data extraction, transformation (if needed), and loading, reducing the need for custom code.
Direct Integration: DMS natively supports both MySQL as a source and Amazon Redshift as a target. This simplifies configuration and reduces the risk of compatibility issues.
Full Load + CDC Approach: This strategy handles the initial bulk data load of existing data and then continuously replicates the subsequent changes to provide up-to-date data for analysis.

Let's contrast this with the other options:

Option A (AWS Glue ETL): While Glue can extract data, scheduling jobs might not provide the near real-time latency desired. It also involves more ETL coding compared to DMS's CDC functionality.
Option C (Amazon AppFlow SDK): AppFlow is better suited for SaaS applications, and using its SDK to build a custom connector for MySQL would require significant development effort.
Option D (AWS DataSync): DataSync is optimized for large-scale file transfers, not database replication. It would also necessitate a scheduled approach and would not easily capture incremental changes in near real time. It is also not designed to directly load into a Redshift data warehouse.

In summary, AWS DMS with full load plus CDC offers the most efficient and least development-intensive solution for continuously replicating data from the on-premises MySQL database to Amazon Redshift, fulfilling the requirements of near real-time insights and integration with other business datasets.

Question: 111 CertyIQ

A marketing company uses Amazon S3 to store clickstream data. The company queries the data at the end of each day by using a SQL JOIN clause on S3 objects that are stored in separate buckets.

The company creates key performance indicators (KPIs) based on the objects. The company needs a serverless solution that will give users the ability to query data by partitioning the data. The solution must maintain the atomicity, consistency, isolation, and durability (ACID) properties of the data.

Which solution will meet these requirements MOST cost-effectively?

A.Amazon S3 Select
B.Amazon Redshift Spectrum
C.Amazon Athena
D.Amazon EMR

Answer: C

Explanation:

The correct answer is **C. Amazon Athena**. Here's why:

Serverless: Athena is a serverless query service, meaning there are no servers to manage. This aligns with the requirement for a serverless solution.
SQL Queries on S3: Athena allows users to execute SQL queries directly against data stored in Amazon S3. This directly addresses the company's need to query data at the end of each day.

Data Partitioning: Athena supports data partitioning. Partitioning involves dividing the data into smaller, manageable parts based on a specific column or attribute (e.g., date). This significantly improves query performance because Athena only scans the relevant partitions rather than the entire dataset.
Cost-Effectiveness: Athena is a pay-per-query service. You are charged based on the amount of data scanned during query execution. Since partitioning reduces the amount of data scanned, it leads to significant cost savings.
ACID Properties: While Athena is not a fully ACID-compliant database, it provides mechanisms to maintain data consistency and atomicity for simple operations when used with data formats like Parquet and ORC which support schema evolution and partition pruning.
Alternatives Analysis:

S3 Select (A): S3 Select allows you to retrieve subsets of data from S3 objects using SQL. While it can reduce data transfer costs, it doesn't offer partitioning capabilities, and the features for joining data from multiple sources and creating complex queries are not as powerful as Athena.
Redshift Spectrum (B): Redshift Spectrum also allows querying data in S3 using SQL. However, it's primarily designed for data warehousing and analytics at scale. Setting up and managing a Redshift cluster would be overkill and more expensive than Athena for this specific use case.
Amazon EMR (D): EMR is a managed Hadoop framework. While it could be used to process and query data in S3, it involves managing clusters, which goes against the serverless requirement and is more complex and costly than Athena.

Authoritative Links:

Amazon Athena: https://aws.amazon.com/athena/
Athena Partitioning: https://docs.aws.amazon.com/athena/latest/ug/partitions.html

Question: 112 CertyIQ

A company wants to migrate data from an Amazon RDS for PostgreSQL DB instance in the eu-east-1 Region of an AWS account named Account_A. The company will migrate the data to an Amazon Redshift cluster in the eu-west-1 Region of an AWS account named Account_B.

Which solution will give AWS Database Migration Service (AWS DMS) the ability to replicate data between two data stores?

A.Set up an AWS DMS replication instance in Account_B in eu-west-1.
B.Set up an AWS DMS replication instance in Account_B in eu-east-1.
C.Set up an AWS DMS replication instance in a new AWS account in eu-west-1.
D.Set up an AWS DMS replication instance in Account_A in eu-east-1.

Answer: A

Explanation:

The correct answer is A: Set up an AWS DMS replication instance in Account_B in eu-west-1.

Here's why:

DMS Replication Instance Location: The DMS replication instance should be located as close as possible to the target database to minimize latency and network transfer costs. Since the target Redshift cluster is in Account_B in eu-west-1, the replication instance should be located in the same region and account.
Cross-Account Access: AWS DMS needs appropriate IAM roles and security group configurations to access both the source (RDS PostgreSQL in Account_A, eu-east-1) and target (Redshift in Account_B, eu-west-1) databases. Setting up the instance in Account_B simplifies permission management for writing to the target.
DMS Functionality: The replication instance handles data transformation and movement. It needs to be

capable of connecting to both source and target, which is best achieved when it resides close to the target.
Incorrect Options:

Option B is wrong because the replication instance needs to be close to the target Redshift cluster, not the source RDS database. Placing it in eu-east-1 would incur higher latency and data transfer costs when writing to Redshift in eu-west-1.
Option C, using a new AWS account, adds unnecessary complexity. While feasible, it does not offer any specific advantage over using Account_B directly. In this case, more accounts equal more management overhead.
Option D is wrong for the same reason as Option B. DMS should be close to the target.

Supporting documentation:

AWS DMS Best Practices: This documentation emphasizes the importance of replication instance placement and network considerations.
AWS DMS Security: This describes the security and IAM roles requirements to use DMS.
AWS DMS Cross-Account Replication: Provides info about how to configure cross-account AWS DMS.

The key takeaway is that the DMS replication instance's location should be optimized for accessing the target database to improve performance and minimize costs.

Question: 113 CertyIQ

A company uses Amazon S3 as a data lake. The company sets up a data warehouse by using a multi-node Amazon Redshift cluster. The company organizes the data files in the data lake based on the data source of each data file.

The company loads all the data files into one table in the Redshift cluster by using a separate COPY command for each data file location. This approach takes a long time to load all the data files into the table. The company must increase the speed of the data ingestion. The company does not want to increase the cost of the process.

Which solution will meet these requirements?

A.Use a provisioned Amazon EMR cluster to copy all the data files into one folder. Use a COPY command to load the data into Amazon Redshift.

B.Load all the data files in parallel into Amazon Aurora. Run an AWS Glue job to load the data into Amazon Redshift.

C.Use an AWS Give job to copy all the data files into one folder. Use a COPY command to load the data into Amazon Redshift.

D.Create a manifest file that contains the data file locations. Use a COPY command to load the data into Amazon Redshift.

Answer: D

Explanation:

The best solution to accelerate data ingestion into Amazon Redshift from an S3 data lake, without increasing costs, is to use a manifest file with the COPY command (Option D).

The problem highlights the inefficiency of using multiple COPY commands, each targeting a single file location. This approach introduces significant overhead due to the repeated connection and authentication processes for each file. Redshift's COPY command is optimized for parallel data loading from multiple files concurrently when pointed to a single location containing multiple files or via a manifest file specifying various file paths.

A manifest file lists the S3 URIs of the data files to be loaded. The COPY command can then reference this manifest file, allowing Redshift to efficiently load data from multiple files in parallel. This significantly

reduces the overall data ingestion time.

The other options are less suitable:

Option A (EMR): Using a provisioned EMR cluster would involve significant costs for cluster creation, management, and operation. While EMR could consolidate files, it adds unnecessary complexity and expense when a simpler solution exists.

Option B (Aurora/Glue): Loading data into Aurora first adds an extra hop and complexity. Using Glue to load from Aurora to Redshift introduces additional operational overhead and latency. This also likely incurs extra costs associated with Aurora.

Option C (Glue/Single Folder): Although Glue could consolidate files into a single folder, using a manifest file directly with COPY is even more straightforward and avoids the need to move the files, further saving time and resources. The overhead associated with Glue is more than necessary.

By leveraging the manifest file feature of the COPY command, the company can achieve parallel data loading, minimizing the number of calls to S3 and maximizing the utilization of Redshift's parallel processing capabilities without incurring additional costs. This approach perfectly addresses the requirements of increasing speed while avoiding increased costs.

Relevant Documentation:

Amazon Redshift COPY command: https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html
Using a Manifest File:
https://docs.aws.amazon.com/redshift/latest/dg/r_COPY_command_examples.html#r_COPY_command_examples-copy-from-manifest

Question: 114 CertyIQ

A company plans to use Amazon Kinesis Data Firehose to store data in Amazon S3. The source data consists of 2 MB .csv files. The company must convert the .csv files to JSON format. The company must store the files in Apache Parquet format.

Which solution will meet these requirements with the LEAST development effort?

A.Use Kinesis Data Firehose to convert the .csv files to JSON. Use an AWS Lambda function to store the files in Parquet format.

B.Use Kinesis Data Firehose to convert the .csv files to JSON and to store the files in Parquet format.

C.Use Kinesis Data Firehose to invoke an AWS Lambda function that transforms the .csv files to JSON and stores the files in Parquet format.

D.Use Kinesis Data Firehose to invoke an AWS Lambda function that transforms the .csv files to JSON. Use Kinesis Data Firehose to store the files in Parquet format.

Answer: D

Explanation:

The correct answer is D. Here's a detailed justification:

The requirement is to convert CSV to JSON, then to Parquet, and store in S3 using Kinesis Data Firehose with minimal development effort.

Option D offers the most straightforward approach using Kinesis Data Firehose's built-in integration with AWS Lambda for data transformation and its ability to convert data format to Parquet.

Kinesis Data Firehose can directly invoke a Lambda function. This Lambda function transforms the initial CSV

data into JSON format. This is a common and efficient practice for real-time data transformation.

After the JSON transformation by Lambda, Kinesis Data Firehose can then leverage its built-in conversion capabilities to convert the JSON data into the Parquet format before storing it in S3. This eliminates the need to implement Parquet conversion logic within the Lambda function. Firehose directly integrates with Apache Hive Metastore and provides flexibility with Parquet versions and column encodings.

Options A, B, and C are less ideal:

Option A: Storing in Parquet using Lambda after JSON transformation adds complexity.
Option B: Kinesis Data Firehose cannot perform the conversion from CSV to JSON natively. It requires Lambda for custom transformations.
Option C: Performing both JSON and Parquet conversion in a single Lambda function increases the function's complexity and might lead to performance bottlenecks. It is better to offload the parquet conversion to Firehose which is specifically designed for this type of batch conversion.

Therefore, Option D strategically divides the transformation steps, leveraging Lambda for the CSV-to-JSON transformation (which Firehose cannot do itself) and then utilizing Firehose's built-in capability for JSON-to-Parquet conversion and efficient storage, minimizing development effort.

Supporting Resources:

AWS Kinesis Data Firehose Data Transformation: https://docs.aws.amazon.com/firehose/latest/dev/data-transformation.html
AWS Lambda: https://aws.amazon.com/lambda/
Kinesis Data Firehose Data Format Conversion : https://docs.aws.amazon.com/firehose/latest/dev/data-format-conversion.html

Question: 115 CertyIQ

A company is using an AWS Transfer Family server to migrate data from an on-premises environment to AWS. Company policy mandates the use of TLS 1.2 or above to encrypt the data in transit.

Which solution will meet these requirements?

A.Generate new SSH keys for the Transfer Family server. Make the old keys and the new keys available for use.
B.Update the security group rules for the on-premises network to allow only connections that use TLS 1.2 or above.
C.Update the security policy of the Transfer Family server to specify a minimum protocol version of TLS 1.2
D.Install an SSL certificate on the Transfer Family server to encrypt data transfers by using TLS 1.2.

Answer: C

Explanation:

The correct answer is **C. Update the security policy of the Transfer Family server to specify a minimum protocol version of TLS 1.2.**

Here's why:

AWS Transfer Family servers provide a built-in mechanism to control the security protocols used for data transfer through security policies. A security policy allows you to specify the minimum TLS protocol version that the server will accept. By setting the minimum protocol version to TLS 1.2, you directly enforce the company's security requirement for data encryption in transit. AWS Transfer Family supports TLS versions 1.2 and 1.3.

Option A is incorrect because SSH keys are used for authentication, not for controlling the TLS protocol version used for encryption. While SSH provides secure transport, it doesn't guarantee TLS 1.2 or above without further configuration. Simply generating and making keys available doesn't enforce the protocol version.

Option B is incorrect because updating security group rules on the on-premises network will not enforce the TLS version used by the Transfer Family server. Security groups control network access, not the protocols used within a connection. The Transfer Family server needs to be configured to enforce the TLS version.

Option D is partially correct in that TLS involves certificates. However, simply installing an SSL certificate doesn't guarantee that TLS 1.2 or above will be used. The certificate enables secure communication, but the server configuration determines the minimum TLS version. Specifically configuring the security policy addresses the requirement directly and correctly.

In summary, directly configuring the AWS Transfer Family server's security policy is the most effective and compliant way to ensure that only TLS 1.2 or higher is used for encrypting data in transit, fulfilling the company's security mandate.

Supporting Documentation:

AWS Transfer Family Security Policies: https://docs.aws.amazon.com/transfer/latest/userguide/security-policies.html
AWS Transfer Family Supported Cipher Suites: https://docs.aws.amazon.com/transfer/latest/userguide/security-policies.html#security-policies-supported

Question: 116 CertyIQ

A company wants to migrate an application and an on-premises Apache Kafka server to AWS. The application processes incremental updates that an on-premises Oracle database sends to the Kafka server. The company wants to use the replatform migration strategy instead of the refactor strategy.

Which solution will meet these requirements with the LEAST management overhead?

A.Amazon Kinesis Data Streams
B.Amazon Managed Streaming for Apache Kafka (Amazon MSK) provisioned cluster
C.Amazon Kinesis Data Firehose
D.Amazon Managed Streaming for Apache Kafka (Amazon MSK) Serverless

Answer: D

Explanation:

The correct answer is **D. Amazon Managed Streaming for Apache Kafka (Amazon MSK) Serverless**. Here's why:

The question emphasizes a "replatform" migration strategy, which means moving the application with minimal code changes. Keeping the Kafka service is preferred, but the company also wants to reduce management overhead.

Amazon MSK Serverless is the best fit because it fully manages the Kafka cluster, eliminating the need for instance provisioning, scaling, and patching. This drastically reduces the management burden compared to a provisioned MSK cluster. MSK Serverless scales automatically based on demand, efficiently handling variable workloads.

Amazon MSK Provisioned Cluster is a valid option for Kafka but requires significant management overhead for cluster sizing, scaling, patching, and monitoring. This contradicts the question's requirement for minimal

management.

Amazon Kinesis Data Streams requires application code changes to adapt to the Kinesis API, as it's not a drop-in replacement for Kafka. This violates the replatform strategy. Kinesis also lacks some Kafka features like message ordering within partitions with consumer groups when scaling.

Amazon Kinesis Data Firehose is designed for loading streaming data into data lakes and warehouses, not for replacing a Kafka message broker used by an application. It is generally suitable for a different use case (ETL rather than near real-time processing).

Therefore, MSK Serverless offers the best balance of keeping the existing Kafka functionality (replatform) while minimizing operational complexity and management overhead, in accordance with the prompt's requirements.

Supporting Links:

Amazon MSK Serverless: https://aws.amazon.com/msk/serverless/
Replatform Migration Strategy: https://aws.amazon.com/cloud-migration/strategies/

Question: 117 CertyIQ

A data engineer is building an automated extract, transform, and load (ETL) ingestion pipeline by using AWS Glue. The pipeline ingests compressed files that are in an Amazon S3 bucket. The ingestion pipeline must support incremental data processing.

Which AWS Glue feature should the data engineer use to meet this requirement?

A.Workflows
B.Triggers
C.Job bookmarks
D.Classifiers

Answer: C

Explanation:

The correct answer is **C. Job bookmarks.** Here's a detailed justification:

The requirement is to process data incrementally. This means the ETL pipeline should only process new or updated files in the S3 bucket since the last run, avoiding reprocessing the entire dataset each time. AWS Glue Job Bookmarks are designed precisely for this scenario. They track the state of the data processed in a previous run.

When a Glue job with bookmarks enabled starts, it checks the bookmark for the last processed file or directory. It then only processes files that have been added or modified since that last run. This significantly reduces processing time and costs, especially with large datasets.

Workflows (Option A) are used to orchestrate complex ETL pipelines, managing dependencies and triggering jobs in a specific order. While useful for overall pipeline management, they don't inherently handle incremental processing.

Triggers (Option B) initiate Glue jobs based on events, such as a schedule or the completion of another job. They don't track the state of processed data for incremental loading. They simply start the job.

Classifiers (Option D) infer the schema of data stored in S3. They help Glue understand the data's structure but are unrelated to incremental data processing.

Therefore, Job Bookmarks directly address the requirement of incremental data processing in the AWS Glue ETL pipeline.

For further research, refer to the official AWS documentation:

AWS Glue Job Bookmarks: https://docs.aws.amazon.com/glue/latest/dg/monitor-continuations.html

Question: 118 CertyIQ

A banking company uses an application to collect large volumes of transactional data. The company uses Amazon Kinesis Data Streams for real-time analytics. The company's application uses the PutRecord action to send data to Kinesis Data Streams.

A data engineer has observed network outages during certain times of day. The data engineer wants to configure exactly-once delivery for the entire processing pipeline.

Which solution will meet this requirement?

A.Design the application so it can remove duplicates during processing by embedding a unique ID in each record at the source.

B.Update the checkpoint configuration of the Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) data collection application to avoid duplicate processing of events.

C.Design the data source so events are not ingested into Kinesis Data Streams multiple times.

D.Stop using Kinesis Data Streams. Use Amazon EMR instead. Use Apache Flink and Apache Spark Streaming in Amazon EMR.

Answer: A

Explanation:

The correct answer is A: Design the application so it can remove duplicates during processing by embedding a unique ID in each record at the source.

Here's a detailed justification:

The problem requires achieving exactly-once delivery in a Kinesis Data Streams pipeline, specifically when network outages can cause retries and thus duplicate data. Kinesis Data Streams itself offers at-least-once delivery. Therefore, mitigating duplicates needs to happen elsewhere.

Option A directly addresses this limitation by implementing idempotency at the application level. By embedding a unique ID in each record, the downstream processing applications can identify and discard duplicate records regardless of how many times the record was initially sent to the stream. This strategy ensures that each record is processed only once effectively.

Option B, modifying the Flink checkpoint configuration, helps improve fault tolerance and reduces data loss, but it doesn't guarantee exactly-once processing from the source if the source is sending the same data multiple times during retries. Checkpointing ensures that the Flink application can recover its state after a failure and continue processing from where it left off, but it doesn't prevent the duplicates from being read from Kinesis in the first place.

Option C, aiming to prevent multiple ingests at the source, is ideal but practically impossible to guarantee in a distributed system facing network outages. Retries are crucial for resilience. Eliminating retries introduces significant data loss risk.

Option D, switching to EMR, Flink, and Spark Streaming, introduces unnecessary complexity and cost. It doesn't fundamentally solve the root problem of duplicate records if the source application is still retrying sends. Moreover, Spark Streaming, by itself, also doesn't guarantee exactly-once processing without

additional effort.

Therefore, the most practical and cost-effective solution is to make the data pipeline idempotent by de-duplicating records using a unique ID. This approach allows for retries (for reliability) while ensuring exactly-once processing semantics are achieved.

Here are some resources for further research:

Idempotency: https://en.wikipedia.org/wiki/Idempotence
Kinesis Data Streams best practices: https://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-duplicates.html (specifically regarding record processor duplicates)
Exactly-once processing with Flink: https://nightlies.apache.org/flink/flink-docs-stable/docs/connectors/datastream/overview/#delivery-guarantees

Question: 119 CertyIQ

A company stores logs in an Amazon S3 bucket. When a data engineer attempts to access several log files, the data engineer discovers that some files have been unintentionally deleted.

The data engineer needs a solution that will prevent unintentional file deletion in the future.

Which solution will meet this requirement with the LEAST operational overhead?

A.Manually back up the S3 bucket on a regular basis.
B.Enable S3 Versioning for the S3 bucket.
C.Configure replication for the S3 bucket.
D.Use an Amazon S3 Glacier storage class to archive the data that is in the S3 bucket.

Answer: B

Explanation:

The correct answer is B, enabling S3 Versioning for the S3 bucket. Here's why:

S3 Versioning is designed to protect data against accidental deletion or overwrites. When Versioning is enabled, every object in the bucket has a unique version ID. If a user attempts to delete an object, S3 does not permanently delete it; instead, it creates a delete marker, indicating that the current version is the deleted one. The previous version remains in the bucket and can be easily restored.

Option A (manual backups) is operationally heavy. It requires scheduling, managing storage for backups, and potentially complex restoration procedures. This incurs higher administrative burden and cost compared to Versioning.

Option C (replication) is primarily for disaster recovery and high availability, not accidental deletion protection. Replication creates copies of data in different regions, which is helpful if an entire region becomes unavailable. However, if a file is deleted from the source bucket, the deletion is typically replicated to the destination bucket, so it won't prevent accidental deletion.

Option D (S3 Glacier) is designed for long-term archival and infrequent access. While it offers data preservation, it's not a solution for preventing accidental deletion. Moving data to Glacier involves retrieval costs and retrieval times are significantly longer than accessing data stored in standard S3. This makes it unsuitable for a scenario where the data is actively being accessed, as described in the question. Also, migrating to Glacier does not prevent accidental deletion in the source S3 bucket before the migration.

Versioning directly addresses the problem with minimal administrative overhead. Enabling it is a simple configuration change on the S3 bucket. It allows easy recovery of accidentally deleted files without requiring

complex restoration processes. The previous versions are stored within S3 and can be restored via the AWS console or API.

In summary, S3 Versioning provides the simplest, most cost-effective, and least operationally heavy way to prevent accidental file deletion. It offers version control, allowing for quick and easy recovery of previous file versions.

Supporting links:

AWS S3 Versioning Documentation
AWS S3 Replication Documentation
AWS S3 Glacier Documentation

Question: 120 CertyIQ

A telecommunications company collects network usage data throughout each day at a rate of several thousand data points each second. The company runs an application to process the usage data in real time. The company aggregates and stores the data in an Amazon Aurora DB instance.

Sudden drops in network usage usually indicate a network outage. The company must be able to identify sudden drops in network usage so the company can take immediate remedial actions.

Which solution will meet this requirement with the LEAST latency?

A.Create an AWS Lambda function to query Aurora for drops in network usage. Use Amazon EventBridge to automatically invoke the Lambda function every minute.

B.Modify the processing application to publish the data to an Amazon Kinesis data stream. Create an Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) application to detect drops in network usage.

C.Replace the Aurora database with an Amazon DynamoDB table. Create an AWS Lambda function to query the DynamoDB table for drops in network usage every minute. Use DynamoDB Accelerator (DAX) between the processing application and DynamoDB table.

D.Create an AWS Lambda function within the Database Activity Streams feature of Aurora to detect drops in network usage.

Answer: B

Explanation:

Here's a detailed justification for why option B is the best solution for real-time network outage detection, emphasizing low latency:

Option B utilizes a stream processing approach, specifically Amazon Kinesis Data Streams and Amazon Managed Service for Apache Flink (Kinesis Data Analytics). Kinesis Data Streams ingest the high-volume, real-time data. Flink is designed for real-time analytics on streaming data. This allows for immediate analysis as data arrives, enabling the detection of sudden drops in network usage with minimal delay. Flink applications can continuously monitor the data stream and trigger alerts upon detecting a significant decrease.

Option A, while using Lambda, relies on querying Aurora periodically (every minute). This polling approach introduces a minimum latency of one minute, making it unsuitable for "immediate" action as required. Additionally, querying a database for anomaly detection puts extra load on the database which isn't optimized for real-time stream analysis.

Option C replaces Aurora with DynamoDB and introduces DAX. While DynamoDB is fast, periodic querying via Lambda still introduces latency, and DAX only caches reads. The underlying architectural limitation of the polling approach remains. Furthermore, migrating from a relational database to a NoSQL database like

DynamoDB might necessitate significant application rework.

Option D suggests using Database Activity Streams with Lambda within Aurora. While Database Activity Streams is useful for auditing, it's not inherently designed for real-time anomaly detection on data flowing into the database. It primarily focuses on activity surrounding the database operations themselves. Furthermore, processing the data within the database can affect the overall DB performance. Kinesis/Flink is specifically built to do that without impacting the db.

Stream processing offers the lowest latency because it analyzes data as it's generated, unlike solutions that rely on periodic queries or batch processing. Flink's ability to process data in-flight makes it ideal for identifying patterns and anomalies in real-time. This directly addresses the requirement of detecting sudden drops and triggering "immediate remedial actions."

Therefore, by continuously analyzing the stream of network usage data with Flink, option B provides the fastest detection and allows the telecommunications company to respond to network outages with minimal delay.

Further Research:

Amazon Kinesis Data Streams: https://aws.amazon.com/kinesis/data-streams/
Amazon Managed Service for Apache Flink: https://aws.amazon.com/flink/
Real-Time Processing: https://docs.aws.amazon.com/whitepapers/latest/real-time-data-processing/real-time-data-processing.html

Question: 121 CertyIQ

A data engineer is processing and analyzing multiple terabytes of raw data that is in Amazon S3. The data engineer needs to clean and prepare the data. Then the data engineer needs to load the data into Amazon Redshift for analytics.

The data engineer needs a solution that will give data analysts the ability to perform complex queries. The solution must eliminate the need to perform complex extract, transform, and load (ETL) processes or to manage infrastructure.

Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon EMR to prepare the data. Use AWS Step Functions to load the data into Amazon Redshift. Use Amazon QuickSight to run queries.

B.Use AWS Glue DataBrew to prepare the data. Use AWS Glue to load the data into Amazon Redshift. Use Amazon Redshift to run queries.

C.Use AWS Lambda to prepare the data. Use Amazon Kinesis Data Firehose to load the data into Amazon Redshift. Use Amazon Athena to run queries.

D.Use AWS Glue to prepare the data. Use AWS Database Migration Service (AVVS DMS) to load the data into Amazon Redshift. Use Amazon Redshift Spectrum to run queries.

Answer: B

Explanation:

The correct answer is **B. Use AWS Glue DataBrew to prepare the data. Use AWS Glue to load the data into Amazon Redshift. Use Amazon Redshift to run queries.**

Here's why:

AWS Glue DataBrew: This service is designed for visual data preparation. It allows data engineers and data analysts to clean and normalize data without writing code, which reduces operational overhead. It directly addresses the requirement for data cleaning and preparation with minimal management effort.

https://aws.amazon.com/glue/databrew/

AWS Glue: AWS Glue is a fully managed ETL service. It provides a centralized metadata repository (AWS Glue Data Catalog) and can automatically generate ETL code to load data into Amazon Redshift. This simplifies the data loading process and further minimizes the need for manual infrastructure management. https://aws.amazon.com/glue/

Amazon Redshift: Amazon Redshift is a fully managed, petabyte-scale data warehouse service. It is optimized for complex analytical queries and provides high performance. Since the requirement is to enable complex queries, Redshift is the appropriate choice for the data warehouse. https://aws.amazon.com/redshift/

The combination of these three services fulfills the requirement of cleaning, preparing, and loading data into a data warehouse with the least operational overhead.

Here's why the other options are less suitable:

A: While Amazon EMR can process data, it involves more infrastructure management compared to Glue DataBrew. Step Functions can orchestrate tasks but doesn't directly address the data loading requirement as efficiently as Glue. QuickSight is a BI tool, not a database for running queries.
C: AWS Lambda has execution time limits and might not be suitable for processing terabytes of data. Kinesis Data Firehose is primarily for streaming data, not for batch loading into Redshift after cleaning. Athena is suitable for ad-hoc querying of data in S3, not as a replacement for a data warehouse like Redshift for complex analytical queries.
D: While AWS Glue can prepare data, DMS is mainly for migrating databases. Redshift Spectrum allows querying data in S3 from Redshift, but it doesn't address the initial data preparation requirement and isn't ideal as the primary query engine in this scenario, as the data needs to be in Redshift for optimal query performance. Using Redshift for analytical queries provides better performance.

Question: 122 CertyIQ

A company uses an AWS Lambda function to transfer files from a legacy SFTP environment to Amazon S3 buckets. The Lambda function is VPC enabled to ensure that all communications between the Lambda function and other AVS services that are in the same VPC environment will occur over a secure network.

The Lambda function is able to connect to the SFTP environment successfully. However, when the Lambda function attempts to upload files to the S3 buckets, the Lambda function returns timeout errors. A data engineer must resolve the timeout issues in a secure way.

Which solution will meet these requirements in the MOST cost-effective way?

A.Create a NAT gateway in the public subnet of the VPC. Route network traffic to the NAT gateway.
B.Create a VPC gateway endpoint for Amazon S3. Route network traffic to the VPC gateway endpoint.
C.Create a VPC interface endpoint for Amazon S3. Route network traffic to the VPC interface endpoint.
D.Use a VPC internet gateway to connect to the internet. Route network traffic to the VPC internet gateway.

Answer: B

Explanation:

The problem describes a Lambda function within a VPC that can connect to an SFTP server but times out when uploading to S3. The requirement is to resolve the timeout issue securely and cost-effectively.

Option B, creating a VPC gateway endpoint for S3, is the most appropriate solution. Gateway endpoints are virtual devices that allow direct, private connectivity to S3 from within your VPC without traversing the internet. This ensures a secure connection as traffic remains within the AWS network. Furthermore, gateway endpoints are free of charge, contributing to cost-effectiveness. Traffic to S3 is routed directly through the

endpoint using a route table entry in the VPC subnet where the Lambda function resides.

Option A, creating a NAT gateway, would also provide internet access, allowing the Lambda function to reach S3. However, it's more expensive than a gateway endpoint, as NAT gateways incur hourly charges and data processing fees. While secure, it's not the most cost-effective.

Option C, creating a VPC interface endpoint (powered by PrivateLink), also establishes a private connection to S3. However, interface endpoints are more complex and costly than gateway endpoints. They involve additional charges based on availability zone association and data processing. Gateway endpoints are purpose-built for S3 and DynamoDB.

Option D, using an internet gateway, provides internet access to the Lambda function. This is the least secure option, as traffic to S3 would traverse the public internet. While it may be cheaper than a NAT gateway if usage is low, it introduces unnecessary security risks.

Therefore, a gateway endpoint offers the best balance of security, cost-effectiveness, and simplicity for enabling private S3 access from a VPC-enabled Lambda function.

Supporting links:

VPC Endpoints
Gateway Endpoints for S3
Interface Endpoints (AWS PrivateLink)
NAT Gateways

Question: 123 CertyIQ

A company reads data from customer databases that run on Amazon RDS. The databases contain many inconsistent fields. For example, a customer record field that iPnamed place_id in one database is named location_id in another database. The company needs to link customer records across different databases, even when customer record fields do not match.

Which solution will meet these requirements with the LEAST operational overhead?

A.Create a provisioned Amazon EMR cluster to process and analyze data in the databases. Connect to the Apache Zeppelin notebook. Use the FindMatches transform to find duplicate records in the data.

B.Create an AWS Glue crawler to craw the databases. Use the FindMatches transform to find duplicate records in the data. Evaluate and tune the transform by evaluating the performance and results.

C.Create an AWS Glue crawler to craw the databases. Use Amazon SageMaker to construct Apache Spark ML pipelines to find duplicate records in the data.

D.Create a provisioned Amazon EMR cluster to process and analyze data in the databases. Connect to the Apache Zeppelin notebook. Use an Apache Spark ML model to find duplicate records in the data. Evaluate and tune the model by evaluating the performance and results.

Answer: B

Explanation:

The correct answer is B. Here's a detailed justification:

The core requirement is to link customer records across disparate databases with inconsistent field names, necessitating a solution for data integration and deduplication with minimal operational overhead. AWS Glue, specifically leveraging the FindMatches transform, offers the most suitable approach.

Option B utilizes AWS Glue crawlers to automatically discover the schema of the customer databases in Amazon RDS. This eliminates manual schema definition, reducing operational overhead. The FindMatches transform within AWS Glue is designed specifically for record linkage and deduplication tasks. It uses

machine learning algorithms under the hood to identify records representing the same entity, even when field names differ or have variations. This allows Glue to normalize and standardize data from the customer databases. Crucially, it has built-in capabilities to evaluate performance and tune the transform, making it easier to improve accuracy.

Options A and D involve setting up and managing an Amazon EMR cluster, which is significantly more complex than using Glue's serverless capabilities. EMR requires provisioning, configuration, and ongoing maintenance, increasing operational overhead substantially. While EMR can certainly perform the necessary data transformation, it is an overly complex solution for this particular use case.

Option C proposes using Amazon SageMaker and Apache Spark ML pipelines. While SageMaker is powerful for machine learning, it adds an extra layer of complexity compared to Glue's built-in FindMatches transform. Setting up SageMaker pipelines requires expertise in machine learning model development and deployment. Glue's FindMatches is purpose built for this, meaning it is faster to implement and requires less tuning.

AWS Glue is a serverless data integration service, meaning no infrastructure management is required. This significantly reduces operational overhead, aligning with the problem's requirement. The FindMatches transform encapsulates the complexities of record linkage, simplifying the process for the user.

Therefore, AWS Glue with FindMatches is the most efficient and least operationally intensive solution.

Authoritative Links:

AWS Glue: https://aws.amazon.com/glue/
AWS Glue FindMatches: https://docs.aws.amazon.com/glue/latest/dg/deduplication-findmatches.html

Question: 124 CertyIQ

A finance company receives data from third-party data providers and stores the data as objects in an Amazon S3 bucket.

The company ran an AWS Glue crawler on the objects to create a data catalog. The AWS Glue crawler created multiple tables. However, the company expected that the crawler would create only one table.

The company needs a solution that will ensure the AVS Glue crawler creates only one table.

Which combination of solutions will meet this requirement? (Choose two.)

A.Ensure that the object format, compression type, and schema are the same for each object.

B.Ensure that the object format and schema are the same for each object. Do not enforce consistency for the compression type of each object.

C.Ensure that the schema is the same for each object. Do not enforce consistency for the file format and compression type of each object.

D.Ensure that the structure of the prefix for each S3 object name is consistent.

E.Ensure that all S3 object names follow a similar pattern.

Answer: AD

Explanation:

The correct answer is AD. Here's why:

Option A: Ensure that the object format, compression type, and schema are the same for each object. AWS Glue crawlers infer schema and table structure based on the data present in S3. For the crawler to combine multiple files into a single table, it needs a consistent schema, file format (e.g., CSV, Parquet, JSON), and compression type (e.g., GZIP, Snappy). If these elements differ, the crawler will interpret each group of similar

files as a separate table. A consistent schema allows Glue to understand that all files contain the same data fields in the same order, and the matching file format and compression ensure that the data can be read consistently.

Option D: Ensure that the structure of the prefix for each S3 object name is consistent. AWS Glue crawlers can be configured to target specific S3 prefixes. A consistent prefix structure allows the crawler to target all the relevant files for a single table. If the S3 objects are scattered across different prefixes or have very different prefix structures, the crawler may not be able to group them effectively into a single table, particularly when using partitioning. By keeping the prefix structure consistent, you guide the crawler towards treating all files with that prefix structure as belonging to the same table, based on other criteria like schema and format.

Option B is incorrect because even though the schema and format are important for the crawler to recognize data types and structure, the compression type should also be the same in order for the crawler to efficiently combine all files.

Option C is incorrect because the file format is important for the crawler to recognize and efficiently combine files into a single table.

Option E is incorrect because the name pattern is not as important as the prefix. It helps in identifying similar files but isn't crucial as schema and format.

Supporting Links:

AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
Populating the AWS Glue Data Catalog: https://docs.aws.amazon.com/glue/latest/dg/populate-data-catalog.html

Question: 125 CertyIQ

An application consumes messages from an Amazon Simple Queue Service (Amazon SQS) queue. The application experiences occasional downtime. As a result of the downtime, messages within the queue expire and are deleted after 1 day. The message deletions cause data loss for the application.

Which solutions will minimize data loss for the application? (Choose two.)

A.Increase the message retention period
B.Increase the visibility timeout.
C.Attach a dead-letter queue (DLQ) to the SQS queue.
D.Use a delay queue to delay message delivery
E.Reduce message processing time.

Answer: AC

Explanation:

The correct answer is **A. Increase the message retention period** and **C. Attach a dead-letter queue (DLQ) to the SQS queue**.

Let's break down why these solutions are effective in preventing data loss when an application consuming from an SQS queue experiences downtime and messages are expiring:

A. Increase the message retention period: SQS has a configurable message retention period, which determines how long messages remain in the queue if they are not deleted by a consumer. The default is 4 days, but it can be set from 1 minute to 14 days. The problem statement indicates that messages are expiring after 1 day due to downtime, leading to data loss. By increasing the retention period to, say, 7 days or more,

you give the application a longer window to recover from downtime and process the messages before they are automatically deleted by SQS. This doesn't address the underlying downtime issue, but it provides a buffer against data loss due to expiration.

C. Attach a dead-letter queue (DLQ) to the SQS queue: A DLQ is a separate SQS queue where messages that cannot be processed successfully are sent. In this scenario, messages are expiring and being deleted, but attaching a DLQ allows messages that would have expired to be moved to the DLQ before they expire. You configure the main queue (the "source" queue) to send messages to the DLQ after a certain number of unsuccessful processing attempts or after exceeding the retention period. Even if the application is down for longer than the initial retention period of the source queue, the messages will still be safely stored in the DLQ. After the application recovers, it can then process the messages from the DLQ. This guarantees that no data is permanently lost due to the initial application downtime.

Let's examine why the other options are not the best solutions:

B. Increase the visibility timeout: The visibility timeout is the amount of time a message is hidden from other consumers after one consumer receives it. Increasing this timeout only helps if the processing of the message is taking longer than expected. It doesn't prevent messages from expiring if the application is down and not consuming any messages at all. If the application is down, no consumer can retrieve the message to begin with; increasing visibility timeout will not keep the message from eventually expiring.
D. Use a delay queue to delay message delivery: Delay queues delay the initial delivery of messages. This is useful if you want to delay processing for a specific amount of time (e.g., to schedule a job). It doesn't address the problem of messages expiring due to application downtime. Delaying the initial delivery will just delay the potential for expiration, not solve the problem when the application is offline.
E. Reduce message processing time: Reducing message processing time is a general best practice, but it doesn't directly solve the problem of messages expiring when the application is down and unable to process them. Reducing message processing time does not prevent expiration, but it can lessen the amount of messages that are in-flight at any given time. It will not prevent data loss due to downtime.

Authoritative Links:

SQS Message Retention Period:
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-configure-message-retention-period.html
SQS Dead-Letter Queues:
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html
SQS Visibility Timeout:
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html

Question: 126 CertyIQ

A company is creating near real-time dashboards to visualize time series data. The company ingests data into Amazon Managed Streaming for Apache Kafka (Amazon MSK). A customized data pipeline consumes the data. The pipeline then writes data to Amazon Keyspaces (for Apache Cassandra), Amazon OpenSearch Service, and Apache Avro objects in Amazon S3.

Which solution will make the data available for the data visualizations with the LEAST latency?

A.Create OpenSearch Dashboards by using the data from OpenSearch Service.

B.Use Amazon Athena with an Apache Hive metastore to query the Avro objects in Amazon S3. Use Amazon Managed Grafana to connect to Athena and to create the dashboards.

C.Use Amazon Athena to query the data from the Avro objects in Amazon S3. Configure Amazon Keyspaces as the data catalog. Connect Amazon QuickSight to Athena to create the dashboards.

D.Use AWS Glue to catalog the data. Use S3 Select to query the Avro objects in Amazon S3. Connect Amazon QuickSight to the S3 bucket to create the dashboards.

Answer: A

Explanation:

The optimal solution for near real-time data visualization with minimal latency involves utilizing a combination of services designed for speed and direct integration.

Option A, creating OpenSearch Dashboards using data from OpenSearch Service, presents the lowest latency path. The data pipeline already streams data into OpenSearch Service, which is specifically designed for fast indexing and searching of large datasets. OpenSearch Dashboards directly integrates with OpenSearch Service, providing near real-time visualization capabilities without introducing additional query layers or data transformations. This direct connection and optimized indexing ensure visualizations are updated quickly, making it ideal for near real-time dashboards.

Options B, C, and D involve using Amazon Athena or S3 Select to query Avro objects in Amazon S3. These services, while powerful for ad-hoc querying and data analysis, introduce significant latency compared to directly querying OpenSearch Service. Athena requires scanning potentially large Avro files, which adds overhead. Additionally, connecting Amazon Managed Grafana or QuickSight to Athena introduces further layers of data processing, impacting performance and making them less suitable for near real-time visualization requirements. Although AWS Glue cataloging can improve query performance, it doesn't address the inherent latency associated with querying data at rest in S3. S3 select has similar performance issues to Athena. Amazon Keyspaces, while low latency, is not directly used for visualization.

Therefore, leveraging the existing OpenSearch Service data ingestion with OpenSearch Dashboards provides the most direct and efficient path for creating low-latency, near real-time visualizations.

Supporting links:

Amazon OpenSearch Service: https://aws.amazon.com/opensearch-service/
OpenSearch Dashboards: https://opensearch.org/docs/latest/dashboards/index/

Question: 127 CertyIQ

A company stores petabytes of data in thousands of Amazon S3 buckets in the S3 Standard storage class. The data supports analytics workloads that have unpredictable and variable data access patterns.

The company does not access some data for months. However, the company must be able to retrieve all data within milliseconds. The company needs to optimize S3 storage costs.

Which solution will meet these requirements with the LEAST operational overhead?

A.Use S3 Storage Lens standard metrics to determine when to move objects to more cost-optimized storage classes. Create S3 Lifecycle policies for the S3 buckets to move objects to cost-optimized storage classes. Continue to refine the S3 Lifecycle policies in the future to optimize storage costs.

B.Use S3 Storage Lens activity metrics to identify S3 buckets that the company accesses infrequently. Configure S3 Lifecycle rules to move objects from S3 Standard to the S3 Standard-Infrequent Access (S3 Standard-IA) and S3 Glacier storage classes based on the age of the data.

C.Use S3 Intelligent-Tiering. Activate the Deep Archive Access tier.

D.Use S3 Intelligent-Tiering. Use the default access tier.

Answer: D

Explanation:

The correct answer is D: Use S3 Intelligent-Tiering. Use the default access tier. Here's why:

S3 Intelligent-Tiering is designed to automatically optimize storage costs by moving data between frequent, infrequent, and archive access tiers based on changing access patterns. This directly addresses the company's need to optimize storage costs for data with unpredictable access patterns without manual intervention.

Option A requires manual analysis using S3 Storage Lens and creation of S3 Lifecycle policies. While it can achieve cost optimization, it introduces significant operational overhead through ongoing monitoring and policy adjustments. S3 Storage Lens primarily provides insights into storage usage and activity trends, aiding in identifying candidates for tiering, but doesn't automate the process.

Option B suggests moving data to S3 Standard-IA and S3 Glacier based on age. This approach lacks the intelligence to react to unpredictable access patterns. Glacier provides the lowest cost storage, but does not fulfill the requirement of millisecond retrieval. S3 Standard-IA is not ideal for data that's occasionally accessed and the lifecycle policy based on age might prematurely archive active data or fail to archive inactive data, leading to suboptimal cost savings.

Option C activates the Deep Archive Access tier within S3 Intelligent-Tiering. However, Deep Archive is optimized for long-term archiving where retrieval times of hours are acceptable. The company requires millisecond retrieval, which Deep Archive does not support. The default access tier in S3 Intelligent-Tiering provides automatic tiering between Frequent and Infrequent access tiers, ensuring data that is accessed less frequently is moved to a cheaper tier while still being available within milliseconds when needed. No additional settings are required to meet the company's requirement.

Therefore, using S3 Intelligent-Tiering with its default access tier offers the least operational overhead while automatically optimizing storage costs and providing millisecond retrieval times, perfectly aligning with the company's needs.

Supporting Links:

S3 Intelligent-Tiering
S3 Storage Classes
S3 Lifecycle Management

Question: 128 CertyIQ

A media company wants to use Amazon OpenSearch Service to analyze rea-time data about popular musical artists and songs. The company expects to ingest millions of new data events every day. The new data events will arrive through an Amazon Kinesis data stream. The company must transform the data and then ingest the data into the OpenSearch Service domain.

Which method should the company use to ingest the data with the LEAST operational overhead?

A.Use Amazon Kinesis Data Firehose and an AWS Lambda function to transform the data and deliver the transformed data to OpenSearch Service.

B.Use a Logstash pipeline that has prebuilt filters to transform the data and deliver the transformed data to OpenSearch Service.

C.Use an AWS Lambda function to call the Amazon Kinesis Agent to transform the data and deliver the transformed data OpenSearch Service.

D.Use the Kinesis Client Library (KCL) to transform the data and deliver the transformed data to OpenSearch Service.

Answer: A

Explanation:

The best approach to ingest and transform data from a Kinesis data stream into an OpenSearch Service domain with minimal operational overhead is using Kinesis Data Firehose with an AWS Lambda function.

Here's why:

Kinesis Data Firehose: Firehose is designed specifically for reliably loading streaming data into data stores and analytics tools. It offers built-in capabilities for buffering, batching, and compressing data, optimizing the data transfer process to OpenSearch. https://aws.amazon.com/kinesis/data-firehose/
AWS Lambda for Transformation: Firehose integrates seamlessly with Lambda, allowing you to perform custom data transformations inline. Lambda functions can be invoked by Firehose to modify the data before it's delivered to OpenSearch, fulfilling the company's requirement to transform data. https://aws.amazon.com/lambda/
Minimal Operational Overhead: Firehose is a fully managed service, which significantly reduces the operational burden. You don't need to manage servers or infrastructure. Lambda is also serverless, minimizing operational overhead associated with data transformation. This eliminates the complexities of managing and scaling data ingestion pipelines.
Scalability and Reliability: Firehose automatically scales to handle the volume of incoming data, which is crucial given the millions of daily events. It also provides built-in retry mechanisms to ensure data delivery even in the face of transient errors.

Let's analyze why other options are less suitable:

Logstash: While Logstash can transform data, setting up and managing a Logstash pipeline requires significant operational effort, including provisioning servers, configuring pipelines, and monitoring performance.
Kinesis Agent with Lambda: Kinesis Agent is typically used for streaming data from servers, not for transforming data from a Kinesis stream. Using Lambda to call an agent adds unnecessary complexity.
Kinesis Client Library (KCL): KCL requires you to write and manage application code to consume and transform the data. It places the burden of managing scaling, fault tolerance, and checkpointing onto the application developer.

Question: 129 CertyIQ

A company stores customer data tables that include customer addresses in an AWS Lake Formation data lake. To comply with new regulations, the company must ensure that users cannot access data for customers who are in Canada.

The company needs a solution that will prevent user access to rows for customers who are in Canada.

Which solution will meet this requirement with the LEAST operational effort?

A.Set a row-level filter to prevent user access to a row where the country is Canada.
B.Create an IAM role that restricts user access to an address where the country is Canada.
C.Set a column-level filter to prevent user access to a row where the country is Canada.
D.Apply a tag to all rows where Canada is the country. Prevent user access where the tag is equal to “Canada”.

Answer: A

Explanation:

The correct answer is A: Set a row-level filter to prevent user access to a row where the country is Canada.

Justification:

Row-level filtering in AWS Lake Formation provides a mechanism to control access to specific rows within a

table based on certain conditions. In this scenario, the requirement is to restrict access to customer data based on the customer's country, which is a row-level attribute. By setting a row-level filter, the company can define a rule that prevents users from accessing rows where the "country" column contains the value "Canada."

Option B is incorrect because IAM roles primarily control access to AWS services and resources at a higher level and don't provide granular row-level access control within a Lake Formation data lake. IAM roles are not designed for filtering data within a database table.

Option C is incorrect because column-level filters control access to entire columns, not specific rows. While it could hide the "country" column itself, it wouldn't prevent access to other data within the same row for Canadian customers. Column-level filtering is insufficient to meet the precise requirement of row-based restriction.

Option D is incorrect because tagging individual rows is not a native functionality within Lake Formation or related AWS data lake services. There's no direct mechanism to apply tags to specific rows and subsequently use those tags for access control. While tags are useful for organizing and managing resources, they are not suitable for dynamic data filtering within a database table. Moreover, the operational overhead of manually tagging each Canadian customer record would be substantial and unsustainable.

Row-level filtering offers the most straightforward and efficient way to achieve the desired outcome with the least operational effort. It directly addresses the requirement by filtering rows based on the "country" attribute, without requiring complex IAM configurations or manual tagging.

Authoritative Links:

AWS Lake Formation Row-Level Security: https://docs.aws.amazon.com/lake-formation/latest/dg/row-filter.html
AWS Lake Formation Fine-Grained Access Control: https://aws.amazon.com/blogs/big-data/fine-grained-access-control-with-aws-lake-formation-simplifies-data-lake-security/

Question: 130 CertyIQ

A company has implemented a lake house architecture in Amazon Redshift. The company needs to give users the ability to authenticate into Redshift query editor by using a third-party identity provider (IdP).

A data engineer must set up the authentication mechanism.

What is the first step the data engineer should take to meet this requirement?

A.Register the third-party IdP as an identity provider in the configuration settings of the Redshift cluster.

B.Register the third-party IdP as an identity provider from within Amazon Redshift.

C.Register the third-party IdP as an identity provider for AVS Secrets Manager. Configure Amazon Redshift to use Secrets Manager to manage user credentials.

D.Register the third-party IdP as an identity provider for AWS Certificate Manager (ACM). Configure Amazon Redshift to use ACM to manage user credentials.

Answer: A

Explanation:

The correct first step is to register the third-party IdP as an identity provider in the configuration settings of the Redshift cluster. Amazon Redshift integrates directly with identity providers to enable federated access, eliminating the need to manage individual user credentials within Redshift itself. This allows users to authenticate using their existing IdP credentials and then access Redshift query editor based on the roles and permissions assigned to them within the IdP.

Option A is correct because Redshift needs to be aware of the third-party IdP to trust and validate authentication requests originating from it. Directly configuring the Redshift cluster to recognize the IdP is the most straightforward approach for setting up federated authentication.

Option B is partially correct, but the process usually starts outside the Redshift query editor itself, involving the Redshift cluster configuration.

Option C and D are incorrect because AWS Secrets Manager and AWS Certificate Manager are not directly involved in the initial step of registering an IdP with Redshift for federated authentication. While Secrets Manager can be used to store database credentials, and ACM manages certificates, the primary mechanism for establishing trust between Redshift and an external IdP is direct registration within the Redshift cluster settings. The subsequent configuration can use Secrets Manager, but first, the trust relationship with the IdP has to be established.

Registering the IdP enables Redshift to receive and validate authentication tokens provided by the IdP when users attempt to connect using the query editor. This initial step is crucial for enabling single sign-on (SSO) and managing user access in a centralized and secure manner.

Reference links:

Federated authentication to your Amazon Redshift cluster using IAM - AWS Blog
CREATE IAM ROLE - Amazon Redshift (Mentions the need for IAM role configuration related to IdP trust policies)

Question: 131 CertyIQ

A company currently uses a provisioned Amazon EMR cluster that includes general purpose Amazon EC2 instances. The EMR cluster uses EMR managed scaling between one to five task nodes for the company's long-running Apache Spark extract, transform, and load (ETL) job. The company runs the ETL job every day.

When the company runs the ETL job, the EMR cluster quickly scales up to five nodes. The EMR cluster often reaches maximum CPU usage, but the memory usage remains under 30%.

The company wants to modify the EMR cluster configuration to reduce the EMR costs to run the daily ETL job.

Which solution will meet these requirements MOST cost-effectively?

A.Increase the maximum number of task nodes for EMR managed scaling to 10.
B.Change the task node type from general purpose EC2 instances to memory optimized EC2 instances.
C.Switch the task node type from general purpose Re instances to compute optimized EC2 instances.
D.Reduce the scaling cooldown period for the provisioned EMR cluster.

Answer: C

Explanation:

The most cost-effective solution to address the EMR cluster's high CPU utilization and low memory utilization is to switch to compute-optimized EC2 instances (Option C). Here's why:

The problem is that the current general-purpose instances are CPU-bound. These instances offer a balance of compute, memory, and networking, which isn't ideal when CPU is the bottleneck. Since memory usage is low, investing in memory-optimized instances (Option B) would be an unnecessary expense.

Compute-optimized instances are specifically designed for compute-intensive workloads like the Spark ETL job. They offer higher CPU performance compared to general-purpose instances at a similar price point (or potentially lower, depending on instance size). By switching to these instances, the company can improve the

ETL job's performance without increasing costs as significantly as adding more nodes or switching to memory-optimized instances.

Increasing the maximum number of task nodes (Option A) would scale out the EMR cluster, but it would also increase the overall cost. This approach doesn't address the underlying issue of CPU bottleneck per instance and would consume more resources unnecessarily. Reducing the scaling cooldown period (Option D) would simply make the cluster scale up faster, it doesn't fix the bottleneck or reduce cost.

Therefore, switching to compute-optimized instances directly addresses the performance bottleneck related to CPU usage and offers the most cost-effective way to improve performance and potentially reduce costs compared to the other options.

https://aws.amazon.com/emr/features/managed-scaling/https://aws.amazon.com/ec2/instance-types/https://aws.amazon.com/ec2/pricing/

Question: 132 CertyIQ

A company uploads .csv files to an Amazon S3 bucket. The company's data platform team has set up an AWS Glue crawler to perform data discovery and to create the tables and schemas.

An AWS Glue job writes processed data from the tables to an Amazon Redshift database. The AWS Glue job handles column mapping and creates the Amazon Redshift tables in the Redshift database appropriately.

If the company reruns the AWS Glue job for any reason, duplicate records are introduced into the Amazon Redshift tables. The company needs a solution that will update the Redshift tables without duplicates.

Which solution will meet these requirements?

A.Modify the AWS Glue job to copy the rows into a staging Redshift table. Add SQL commands to update the existing rows with new values from the staging Redshift table.

B.Modify the AWS Glue job to load the previously inserted data into a MySQL database. Perform an upsert operation in the MySQL database. Copy the results to the Amazon Redshift tables.

C.Use Apache Spark's DataFrame dropDuplicates() API to eliminate duplicates. Write the data to the Redshift tables.

D.Use the AWS Glue ResolveChoice built-in transform to select the value of the column from the most recent record.

Answer: A

Explanation:

The correct answer is A. Here's a detailed justification:

Option A provides a robust and efficient method for preventing duplicate records in Amazon Redshift when rerunning an AWS Glue job. The strategy of writing to a staging table first is a common and reliable approach to handle upserts (update or insert) in data warehouses like Redshift. It minimizes the risk of data inconsistencies during the update process. Specifically, copying rows to a staging table allows you to compare the existing data in the target table with the new data in the staging table. By executing SQL UPDATE commands, you can selectively update existing rows in the target table with the corresponding values from the staging table only when necessary. This approach guarantees that the existing data is overwritten with the new data, while simultaneously preventing the introduction of any duplicate records in the Redshift table.

Option B introduces an unnecessary complexity and dependencies on an additional database (MySQL). Transferring the data into a MySQL database and then perform an upsert operation and further copy the data to Redshift will take more time and resources.

Option C, using dropDuplicates() in Spark, might remove duplicates based on all columns, which may not be the desired behavior. Furthermore, this method assumes that the entire dataset is available in Spark's memory, which could be an issue for large datasets. It's a simplistic deduplication strategy that doesn't cater to nuanced upsert logic.

Option D, using ResolveChoice, attempts to select values from the most recent record, but does not perform the necessary check to ensure the data already exists, meaning duplicates could still be introduced. It simply tries to resolve conflicting column types, not handle the potential duplicates from re-running the same job.

In conclusion, writing to a staging table and implementing UPDATE statements provides a transactional and controlled mechanism to prevent duplicates during updates in Amazon Redshift. This is the most reliable and best approach, as it ensures data consistency and integrity upon subsequent reruns of the AWS Glue job.

Reference Links:

AWS Glue: https://aws.amazon.com/glue/
Amazon Redshift: https://aws.amazon.com/redshift/
Upsert using staging tables: https://docs.aws.amazon.com/redshift/latest/dg/merge-specify-a-staging-table.html

Question: 133 CertyIQ

A company is using Amazon Redshift to build a data warehouse solution. The company is loading hundreds of files into a fact table that is in a Redshift cluster.

The company wants the data warehouse solution to achieve the greatest possible throughput. The solution must use cluster resources optimally when the company loads data into the fact table.

Which solution will meet these requirements?

A.Use multiple COPY commands to load the data into the Redshift cluster.

B.Use S3DistCp to load multiple files into Hadoop Distributed File System (HDFS). Use an HDFS connector to ingest the data into the Redshift cluster.

C.Use a number of INSERT statements equal to the number of Redshift cluster nodes. Load the data in parallel into each node.

D.Use a single COPY command to load the data into the Redshift cluster.

Answer: D

Explanation:

The most efficient way to load data into an Amazon Redshift fact table for maximum throughput and optimal resource utilization is to use a single COPY command. Redshift is designed to parallelize data loading from Amazon S3 into all compute nodes. A single COPY command enables this parallelism by automatically distributing the load across all nodes. This leverages the cluster's distributed architecture, optimizing resource utilization for the fastest possible loading speed. Multiple COPY commands (option A) would create multiple connections and might lead to resource contention and less efficient parallelism management.

Using S3DistCp and HDFS (option B) introduces unnecessary complexity and additional layers (Hadoop) without providing any performance benefit over directly loading from S3 using COPY. HDFS adds overhead, and the connection between HDFS and Redshift would still bottleneck data transfer compared to Redshift's native S3 integration.

Employing a number of INSERT statements equal to the number of Redshift nodes (option C), while attempting parallelism, is highly inefficient for large data volumes. INSERT statements are row-by-row

operations and don't take advantage of Redshift's massively parallel processing (MPP) architecture as effectively as the COPY command. The COPY command is optimized for bulk data loading. Furthermore, managing the parallelism with individual INSERT statements manually is a complex and error-prone approach.

The COPY command implicitly parallelizes the data load across the available nodes, resulting in optimal resource usage and highest throughput. Redshift automatically divides the input data from S3 into multiple slices and assigns a slice to each node in the cluster, facilitating parallel processing.

Therefore, a single COPY command utilizing Redshift's built-in parallelism and integration with S3 provides the best performance and resource utilization for loading large datasets into Redshift.

Relevant Links:

AWS Documentation on COPY command: https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html
Best Practices for Loading Data: https://docs.aws.amazon.com/redshift/latest/dg/loading-data-best-practices.html

Question: 134 CertyIQ

A company ingests data from multiple data sources and stores the data in an Amazon S3 bucket. An AWS Glue extract, transform, and load (ETL) job transforms the data and writes the transformed data to an Amazon S3 based data lake. The company uses Amazon Athena to query the data that is in the data lake.

The company needs to identify matching records even when the records do not have a common unique identifier.

Which solution will meet this requirement?

A.Use Amazon Macie pattern matching as part of the ETL job.
B.Train and use the AWS Glue PySpark Filter class in the ETL job.
C.Partition tables and use the ETL job to partition the data on a unique identifier.
D.Train and use the AWS Lake Formation FindMatches transform in the ETL job.

Answer: D

Explanation:

Here's a detailed justification for why option D is the best solution, along with supporting information and authoritative links:

The core challenge is to identify matching records across various data sources without a common unique identifier. This requires fuzzy matching or record linkage, identifying similar records based on probabilistic comparisons of various attributes.

Option D: Train and use the AWS Lake Formation FindMatches transform in the ETL job directly addresses this requirement. AWS Lake Formation's FindMatches is specifically designed for deduplication and record linkage, precisely the scenario described. It uses machine learning to learn similarity between records, even with variations in data entry or data source discrepancies. This transform creates a model that can then be used to identify and link similar records. This approach leverages probabilistic matching to handle scenarios where exact matches are unavailable. The process involves creating a training dataset, training the FindMatches transform on this dataset, and then applying the trained model to the larger dataset to identify matches.

AWS Lake Formation FindMatches documentation: https://docs.aws.amazon.com/lake-formation/latest/dg/find-matches.html

Let's examine why the other options are less suitable:

Option A: Use Amazon Macie pattern matching as part of the ETL job focuses on identifying and protecting sensitive data based on predefined patterns (e.g., credit card numbers, social security numbers). It is not designed for record linkage or identifying similar records based on multiple attribute comparisons. Macie is primarily for data security and compliance, not data integration or deduplication.

Amazon Macie documentation: https://aws.amazon.com/macie/

Option B: Train and use the AWS Glue PySpark Filter class in the ETL job allows filtering data based on defined conditions. While filtering can be useful in data processing, it is not a solution for identifying similar records without a unique identifier. PySpark filter functionality is for conditional data selection, not fuzzy matching.

AWS Glue PySpark documentation: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-transforms-filter.html

Option C: Partition tables and use the ETL job to partition the data on a unique identifier assumes the existence of a unique identifier, which the problem statement explicitly states is not available. Partitioning optimizes query performance, but is irrelevant to the task of finding matching records when there are no common keys.

In summary, AWS Lake Formation's FindMatches transform is uniquely positioned to address the core problem of identifying similar records without a common identifier by using a machine learning based approach for fuzzy matching and record linkage within the ETL process, making it the correct and most effective solution.

Question: 135 CertyIQ

A data engineer is using an AWS Glue crawler to catalog data that is in an Amazon S3 bucket. The S3 bucket contains both .csv and json files. The data engineer configured the crawler to exclude the .json files from the catalog.

When the data engineer runs queries in Amazon Athena, the queries also process the excluded .json files. The data engineer wants to resolve this issue. The data engineer needs a solution that will not affect access requirements for the .csv files in the source S3 bucket.

Which solution will meet this requirement with the SHORTEST query times?

A.Adjust the AWS Glue crawler settings to ensure that the AWS Glue crawler also excludes .json files.
B.Use the Athena console to ensure the Athena queries also exclude the .json files.
C.Relocate the .json files to a different path within the S3 bucket.
D.Use S3 bucket policies to block access to the .json files.

Answer: C

Explanation:

The correct answer is C: Relocate the .json files to a different path within the S3 bucket. Here's why:

The core problem is that Athena is querying data it shouldn't. While the Glue crawler is configured to exclude .json files, Athena queries are evidently still processing them. This indicates the query definition or the table definition in the Glue Data Catalog is not correctly filtering out these files.

Option A, adjusting the Glue crawler settings, won't solve the root issue. The crawler correctly creates the catalog but doesn't enforce what Athena queries. Athena queries the table definition created by the crawler, not the crawler settings themselves. If the table definition includes the .json files path, it'll be queried.

Option B, using the Athena console to exclude .json files, is possible via WHERE clauses or views, but this approach adds overhead to every query. This impacts query performance, conflicting with the "shortest query times" requirement. It also means manually modifying every query, which is not scalable.

Option D, using S3 bucket policies to block access, will prevent Athena (and potentially other services) from reading .json files. While effective, this approach is overly restrictive. The prompt explicitly states the solution should not affect access requirements for .csv files. Using bucket policies in this way would likely require complex policies that could inadvertently affect .csv access or be difficult to maintain.

Option C, relocating the .json files to a different path, is the most efficient and targeted solution. By moving the .json files to a separate path (e.g., s3://your-bucket/json_files/), the Athena table definition can be updated to only point to the .csv files path (e.g., s3://your-bucket/csv_files/). This ensures that Athena only processes the .csv files without requiring changes to bucket policies or query modifications. This provides the shortest query times because Athena processes less data. The Glue Crawler is then directed to the csv_files path.

Here's why this works and why it's optimal:

Data Partitioning: Separating data into different paths is a fundamental data warehousing technique. It allows for efficient querying and processing of specific datasets.
Targeted Queries: Athena and other query engines can directly access and process data from the designated path, ignoring the other irrelevant files.
Minimal Overhead: There's no need for complex WHERE clauses or views to filter data.
Clean Data Catalog: The table definition in the Glue Data Catalog accurately reflects the data schema for the .csv files without contamination from .json files.
Scalability: This approach scales easily as the data volume grows.

For further reading on data lake design and partitioning with AWS services, consider exploring these resources:

AWS Big Data Blog - Top 10 Security Best Practices for Your S3 Data Lake: https://aws.amazon.com/blogs/big-data/top-10-security-best-practices-for-your-s3-data-lake/ (While this article focuses on security, it implicitly touches on data organization)
AWS Documentation - Creating a Table Manually: https://docs.aws.amazon.com/athena/latest/ug/create-table.html (Understanding table creation in Athena helps in appreciating how the table location affects query behavior)
AWS Documentation - AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html (Understanding Glue crawlers helps in appreciating how crawler exclusions work, and how the exclusion here is insufficient for the Athena use case)

Question: 136 CertyIQ

A data engineer set up an AWS Lambda function to read an object that is stored in an Amazon S3 bucket. The object is encrypted by an AWS KMS key.

The data engineer configured the Lambda function's execution role to access the S3 bucket. However, the Lambda function encountered an error and failed to retrieve the content of the object.

What is the likely cause of the error?

A.The data engineer misconfigured the permissions of the S3 bucket. The Lambda function could not access the object.

B.The Lambda function is using an outdated SDK version, which caused the read failure.

C.The S3 bucket is located in a different AWS Region than the Region where the data engineer works. Latency issues caused the Lambda function to encounter an error.

D.The Lambda function's execution role does not have the necessary permissions to access the KMS key that

can decrypt the S3 object.

Answer: D

Explanation:

The error arises when a Lambda function, configured to read a KMS-encrypted S3 object, fails despite having S3 bucket access through its execution role. The most likely cause is insufficient KMS key permissions.

Here's a detailed explanation:

1. **S3 Encryption with KMS:** When an object is encrypted using KMS in S3, accessing it requires more than just S3 bucket permissions. The calling entity (in this case, the Lambda function) must also be authorized to use the KMS key to decrypt the object.
2. **Lambda Execution Role:** The Lambda execution role defines the permissions granted to the Lambda function. It's crucial that this role includes the necessary permissions to interact with all AWS resources required by the function.
3. **Missing KMS Permissions:** Even if the Lambda function's execution role has permissions to read objects from the S3 bucket, it also needs permission to use the KMS key to decrypt the object. Without this KMS permission, the Lambda function will be denied access to the decrypted content.
4. **KMS Key Policy:** The KMS key policy is a resource-based policy that controls who can use the KMS key. The Lambda function's execution role must be explicitly granted permission in the KMS key policy to kms:Decrypt or kms:GenerateDataKey (depending on how encryption was done).
5. **Why other options are less likely:**

   A (S3 Bucket Permissions): The question states that the Lambda function's execution role does have access to the S3 bucket. Therefore, misconfigured S3 bucket permissions are less probable.
   B (Outdated SDK): While an outdated SDK can cause issues, it's less likely than a missing permission, especially since KMS encryption requires explicit permission management. SDK issues typically manifest differently.
   C (Region Issues): Region mismatches can lead to errors, but the problem description focuses on permissions, making a permissions issue within the same region the more probable cause. Cross-region access also typically requires specific configuration steps, and the problem description doesn't indicate such setup.
6. **The correct solution:** To resolve this issue, the data engineer needs to modify the KMS key policy associated with the encryption key. The policy should grant the Lambda function's execution role (arn:aws:iam::<account-id>:role/<lambda-execution-role-name>) permission to use the KMS key for decryption using kms:Decrypt.

In summary, even with the correct S3 bucket permissions, the Lambda function will fail to retrieve the content of the KMS-encrypted S3 object if the execution role lacks the necessary KMS permissions. The KMS key policy must be updated to grant the Lambda function's execution role permission to decrypt the object.

Authoritative Links:

AWS KMS Key Policies
Using AWS Lambda with Amazon S3
Encrypting and Decrypting Data with AWS KMS

Question: 137

A data engineer has implemented data quality rules in 1,000 AWS Glue Data Catalog tables. Because of a recent change in business requirements, the data engineer must edit the data quality rules.

How should the data engineer meet this requirement with the LEAST operational overhead?

A.Create a pipeline in AWS Glue ETL to edit the rules for each of the 1,000 Data Catalog tables. Use an AWS Lambda function to call the corresponding AWS Glue job for each Data Catalog table.

B.Create an AWS Lambda function that makes an API call to AWS Glue Data Quality to make the edits.

C.Create an Amazon EMR cluster. Run a pipeline on Amazon EMR that edits the rules for each Data Catalog table. Use an AWS Lambda function to run the EMR pipeline.

D.Use the AWS Management Console to edit the rules within the Data Catalog.

Answer: B

Explanation:

The best approach to editing data quality rules in 1,000 AWS Glue Data Catalog tables with the least operational overhead is option B: creating an AWS Lambda function that makes an API call to AWS Glue Data Quality.

Here's why:

Scalability and Automation: Lambda functions are designed to be highly scalable and can be easily triggered to process multiple tables concurrently. This approach avoids manual intervention and automates the editing process.
Minimal Infrastructure Management: Lambda is a serverless compute service, which means there's no need to provision or manage any servers. This drastically reduces operational overhead compared to options A and C, which involve managing AWS Glue ETL jobs or an Amazon EMR cluster.
Direct Integration with AWS Glue Data Quality: Lambda can directly interact with the AWS Glue Data Quality API to programmatically edit the rules. This avoids the need for ETL jobs or complex pipelines, simplifying the process.
Cost-Effectiveness: Lambda's pay-per-execution pricing model makes it a cost-effective solution for this task, as you only pay for the actual compute time used to edit the rules.
Maintainability: A single Lambda function that edits the data quality rules is easier to maintain and update compared to managing multiple AWS Glue ETL jobs or an EMR pipeline.
Efficiency: Calling the Data Quality API directly is a streamlined and efficient way to make the changes, as opposed to methods that require intermediate processing or data movement.

Options A and C involve more complex infrastructure management and are less efficient. Option D, using the AWS Management Console, would be impractical for 1,000 tables due to the immense manual effort required.

Relevant Cloud Computing Concepts:

Serverless Computing: Utilizing AWS Lambda eliminates the need for server management, reducing operational overhead.
API Integration: Directly interacting with the AWS Glue Data Quality API provides a programmatic and efficient way to manage data quality rules.
Automation: Automating the editing process through a Lambda function ensures consistency and reduces manual effort.

Authoritative Links for Further Research:

AWS Lambda: https://aws.amazon.com/lambda/
AWS Glue Data Quality: https://aws.amazon.com/glue/features/data-quality/
AWS Glue API Reference: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api.html

Question: 138 CertyIQ

Two developers are working on separate application releases. The developers have created feature branches named Branch A and Branch B by using a GitHub repository's master branch as the source.

The developer for Branch A deployed code to the production system. The code for Branch B will merge into a master branch in the following week's scheduled application release.

Which command should the developer for Branch B run before the developer raises a pull request to the master branch?

A.git diff branchB master
git commit -m

B.git pull master

C.git rebase master

D.git fetch -b master

Answer: C

Explanation:

Here's a detailed justification for why the correct answer is C (git rebase master), along with explanations of why the other options are less suitable:

Why git rebase master is the best option:

The key here is ensuring Branch B's code is up-to-date with the changes introduced by Branch A's deployment to the master branch before creating a pull request. git rebase master achieves this. Rebasing replays the commits from Branch B on top of the latest state of the master branch. This essentially rewrites the history of Branch B as if it had been branched off the master after Branch A's changes were merged.

The primary benefit is a cleaner, linear history. When Branch B is eventually merged into master, it results in a straightforward fast-forward merge, avoiding a merge commit and making the project's history easier to understand and follow. Rebasing minimizes merge conflicts because any conflicts arising from the integration of Branch A's changes into Branch B are resolved before the pull request is even submitted. This makes the review process much smoother. It promotes a best practice of keeping feature branches synchronized with the latest changes in the main branch.

Why the other options are less suitable:

A. git diff branchB master; git commit -m: This is completely incorrect. git diff branchB master shows the differences between Branch B and master, but doesn't integrate those changes. The subsequent git commit would only commit the output of the diff, which isn't a meaningful way to merge or incorporate changes. This won't update Branch B with the latest code from the master.

B. git pull master: While git pull master does integrate changes from the master branch into Branch B, it does so by creating a merge commit. This results in a more complex and potentially messier history, especially if there are frequent integrations. It introduces a non-linear flow and additional nodes on the graph making it difficult to read. The primary downside to this option is that the history can be difficult to visualize.

D. git fetch -b master: git fetch master downloads the latest commits from the remote master branch but doesn't integrate them into your local Branch B. It just updates your local tracking branch for origin/master. This is only the first step in other operations such as git merge or git rebase, but is not an integration method on its own. It provides the data, but does not apply changes.

In summary:

git rebase master is the correct command because it integrates the latest changes from the master branch into Branch B, creating a clean and linear history. It reduces merge conflicts by addressing them before the pull request and facilitates a smoother review and integration process. git pull master is acceptable, but less preferable due to the merge commits. The other options are either not designed for integration or are incomplete.

Authoritative Links:

Git Rebasing: https://git-scm.com/docs/git-rebase
Git Pull vs. Git Rebase: https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Question: 139 CertyIQ

A company stores employee data in Amazon Resdshift. A table names Employee uses columns named Region ID, Department ID, and Role ID as a compound sort key.

Which queries will MOST increase the speed of query by using a compound sort key of the table? (Choose two.)

A.Select *from Employee where Region ID='North America';
B.Select *from Employee where Region ID='North America' and Department ID=20;
C.Select *from Employee where Department ID=20 and Region ID='North America';
D.Select *from Employee where Role ID=50;
E.Select *from Employee where Region ID='North America' and Role ID=50;

Answer: BE

Explanation:

The correct answer is B and E. Here's a detailed justification:

Amazon Redshift's compound sort key is most effective when queries filter data based on the prefix of the sort key. A compound sort key works by sorting the data on disk in the order of the columns defined in the sort key. In this case, the sort key is (Region ID, Department ID, Role ID).

Option B, "Select * from Employee where Region ID='North America' and Department ID=20;", leverages the compound sort key effectively. It filters by Region ID and Department ID, which are the first two components of the compound sort key. Redshift can quickly locate the relevant data blocks on disk because the data is sorted first by Region ID and then by Department ID within each region. This reduces the number of blocks that Redshift needs to scan, significantly improving query performance.

Option E, "Select * from Employee where Region ID='North America' and Role ID=50;", also benefits from the sort key. Although it skips the Department ID, it starts with the first column Region ID and also includes the Role ID. Redshift would filter using Region ID = 'North America' first, which greatly reduces the rows needed for processing, then it filter on Role ID. Without the first column from the sort key, the advantage would be diminished.

Option A, "Select *from Employee where Region ID='North America';", uses the first element of the sort key, so can leverage the sort order. However, it is a much broader filter. The query has to search based on the Region ID first.

Option C, "Select *from Employee where Department ID=20 and Region ID='North America';", uses the columns from the sort key, but in the wrong order. Redshift cannot directly use the sort order to optimize the query because it starts with Department ID, which is not the first column in the sort key. Redshift would likely perform a full table scan or rely on other optimizations.

Option D, "Select *from Employee where Role ID=50;", does not use the initial element of the sort key. The query will most likely result in a full table scan, negating any performance benefits of the compound sort key, since Role ID is the last column specified.

Therefore, options B and E are the only queries that make efficient use of the compound sort key in the Employee table and are therefore the MOST likely to increase query speed. Option B because it utilizes the first two columns of the sort key, while option E uses the first and last columns.

Further reading:

Amazon Redshift Sort Keys: https://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_data.html

Question: 140 CertyIQ

A company receives test results from testing facilities that are located around the world. The company stores the test results in millions of 1 KB JSON files in an Amazon S3 bucket. A data engineer needs to process the files, convert them into Apache Parquet format, and load them into Amazon Redshift tables. The data engineer uses AWS Glue to process the files, AWS Step Functions to orchestrate the processes, and Amazon EventBridge to schedule jobs.

The company recently added more testing facilities. The time required to process files is increasing. The data engineer must reduce the data processing time.

Which solution will MOST reduce the data processing time?

A.Use AWS Lambda to group the raw input files into larger files. Write the larger files back to Amazon S3. Use AWS Glue to process the files. Load the files into the Amazon Redshift tables.

B.Use the AWS Glue dynamic frame file-grouping option to ingest the raw input files. Process the files. Load the files into the Amazon Redshift tables.

C.Use the Amazon Redshift COPY command to move the raw input files from Amazon S3 directly into the Amazon Redshift tables. Process the files in Amazon Redshift.

D.Use Amazon EMR instead of AWS Glue to group the raw input files. Process the files in Amazon EMR. Load the files into the Amazon Redshift tables.

Answer: B

Explanation:

The optimal solution is **B. Use the AWS Glue dynamic frame file-grouping option to ingest the raw input files. Process the files. Load the files into the Amazon Redshift tables.**

Here's why:

The primary bottleneck is processing millions of small 1KB JSON files. AWS Glue, by default, processes each file individually, leading to significant overhead.

Option B directly addresses this bottleneck by utilizing the dynamic frame file-grouping functionality within AWS Glue. This feature allows Glue to combine multiple small files into larger chunks before processing, effectively reducing the number of individual processing operations. This minimizes overhead associated with starting and stopping jobs for each file.

https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-dynamic-frames.html

Option A is less efficient because it requires an additional step of using AWS Lambda to group the files, adding complexity and potential latency. While grouping helps, the file-grouping option within Glue provides a more integrated and streamlined approach.

Option C is unsuitable as it suggests loading raw JSON directly into Redshift and processing it there. Redshift

is optimized for structured data and analytic queries, not for complex data transformations from semi-structured JSON. This approach would be computationally expensive and inefficient within Redshift.

Option D involves using Amazon EMR, which is a powerful big data processing platform, but it's overkill for this scenario. EMR is better suited for complex transformations and larger datasets. Using EMR for this simple transformation would introduce unnecessary complexity and cost compared to leveraging Glue's built-in file-grouping feature. EMR also has a longer startup time compared to glue, making it inefficient for frequently processing small batches of files.

In summary, option B provides the most direct and efficient solution by optimizing the input stage of the AWS Glue job, minimizing overhead and reducing processing time without adding unnecessary complexity or cost.

Question: 141 CertyIQ

A data engineer uses Amazon Managed Workflows for Apache Airflow (Amazon MWAA) to run data pipelines in an AWS account.

A workflow recently failed to run. The data engineer needs to use Apache Airflow logs to diagnose the failure of the workflow.

Which log type should the data engineer use to diagnose the cause of the failure?

A.YourEnvironmentName-WebServer
B.YourEnvironmentName-Scheduler
C.YourEnvironmentName-DAGProcessing
D.YourEnvironmentName-Task

Answer: D

Explanation:

The correct answer is D: YourEnvironmentName-Task. Here's why:

When troubleshooting failed workflows in Amazon MWAA, the most granular and directly relevant logs are the Task logs. Task logs contain the standard output (stdout) and standard error (stderr) streams generated by each individual task instance within a DAG. If a specific task within the workflow is failing, the Task logs will provide the error messages, stack traces, and other diagnostic information necessary to pinpoint the root cause of the failure. This allows the data engineer to examine the specific commands, code, or operations that resulted in the error.

The other options are less relevant:

WebServer logs: Primarily capture information about HTTP requests to the Airflow web interface. These are useful for troubleshooting web UI issues but not for diagnosing data pipeline failures.
Scheduler logs: Focus on scheduling decisions, DAG parsing, and task queueing. They indicate whether tasks were scheduled correctly, but won't reveal the internal errors within a task execution.
DAGProcessing logs: These logs track the processing of the DAG file itself, such as DAG parsing errors or issues loading the DAG definition. While important, they are not useful for debugging failures happening during task execution.

Therefore, since the engineer needs to diagnose the failure of the workflow, examining the output and errors of the individual tasks within the workflow is the most direct path to resolution. The YourEnvironmentName-Task logs provide this specific information.

Further Reading:

Monitoring Amazon MWAA with Amazon CloudWatch Logs - AWS Documentation
Apache Airflow Documentation on Logging (Conceptual understanding of Task logging)

Question: 142 CertyIQ

A finance company uses Amazon Redshift as a data warehouse. The company stores the data in a shared Amazon S3 bucket. The company uses Amazon Redshift Spectrum to access the data that is stored in the S3 bucket. The data comes from certified third-party data providers. Each third-party data provider has unique connection details.

To comply with regulations, the company must ensure that none of the data is accessible from outside the company's AWS environment.

Which combination of steps should the company take to meet these requirements? (Choose two.)

A.Replace the existing Redshift cluster with a new Redshift cluster that is in a private subnet. Use an interface VPC endpoint to connect to the Redshift cluster. Use a NAT gateway to give Redshift access to the S3 bucket.

B.Create an AWS CloudHSM hardware security module (HSM) for each data provider. Encrypt each data provider's data by using the corresponding HSM for each data provider.

C.Turn on enhanced VPC routing for the Amazon Redshift cluster. Set up an AWS Direct Connect connection and configure a connection between each data provider and the finance company’s VPC.

D.Define table constraints for the primary keys and the foreign keys.

E.Use federated queries to access the data from each data provider. Do not upload the data to the S3 bucket. Perform the federated queries through a gateway VPC endpoint.

Answer: AC

Explanation:

Let's break down why options A and C are the correct choices to meet the finance company's requirements of securing data access to their Amazon Redshift data warehouse, while using Redshift Spectrum to query data in S3, and preventing external access.

Option A: Enhanced VPC Routing with Interface Endpoint and NAT Gateway

This option leverages the security and isolation provided by Amazon VPC. By placing the Redshift cluster within a private subnet, you prevent direct internet access to it. Enhanced VPC routing forces all traffic between Redshift and other AWS services (like S3 in this case) to go through the VPC. An interface VPC endpoint for Redshift allows secure and private connectivity between applications within the VPC and the Redshift data warehouse without traversing the public internet. It creates a private network interface in your VPC that is associated with Redshift, essentially extending your private network to include Redshift. The NAT gateway allows the Redshift cluster, which resides in a private subnet, to initiate outbound connections to S3 to access the data for Redshift Spectrum queries without making the Redshift cluster publicly accessible.

This combination ensures all traffic to and from Redshift stays within the company's controlled VPC environment.

Option C: Enhanced VPC Routing with Direct Connect

This option builds upon the security of VPC routing by also creating secure connectivity to the data providers. Enhanced VPC routing, as explained above, is essential. AWS Direct Connect establishes a dedicated network connection from the finance company's on-premises network to AWS, bypassing the public internet. By configuring a connection between each third-party data provider and the finance company's VPC, the data providers can privately and securely transfer data into the finance company's S3 bucket and thus make the data available to Redshift Spectrum. This direct connection eliminates the need to expose data over the public internet. By using a VPN or Direct Connect, the company is able to create an access to the providers' data without having to go through any external access to the AWS environment.

Why other options are incorrect:

B: While AWS CloudHSM is a valid option for encryption, it does not address the core requirement of preventing access from outside the company's AWS environment. Each data provider has unique connection details, and they should be configured to only communicate to the company AWS environment
D: Table constraints (primary keys, foreign keys) are for data integrity and do not enforce network security.
E: Federated queries, while useful, don't prevent data from being accessed outside the company's AWS environment if they are exposed to the public internet.

Authoritative Links:

Amazon Redshift Enhanced VPC Routing: https://docs.aws.amazon.com/redshift/latest/mgmt/enhanced-vpc-routing.html
Amazon VPC Endpoints: https://docs.aws.amazon.com/vpc/latest/privatelink/vpc-endpoints.html
AWS Direct Connect: https://aws.amazon.com/directconnect/
NAT Gateway: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

Question: 143 CertyIQ

Files from multiple data sources arrive in an Amazon S3 bucket on a regular basis. A data engineer wants to ingest new files into Amazon Redshift in near real time when the new files arrive in the S3 bucket.

Which solution will meet these requirements?

A.Use the query editor v2 to schedule a COPY command to load new files into Amazon Redshift.

B.Use the zero-ETL integration between Amazon Aurora and Amazon Redshift to load new files into Amazon Redshift.

C.Use AWS Glue job bookmarks to extract, transform, and load (ETL) load new files into Amazon Redshift.

D.Use S3 Event Notifications to invoke an AWS Lambda function that loads new files into Amazon Redshift.

Answer: D

Explanation:

Here's a detailed justification for why option D is the correct answer:

The problem requires near real-time ingestion of files from S3 into Redshift. The most suitable solution is option D, leveraging S3 Event Notifications and a Lambda function. S3 Event Notifications allow you to trigger an event (like the creation of a new object) in S3 and send a notification to another AWS service. In this scenario, the notification is sent to an AWS Lambda function. The Lambda function, upon receiving the notification, will execute code to load the new file directly into Amazon Redshift. This approach offers near real-time processing, triggered directly by the file's arrival in S3.

Option A, using the query editor to schedule COPY commands, is not suitable for near real-time ingestion. Scheduled tasks run periodically, which doesn't guarantee immediate processing of new files.

Option B, zero-ETL integration between Aurora and Redshift, is irrelevant. This integration deals with replicating changes from Aurora databases to Redshift data warehouses, not the ingestion of files from S3.

Option C, AWS Glue job bookmarks, are also not optimal for near real-time ingestion. While Glue jobs can handle ETL processes, relying solely on bookmarks will not trigger immediately when a new file arrives. You'd still need a mechanism to start the Glue job upon file arrival, making it less responsive than the S3 event-driven approach. Furthermore, the Glue job bookmarks are designed to prevent reprocessing of the same data, but the core problem here is triggering the processing, not preventing re-processing.

In summary, option D provides the most direct and efficient method for near real-time ingestion of data from

S3 to Redshift because of its event-driven architecture. S3 immediately triggers processing when a new file is created, using a lightweight and scalable serverless function to load the data into Redshift.

Supporting Links:

Amazon S3 Event Notifications:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html
AWS Lambda: https://aws.amazon.com/lambda/
Amazon Redshift COPY command: https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html

Question: 144 CertyIQ

A technology company currently uses Amazon Kinesis Data Streams to collect log data in real time. The company wants to use Amazon Redshift for downstream real-time queries and to enrich the log data.

Which solution will ingest data into Amazon Redshift with the LEAST operational overhead?

A.Set up an Amazon Kinesis Data Firehose delivery stream to send data to a Redshift provisioned cluster table.

B.Set up an Amazon Kinesis Data Firehose delivery stream to send data to Amazon S3. Configure a Redshift provisioned cluster to load data every minute.

C.Configure Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to send data directly to a Redshift provisioned cluster table.

D.Use Amazon Redshift streaming ingestion from Kinesis Data Streams and to present data as a materialized view.

Answer: D

Explanation:

The best solution for ingesting real-time log data from Kinesis Data Streams into Amazon Redshift with the least operational overhead is **D. Use Amazon Redshift streaming ingestion from Kinesis Data Streams and to present data as a materialized view.**

Here's why:

Reduced Operational Overhead: Redshift streaming ingestion is designed to directly ingest data from Kinesis Data Streams without requiring intermediary services like Kinesis Data Firehose or complex ETL pipelines. This simplifies the architecture and reduces the management burden.
Real-time Analytics: Streaming ingestion allows near real-time analysis of the data. As data streams into Redshift, you can immediately query it.
Materialized Views: Presenting the data as a materialized view in Redshift further optimizes query performance. Materialized views precompute and store the results of a query, improving query speed for frequently accessed data.
Direct Integration: Redshift streaming ingestion directly integrates with Kinesis Data Streams, eliminating the need to manage data transfer configurations and potential points of failure.

Here's why other options are not ideal:

A. Kinesis Data Firehose to Redshift: While Firehose can deliver data to Redshift, it typically involves buffering and batching, which can introduce latency and require additional configuration. Firehose is more suited for data transformation and batch loading, adding complexity.
B. Kinesis Data Firehose to S3, then COPY to Redshift: This approach introduces significant operational overhead, including managing S3 buckets, configuring the COPY command to load data regularly, and potentially dealing with data format inconsistencies. This is a common ETL approach, but not the best for minimizing operational overhead and achieving real-time analysis.

C. Amazon Managed Service for Apache Flink: Flink is a powerful stream processing engine, but using it solely to move data from Kinesis to Redshift is an overkill. It requires setting up and managing a Flink application, which adds significant complexity and operational overhead.

In summary, Redshift streaming ingestion from Kinesis Data Streams, combined with materialized views, provides the simplest, most efficient, and lowest-overhead solution for real-time ingestion and analysis of log data.

Authoritative Links:

Amazon Redshift Streaming Ingestion: https://docs.aws.amazon.com/redshift/latest/dg/streaming-ingestion.html
Amazon Redshift Materialized Views: https://docs.aws.amazon.com/redshift/latest/dg/materialized-views.html

Question: 145 CertyIQ

A company maintains a data warehouse in an on-premises Oracle database. The company wants to build a data lake on AWS. The company wants to load data warehouse tables into Amazon S3 and synchronize the tables with incremental data that arrives from the data warehouse every day.

Each table has a column that contains monotonically increasing values. The size of each table is less than 50 GB. The data warehouse tables are refreshed every night between 1 AM and 2 AM. A business intelligence team queries the tables between 10 AM and 8 PM every day.

Which solution will meet these requirements in the MOST operationally efficient way?

A.Use an AWS Database Migration Service (AWS DMS) full load plus CDC job to load tables that contain monotonically increasing data columns from the on-premises data warehouse to Amazon S3. Use custom logic in AWS Glue to append the daily incremental data to a full-load copy that is in Amazon S3.

B.Use an AWS Glue Java Database Connectivity (JDBC) connection. Configure a job bookmark for a column that contains monotonically increasing values. Write custom logic to append the daily incremental data to a full-load copy that is in Amazon S3.

C.Use an AWS Database Migration Service (AWS DMS) full load migration to load the data warehouse tables into Amazon S3 every day. Overwrite the previous day's full-load copy every day.

D.Use AWS Glue to load a full copy of the data warehouse tables into Amazon S3 every day. Overwrite the previous day's full-load copy every day.

Answer: A

Explanation:

Here's a detailed justification for why option A is the most operationally efficient solution:

Option A leverages AWS Database Migration Service (DMS) for both the initial full load and subsequent incremental updates using Change Data Capture (CDC). DMS is designed for database migrations and replication, making it well-suited for this task. The full load efficiently copies the existing data to S3. The CDC functionality captures changes happening in the Oracle database (between 1 AM and 2 AM) and applies them to the S3 data lake. DMS CDC streams the data without requiring full table scans, enhancing efficiency. The monotonically increasing column provides a natural key for identifying changes.

The custom logic in AWS Glue is used to append the incremental data to the full-load copy in S3. Glue provides a serverless environment for data integration and transformation, making it ideal for appending data and managing the data lake. This approach avoids repeated full loads.

Option B, while using Glue job bookmarks, would require significant custom coding to handle the CDC-like functionality and append data, making it less operationally efficient than using DMS CDC. Glue is better

suited for transformations after CDC.

Options C and D both involve performing a full load every day. This is highly inefficient and resource-intensive, especially given the 50 GB table size. Overwriting the entire dataset daily consumes excessive bandwidth, storage, and processing power. The business intelligence team needs to query data between 10 AM and 8 PM daily. Using solutions C and D will make the data unavailable between 1 AM and 2 AM.

Therefore, the combination of DMS for full load and CDC, with Glue for final appending, offers the best balance of efficiency, automation, and scalability for the given requirements. It avoids repeated full loads and leverages specialized AWS services for their intended purposes.

Supporting Links:

AWS Database Migration Service (DMS): https://aws.amazon.com/dms/
AWS Glue: https://aws.amazon.com/glue/
AWS DMS Change Data Capture (CDC):
https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Introduction.html

Question: 146 CertyIQ

A company is building a data lake for a new analytics team. The company is using Amazon S3 for storage and Amazon Athena for query analysis. All data that is in Amazon S3 is in Apache Parquet format.

The company is running a new Oracle database as a source system in the company's data center. The company has 70 tables in the Oracle database. All the tables have primary keys. Data can occasionally change in the source system. The company wants to ingest the tables every day into the data lake.

Which solution will meet this requirement with the LEAST effort?

A.Create an Apache Sqoop job in Amazon EMR to read the data from the Oracle database. Configure the Sqoop job to write the data to Amazon S3 in Parquet format.

B.Create an AWS Glue connection to the Oracle database. Create an AWS Glue bookmark job to ingest the data incrementally and to write the data to Amazon S3 in Parquet format.

C.Create an AWS Database Migration Service (AWS DMS) task for ongoing replication. Set the Oracle database as the source. Set Amazon S3 as the target. Configure the task to write the data in Parquet format.

D.Create an Oracle database in Amazon RDS. Use AWS Database Migration Service (AWS DMS) to migrate the on-premises Oracle database to Amazon RDS. Configure triggers on the tables to invoke AWS Lambda functions to write changed records to Amazon S3 in Parquet format.

Answer: B

Explanation:

The correct answer is B because it offers the least amount of effort while fulfilling all requirements. Here's a breakdown:

AWS Glue: Glue is a fully managed ETL (Extract, Transform, Load) service designed for discovering, preparing, and integrating data. This makes it ideal for ingesting data from the Oracle database into the S3 data lake.
AWS Glue Connection: Glue Connections allow Glue jobs to connect to various data sources. This enables a seamless connection to the on-premises Oracle database.
AWS Glue Bookmark Job: Glue bookmark jobs provide incremental data ingestion capabilities. Since the data in the Oracle database changes, bookmarking allows Glue to only ingest new or updated records each day, improving efficiency.
Parquet Format: Glue supports writing data in Parquet format directly to S3.
Least Effort: Setting up a Glue connection and a bookmark job is significantly less complex and time-

consuming than setting up EMR with Sqoop, configuring DMS, or migrating the entire database to RDS and setting up triggers and Lambda functions.

Why other options are not the best choice:

A (EMR with Sqoop): While Sqoop can read from Oracle and write to S3, managing an EMR cluster adds operational overhead compared to the fully managed Glue service. Sqoop also requires more manual configuration and management than Glue's built-in incremental capabilities.
C (AWS DMS): DMS is primarily designed for database migration and continuous replication. While DMS can write to S3, it's more geared towards replicating entire tables and might not be the most efficient solution for daily incremental updates and transformations for data lake purposes.
D (RDS, DMS, Lambda): Migrating the Oracle database to RDS and implementing triggers and Lambda functions is by far the most complex and time-consuming option. It involves significant database administration and development effort. This is a costly and complex approach for simply ingesting data into a data lake.

Supporting Links:

AWS Glue Documentation: For a deep understanding of AWS Glue features and functionalities.
AWS Glue Connections: For understanding connections to different types of data sources
AWS Glue Bookmarks: For detail information on incremental data loading using job bookmarks.

Question: 147 CertyIQ

A transportation company wants to track vehicle movements by capturing geolocation records. The records are 10 bytes in size. The company receives up to 10.000 records every second. Data transmission delays of a few minutes are acceptable because of unreliable network conditions.

The transportation company wants to use Amazon Kinesis Data Streams to ingest the geolocation data. The company needs a reliable mechanism to send data to Kinesis Data Streams. The company needs to maximize the throughput efficiency of the Kinesis shards.

Which solution will meet these requirements in the MOST operationally efficient way?

A.Kinesis Agent
B.Kinesis Producer Library (KPL)
C.Amazon Kinesis Data Firehose
D.Kinesis SDK

Answer: B

Explanation:

The correct answer is B: Kinesis Producer Library (KPL). Here's why:

The primary goal is to efficiently ingest a high volume (10,000 records/second) of small records (10 bytes each) into Kinesis Data Streams, while maximizing shard throughput and tolerating minor transmission delays. KPL excels in this scenario.

KPL Aggregation and Batching: KPL aggregates multiple small records into a single Kinesis Data Streams record, significantly reducing the number of requests and thus increasing throughput. This is crucial because Kinesis Data Streams charges based on PUT requests.
Increased Throughput Efficiency: By reducing the number of PUT requests, KPL effectively reduces the number of Kinesis Data Streams records, allowing you to utilize the provisioned throughput of your shards more efficiently.
Handles Retries: KPL automatically handles retries for failed requests, increasing reliability which is

important given the unreliable network conditions. This reduces operational overhead related to error handling.
Asynchronous API: The KPL is designed to be an asynchronous API, increasing the ingestion performance.
Reduced Costs: By using aggregation, KPL reduces the amount of Kinesis Data Streams records, thus reducing cost of operating the data streams.
Kinesis Agent is less efficient: Kinesis Agent can ingest data but does not inherently provide aggregation/batching functionalities to the same extent as KPL. It's a more basic tool.
Kinesis Data Firehose is unsuitable: Firehose delivers data to destinations like S3 or Redshift, and it doesn't provide the same degree of control over shard utilization as KPL offers. Firehose's buffering is more aimed towards long term storage.
Kinesis SDK is too low level: Using the Kinesis SDK directly requires implementing aggregation, batching, and retry mechanisms yourself, which adds operational complexity.

In summary, KPL's built-in aggregation, batching, and retry capabilities make it the most operationally efficient solution for ingesting a high volume of small records into Kinesis Data Streams while maximizing shard throughput and tolerating short data transmission delays.

Authoritative Links:

Kinesis Producer Library (See "Producers" Section)
Developing Producers Using the Amazon Kinesis Producer Library

Question: 148 CertyIQ

An investment company needs to manage and extract insights from a volume of semi-structured data that grows continuously.

A data engineer needs to deduplicate the semi-structured data, remove records that are duplicates, and remove common misspellings of duplicates.

Which solution will meet these requirements with the LEAST operational overhead?

A.Use the FindMatches feature of AWS Glue to remove duplicate records.
B.Use non-Windows functions in Amazon Athena to remove duplicate records.
C.Use Amazon Neptune ML and an Apache Gremlin script to remove duplicate records.
D.Use the global tables feature of Amazon DynamoDB to prevent duplicate data.

Answer: A

Explanation:

The most efficient solution for deduplicating semi-structured data with misspellings, while minimizing operational overhead, is using AWS Glue's FindMatches feature. FindMatches is specifically designed for record linkage and deduplication tasks. It uses machine learning algorithms to identify near-duplicate records, even with variations in spelling or formatting, in a scalable and automated manner. Options B, C, and D are less suitable. Athena, while useful for querying data, requires manual SQL queries for deduplication, increasing operational overhead. While it can be done using techniques like ROW_NUMBER() and PARTITION BY, it does not inherently handle fuzzy matching or misspellings effectively. This increases the complexity of the query and requires more hands-on work to configure rules for detecting duplicates based on text similarity or string distance. Neptune ML, a graph database service, would be overkill for this scenario. Creating a graph database and running Gremlin scripts adds significant complexity and overhead compared to a dedicated deduplication tool. DynamoDB's global tables are primarily for multi-region replication and do not inherently prevent or handle duplicate data based on content matching. They only ensure data consistency across regions. The advantage of FindMatches lies in its ability to learn from data samples to

create a transformation that identifies similar records using techniques like tokenization, stemming, and fuzzy matching. Its iterative learning process allows it to accurately detect duplicates even when data is inconsistently formatted or contains misspellings, reducing the need for manual configuration and ongoing maintenance. Therefore, FindMatches is the most appropriate choice as it is purpose-built for this scenario, reducing complexity and operational burden.

Relevant Links:

AWS Glue FindMatches: https://docs.aws.amazon.com/glue/latest/dg/deduplication-findmatches.html
Amazon Athena Windows Functions: https://docs.aws.amazon.com/athena/latest/ug/window-functions.html (For alternative solutions, but not preferred here)
Amazon Neptune ML: https://aws.amazon.com/neptune/machine-learning/
Amazon DynamoDB Global Tables: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/globaltables.html

Question: 149 CertyIQ

A company is building an inventory management system and an inventory reordering system to automatically reorder products. Both systems use Amazon Kinesis Data Streams. The inventory management system uses the Amazon Kinesis Producer Library (KPL) to publish data to a stream. The inventory reordering system uses the Amazon Kinesis Client Library (KCL) to consume data from the stream. The company configures the stream to scale up and down as needed.

Before the company deploys the systems to production, the company discovers that the inventory reordering system received duplicated data.

Which factors could have caused the reordering system to receive duplicated data? (Choose two.)

A.The producer experienced network-related timeouts.
B.The stream's value for the IteratorAgeMilliseconds metric was too high.
C.There was a change in the number of shards, record processors, or both.
D.The AggregationEnabled configuration property was set to true.
E.The max_records configuration property was set to a number that was too high.

Answer: AC

Explanation:

Here's a detailed justification for why options A and C are correct regarding duplicated data in a Kinesis Data Streams consumer (inventory reordering system using KCL) when the producer (inventory management system using KPL) sends data to it:

A. The producer experienced network-related timeouts.

Explanation: The KPL guarantees at-least-once delivery. When the producer experiences a network timeout while attempting to send a record to Kinesis Data Streams, it doesn't know if the record was successfully ingested. Therefore, the KPL retries sending the record. If the original record did make it to Kinesis before the timeout, the retry results in a duplicate record being written to the stream. The KCL consumer then reads both the original and the retry, causing duplication.
Relevance: Network instability and the producer's retry mechanism introduce the possibility of duplicates, even with at-least-once guarantees.
Supporting Concept: Idempotency is a common concept in distributed systems that solves the problem of retrying a service operation (in this case, sending data to kinesis).

C. There was a change in the number of shards, record processors, or both.

Explanation: The KCL assigns shards to record processors (consumer instances). When the number of shards in a stream changes (e.g., due to scaling up or splitting shards), or the number of record processors changes (e.g., adding or removing consumer instances), the KCL needs to rebalance the shard assignment. During this rebalancing, it's possible for a record processor to temporarily process records from a shard that was previously being processed by another processor. This can lead to a record being processed twice. Also, if there is a failure of a record processor, the KCL will assign the shards to another record processor.
Relevance: The KCL handles shard assignment to ensure data from a shard is read by only one processor at a given time. However, transitions and failures can briefly violate this, causing overlap and duplication.
Supporting Concept: Shard assignment and the KCL worker model are central to Kinesis Data Streams consumer processing and must be accounted for when designing idempotent consumer applications.

Why the other options are not the primary causes of duplication:

B. The stream's value for the IteratorAgeMilliseconds metric was too high. A high IteratorAgeMilliseconds value indicates that the consumer is lagging behind in processing the data in the stream. While this is an issue (data might expire), it doesn't directly cause duplication. It only means that the consumer is reading data that is old.
D. The AggregationEnabled configuration property was set to true. Aggregation in KPL improves throughput by bundling multiple records into a single Kinesis record. While improper deaggregation on the consumer side could potentially lead to data issues, it wouldn't inherently cause duplication of whole Kinesis records.
E. The max_records configuration property was set to a number that was too high. The max_records setting in KCL controls the maximum number of records a record processor can retrieve in a single getRecords call. This setting can impact throughput and processing time, but it doesn't directly cause the consumer to process the same records multiple times.

Authoritative Links for Further Research:

Kinesis Data Streams Developer Guide: Developing Consumers with the KCL 2:
https://docs.aws.amazon.com/streams/latest/dev/developing-consumers-with-kcl2.html
Kinesis Data Streams Developer Guide: Developing Producers with the KPL:
https://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-kpl.html
AWS Blog: Building Scalable Stream Processing Applications Using Amazon Kinesis Client Library:
https://aws.amazon.com/blogs/big-data/building-scalable-stream-processing-applications-using-amazon-kinesis-client-library/
AWS Kinesis Data Streams FAQs: https://aws.amazon.com/kinesis/data-streams/faqs/ (specifically regarding data durability and delivery guarantees)

Question: 150 CertyIQ

An ecommerce company operates a complex order fulfilment process that spans several operational systems hosted in AWS. Each of the operational systems has a Java Database
Connectivity (JDBC)-compliant relational database where the latest processing state is captured.

The company needs to give an operations team the ability to track orders on an hourly basis across the entire fulfillment process.

Which solution will meet these requirements with the LEAST development overhead?

A.Use AWS Glue to build ingestion pipelines from the operational systems into Amazon Redshift Build dashboards in Amazon QuickSight that track the orders.

B.Use AWS Glue to build ingestion pipelines from the operational systems into Amazon DynamoDBuild dashboards in Amazon QuickSight that track the orders.

C.Use AWS Database Migration Service (AWS DMS) to capture changed records in the operational systems. Publish the changes to an Amazon DynamoDB table in a different AWS region from the source database. Build Grafana dashboards that track the orders.

D.Use AWS Database Migration Service (AWS DMS) to capture changed records in the operational systems. Publish the changes to an Amazon DynamoDB table in a different AWS region from the source database. Build Amazon QuickSight dashboards that track the orders.

Answer: A

Explanation:

The best solution is A because it offers the least development overhead for tracking orders on an hourly basis across multiple JDBC-compliant databases. AWS Glue can readily connect to JDBC data sources, extract, transform, and load (ETL) data into Amazon Redshift, a data warehouse optimized for analytical workloads. Redshift's columnar storage and query optimization are ideal for aggregating and analyzing order fulfillment data. Amazon QuickSight then allows for the quick creation of dashboards to visualize and track order status.

Option B is less suitable as DynamoDB, a NoSQL database, while efficient for high-velocity writes, is not as optimized for complex analytical queries needed to track orders across various stages. This would necessitate additional data transformation and potentially slower query performance.

Options C and D utilize AWS DMS for change data capture (CDC). While CDC is useful for near real-time data replication, it introduces complexity. Setting up DMS tasks and ensuring data consistency across different AWS regions increases the development overhead compared to Glue's batch-oriented ETL. Also, replicating to DynamoDB has the same disadvantages as explained in Option B. Furthermore, while both QuickSight and Grafana are viable for dashboarding, QuickSight integrates more seamlessly within the AWS ecosystem, potentially reducing initial setup time and maintenance overhead, making option D less appealing than A. Given the hourly tracking requirement, a simpler, batch-oriented ETL process with Glue to Redshift is sufficient and less complex than a near real-time CDC approach.

Therefore, the combination of AWS Glue, Amazon Redshift, and Amazon QuickSight delivers the desired functionality with minimal development effort.

Relevant links:

AWS Glue: https://aws.amazon.com/glue/
Amazon Redshift: https://aws.amazon.com/redshift/
Amazon QuickSight: https://aws.amazon.com/quicksight/
AWS Database Migration Service (DMS): https://aws.amazon.com/dms/

Question: 151 CertyIQ

A data engineer needs to use Amazon Neptune to develop graph applications.

Which programming languages should the engineer use to develop the graph applications? (Choose two.)

A.Gremlin
B.SQL
C.ANSI SQL
D.SPARQL
E.Spark SQL

Answer: AD

Explanation:

The correct answer is A and D: Gremlin and SPARQL. Amazon Neptune is a graph database service and supports graph query languages.

Gremlin: This is a graph traversal language widely used for querying and manipulating graph data. Neptune is optimized for Gremlin traversals. It allows developers to express complex graph traversals using a functional programming style. The Apache TinkerPop framework provides Gremlin support across many graph databases.

SPARQL: (SPARQL Protocol and RDF Query Language) is a query language specifically designed for Resource Description Framework (RDF) data, which represents data as a graph. Neptune supports SPARQL 1.1, enabling queries against RDF-based graphs.

SQL, ANSI SQL, and Spark SQL are relational database query languages, and while they can interact with graph data through various connectors or transformations, they are not native graph query languages supported by Neptune. Neptune is specifically designed to work with graph-structured data and is optimized for graph traversal operations using languages like Gremlin and SPARQL. Using SQL with Neptune would involve complex transformations and would not leverage Neptune's native graph querying capabilities. Therefore, using native graph query languages Gremlin and SPARQL for developing graph applications using Amazon Neptune is the appropriate solution.

Relevant links:

Amazon Neptune Documentation
Gremlin query language
SPARQL

Question: 152 CertyIQ

A mobile gaming company wants to capture data from its gaming app. The company wants to make the data available to three internal consumers of the data. The data records are approximately 20 KB in size.

The company wants to achieve optimal throughput from each device that runs the gaming app. Additionally, the company wants to develop an application to process data streams. The stream-processing application must have dedicated throughput for each internal consumer.

Which solution will meet these requirements?

A.Configure the mobile app to call the PutRecords API operation to send data to Amazon Kinesis Data Streams. Use the enhanced fan-out feature with a stream for each internal consumer.

B.Configure the mobile app to call the PutRecordBatch API operation to send data to Amazon Kinesis Data Firehose. Submit an AWS Support case to turn on dedicated throughput for the company's AWS account. Allow each internal consumer to access the stream.

C.Configure the mobile app to use the Amazon Kinesis Producer Library (KPL) to send data to Amazon Kinesis Data Firehose. Use the enhanced fan-out feature with a stream for each internal consumer.

D.Configure the mobile app to call the PutRecords API operation to send data to Amazon Kinesis Data Streams. Host the stream-processing application for each internal consumer on Amazon EC2 instances. Configure auto scaling for the EC2 instances.

Answer: A

Explanation:

The correct answer is A because it effectively addresses the requirements of optimal throughput from devices, dedicated throughput for consumers, and stream processing.

Here's a detailed justification:

Optimal Throughput: Kinesis Data Streams using the PutRecords API operation is designed for high-throughput data ingestion. PutRecords allows sending multiple data records in a single request, reducing overhead compared to sending records individually. This directly addresses the requirement for optimal

throughput from each gaming app device.

Dedicated Throughput for Consumers: The enhanced fan-out feature in Kinesis Data Streams enables each consumer to have its own dedicated throughput, independent of other consumers. This feature is critical as the question explicitly requires dedicated throughput for each of the three internal consumers. Without enhanced fan-out, consumers share the stream's read capacity, which can lead to throttling and performance degradation, which goes against the requirements.

Stream Processing Application: Kinesis Data Streams is ideal for real-time stream processing. The stream-processing application can read data from the Kinesis stream and perform transformations, analytics, or other operations as needed.

Option B is incorrect because Kinesis Data Firehose is primarily designed for loading data into data lakes, warehouses, or analytics services, not for real-time stream processing with dedicated throughput. Firehose does not inherently offer a mechanism for dedicated throughput to different consuming applications. Requesting dedicated throughput from AWS Support is not the correct solution for achieving dedicated throughput for each consumer.

Option C is incorrect because while using the Kinesis Producer Library (KPL) with Firehose does improve the throughput, Firehose does not directly support enhanced fan-out which is needed for providing dedicated throughput to internal consumers.

Option D is incorrect because while it uses Kinesis Data Streams, simply hosting the stream-processing application on EC2 instances with autoscaling doesn't guarantee dedicated throughput per consumer. Multiple consumers would still be competing for the stream's read capacity.

Supporting Links:

Amazon Kinesis Data Streams Enhanced Fan-Out: https://aws.amazon.com/kinesis/data-streams/enhanced-fan-out/
Amazon Kinesis Data Streams PutRecords: https://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecords.html
Choosing Between Kinesis Data Streams and Kinesis Data Firehose: https://aws.amazon.com/blogs/big-data/choosing-between-amazon-kinesis-data-streams-and-amazon-kinesis-data-firehose/

Question: 153 CertyIQ

A retail company uses an Amazon Redshift data warehouse and an Amazon S3 bucket. The company ingests retail order data into the S3 bucket every day.

The company stores all order data at a single path within the S3 bucket. The data has more than 100 columns. The company ingests the order data from a third-party application that generates more than 30 files in CSV format every day. Each CSV file is between 50 and 70 MB in size.

The company uses Amazon Redshift Spectrum to run queries that select sets of columns. Users aggregate metrics based on daily orders. Recently, users have reported that the performance of the queries has degraded. A data engineer must resolve the performance issues for the queries.

Which combination of steps will meet this requirement with LEAST developmental effort? (Choose two.)

A.Configure the third-party application to create the files in a columnar format.
B.Develop an AWS Glue ETL job to convert the multiple daily CSV files to one file for each day.
C.Partition the order data in the S3 bucket based on order date.
D.Configure the third-party application to create the files in JSON format.
E.Load the JSON data into the Amazon Redshift table in a SUPER type column.

Answer: AC

Explanation:

Here's a detailed justification for why options A and C are the best choices to improve query performance in this scenario, along with supporting concepts and links.

The core issue is that Amazon Redshift Spectrum's performance is suffering due to the current data storage format and organization in S3. Redshift Spectrum benefits significantly from columnar storage formats and data partitioning.

Why A is correct: Configure the third-party application to create the files in a columnar format.

Columnar storage benefits: Columnar formats like Parquet or ORC allow Redshift Spectrum to read only the columns required for a query, drastically reducing I/O and improving performance. With the current CSV format, Spectrum must scan the entire file to extract even a single column. This optimization is called "column projection".
Relevance to the scenario: The retail company is performing queries that select sets of columns and aggregations. A columnar format will allow Spectrum to avoid reading unnecessary data, directly addressing the performance bottleneck.
Reduced I/O: Columnar formats are often compressed, further reducing the amount of data that needs to be read from S3.
Least developmental effort: If possible, configuring the third-party application is simpler than building a new ETL pipeline. It avoids developing and maintaining custom code.

Why C is correct: Partition the order data in the S3 bucket based on order date.

Data partitioning benefits: Partitioning data in S3, using date as the partition key, allows Redshift Spectrum to read only the data relevant to a specific date. This technique is called "partition pruning".
Relevance to the scenario: The company aggregates metrics based on daily orders. Partitioning by date means Spectrum can efficiently target the specific partitions required for a given day's analysis.
Improved query performance: By limiting the amount of data scanned, partitioning leads to faster query execution times and reduced cost.
Cost Savings: By scanning less data, organizations can optimize their S3 data retrieval expenses.

Why B, D, and E are incorrect:

B. Develop an AWS Glue ETL job to convert the multiple daily CSV files to one file for each day. While reducing the number of files could help slightly, it doesn't address the fundamental problem of the row-based CSV format or the lack of partitioning. This option requires significant development effort for minimal gain.
D. Configure the third-party application to create the files in JSON format. JSON is still a row-oriented format, so it doesn't provide the columnar read benefits of Parquet or ORC. It might even increase the file size compared to CSV if not compressed, negatively impacting performance.
E. Load the JSON data into the Amazon Redshift table in a SUPER type column. This option involves loading the data into Redshift rather than leveraging Spectrum. It also doesn't solve the problem of scanning large amounts of unnecessary data in external storage. It defeats the purpose of using Redshift Spectrum for querying data directly in S3. The SUPER data type in Redshift is primarily for semi-structured data with varying schemas within Redshift's managed storage, not for optimizing Spectrum queries of external data.

Supporting Links:

Amazon Redshift Spectrum best practices for query performance:
https://docs.aws.amazon.com/redshift/latest/dg/c-spectrum-query.html
Optimizing Amazon S3 performance: https://aws.amazon.com/premiumsupport/knowledge-center/s3-performance/
AWS Glue Partitioning: https://docs.aws.amazon.com/glue/latest/dg/partitioning.html

In conclusion, switching to a columnar format (A) and partitioning the data by date (C) are the most effective approaches to improve Redshift Spectrum query performance with the least developmental effort, as they directly address the inefficiencies caused by the row-based CSV format and lack of data organization.

Question: 154 CertyIQ

A company stores customer records in Amazon S3. The company must not delete or modify the customer record data for 7 years after each record is created. The root user also must not have the ability to delete or modify the data.

A data engineer wants to use S3 Object Lock to secure the data.

Which solution will meet these requirements?

A.Enable governance mode on the S3 bucket. Use a default retention period of 7 years.
B.Enable compliance mode on the S3 bucket. Use a default retention period of 7 years.
C.Place a legal hold on individual objects in the S3 bucket. Set the retention period to 7 years.
D.Set the retention period for individual objects in the S3 bucket to 7 years.

Answer: B

Explanation:

The correct answer is **B. Enable compliance mode on the S3 bucket. Use a default retention period of 7 years.**

Here's a detailed justification:

S3 Object Lock is designed to prevent objects from being deleted or overwritten for a specified period. The requirement to prevent deletion/modification for 7 years aligns perfectly with this functionality. The key lies in choosing the correct mode: Governance or Compliance.

Compliance Mode is the stricter of the two modes. Once an object is locked in compliance mode, the retention settings cannot be shortened, and the object cannot be deleted by any user, including the root user. This directly addresses the requirement that the root user should also be unable to delete or modify the data during the 7-year retention period.

Governance Mode allows privileged users to override the retention settings if they have the s3:BypassGovernanceRetention permission. This does not meet the requirement of restricting the root user.

Setting a default retention period on the bucket applies the retention policy to all new objects automatically, simplifying management. Configuring retention per object would be far more cumbersome and error-prone.

Legal holds are used to preserve data for litigation purposes and are not designed for general, long-term retention policies. While a legal hold could prevent deletion, it doesn't set a retention period of 7 years, and the policy would need to be manually removed after 7 years, adding complexity.

Therefore, Compliance mode with a default retention period ensures that objects cannot be deleted or modified for 7 years, even by the root user, fulfilling the defined requirements in the most effective and secure manner.

Authoritative Links:

AWS Documentation on S3 Object Lock: https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html
AWS Blog on S3 Object Lock: https://aws.amazon.com/blogs/aws/amazon-s3-object-lock/

Question: 155 CertyIQ

A data engineer needs to create a new empty table in Amazon Athena that has the same schema as an existing table named old_table.

Which SQL statement should the data engineer use to meet this requirement?

A.CREATE TABLE new_table AS SELECT * FROM old_tables;
B.INSERT INTO new_table SELECT * FROM old_table;
C.CREATE TABLE new_table (LIKE old_table);
D.CREATE TABLE new_table AS (SELECT * FROM old_table) WITH NO DATA;

Answer: C

Explanation:

The correct SQL statement to create a new empty table in Amazon Athena with the same schema as an existing table old_table is CREATE TABLE new_table (LIKE old_table). This statement leverages Athena's CREATE TABLE LIKE functionality, which is specifically designed to duplicate the schema (column names, data types, and potentially column properties depending on the engine) of an existing table without copying any data.

Option A, CREATE TABLE new_table AS SELECT * FROM old_tables;, would attempt to create a new table and populate it with data from old_table. This violates the requirement of creating an empty table.

Option B, INSERT INTO new_table SELECT * FROM old_table;, assumes that new_table already exists. Further, it tries to insert data from old_table into new_table, again not fulfilling the empty table requirement and risking failure if new_table does not exist.

Option D, CREATE TABLE new_table AS (SELECT * FROM old_table) WITH NO DATA;, although close, is not the most efficient or idiomatic way to achieve the desired outcome in Athena. While it would create an empty table with the same schema, the LIKE clause specifically caters to schema replication and is the intended solution. It might also introduce potential issues related to data type mappings and partitioning compared to the direct LIKE command, depending on the underlying data. The WITH NO DATA clause is often associated with CTAS (Create Table As Select) statements where data transformation or filtering is involved.

The CREATE TABLE LIKE approach is a direct and clear method for schema duplication, making it the preferred solution for this specific requirement. It leverages Athena's built-in capabilities and is a best practice for creating empty tables with an existing table's structure. The LIKE clause precisely targets schema duplication without involving data manipulation. The CREATE TABLE LIKE syntax offers a concise and efficient method, aligning with the objective of mirroring the table's schema while remaining empty.

Refer to the official AWS Athena documentation for more details on the CREATE TABLE syntax and the LIKE clause:

AWS Athena CREATE TABLE documentation
AWS Athena CREATE TABLE LIKE documentation (See section on "Creating an Empty Table Using CREATE TABLE AS SELECT") - while this documentation focuses on CTAS, it showcases the different capabilities.

Question: 156 CertyIQ

A data engineer needs to create an Amazon Athena table based on a subset of data from an existing Athena table

named cities_world. The cities_world table contains cities that are located around the world. The data engineer must create a new table named cities_us to contain only the cities from cities_world that are located in the US.

Which SQL statement should the data engineer use to meet this requirement?

A.INSERT INTO cities_usa (city,state) SELECT city, state FROM cities_world WHERE country='usa';
B.MOVE city, state FROM cities_world TO cities_usa WHERE country='usa';
C.INSERT INTO cities_usa SELECT city, state FROM cities_world WHERE country='usa';
D.UPDATE cities_usa SET (city, state) = (SELECT city, state FROM cities_world WHERE country='usa');

Answer: C

Explanation:

The correct SQL statement for creating a new Athena table cities_us from a subset of data in cities_world where the country is the US is INSERT INTO cities_usa SELECT city, state FROM cities_world WHERE country='usa';.

Here's why:

INSERT INTO: This SQL command is used to insert data into an existing table. In this case, we're populating cities_usa with data derived from another table.
SELECT city, state FROM cities_world: This part of the statement selects specific columns (city and state) from the source table cities_world. These are the columns whose values we want to transfer into the new table.
WHERE country='usa': This is the crucial filtering condition. It ensures that only rows from cities_world where the country column's value is 'usa' are selected and inserted into cities_usa. This satisfies the requirement of creating a table containing only US cities.

Why other options are incorrect:

Option A INSERT INTO cities_usa (city,state) SELECT city, state FROM cities_world WHERE country='usa';: While this might seem correct at first glance, it is incorrect syntax within Athena. Athena requires the columns be selected in the same order and number as the underlying table definitions. If cities_usa had any other columns aside from city and state, this statement will fail. Option C is more suitable for this reason.
Option B MOVE city, state FROM cities_world TO cities_usa WHERE country='usa';: There's no MOVE command in standard SQL, especially not for transferring data between tables. MOVE typically implies deleting the data from the source, which is definitely not what the requirement asks for. This option is syntactically incorrect and functionally inappropriate.
Option D UPDATE cities_usa SET (city, state) = (SELECT city, state FROM cities_world WHERE country='usa');: The UPDATE command is for modifying existing rows in a table, not for creating new rows. Also, this statement is structured to attempt setting city and state of all rows in cities_usa to only one city and state from cities_world (the first one it finds where the country is 'usa'). This contradicts the goal of adding all US cities from cities_world to cities_usa.

In summary, option C is the only SQL command that correctly selects a subset of rows based on a condition and inserts them into a new table, meeting the given requirements. INSERT INTO...SELECT...WHERE is a standard pattern for creating new tables populated with data from existing ones.

Athena INSERT INTO documentation

Question: 157 CertyIQ

A company implements a data mesh that has a central governance account. The company needs to catalog all data

in the governance account. The governance account uses AWS Lake Formation to centrally share data and grant access permissions.

The company has created a new data product that includes a group of Amazon Redshift Serverless tables. A data engineer needs to share the data product with a marketing team. The marketing team must have access to only a subset of columns. The data engineer needs to share the same data product with a compliance team. The compliance team must have access to a different subset of columns than the marketing team needs access to.

Which combination of steps should the data engineer take to meet these requirements? (Choose two.)

A.Create views of the tables that need to be shared. Include only the required columns.

B.Create an Amazon Redshift data share that includes the tables that need to be shared.

C.Create an Amazon Redshift managed VPC endpoint in the marketing team’s account. Grant the marketing team access to the views.

D.Share the Amazon Redshift data share to the Lake Formation catalog in the governance account.

E.Share the Amazon Redshift data share to the Amazon Redshift Serverless workgroup in the marketing team's account.

Answer: BD

Explanation:

The correct answer is B and D. Here's a detailed justification:

B. Create an Amazon Redshift data share that includes the tables that need to be shared.

Redshift data sharing is designed to enable secure sharing of data across Redshift clusters or Serverless workgroups without the need to copy or move data. In a data mesh architecture with centralized governance, data sharing facilitates providing data products to different teams. By creating a data share, the data engineer can package the relevant tables from the Redshift Serverless instance into a shareable unit. This aligns with the requirement of sharing the data product.

D. Share the Amazon Redshift data share to the Lake Formation catalog in the governance account.

Integrating Redshift data shares with the Lake Formation catalog is critical for centralized data governance. Lake Formation provides a central repository for metadata, enabling discovery, access control, and auditing. Sharing the Redshift data share to Lake Formation makes the data product discoverable and manageable within the governed environment. The governance account can then manage permissions and access to the data share. This integration enables granular control over data access, a key principle of data mesh and centralized governance. Lake Formation can then grant permissions based on the user or role accessing the data in the share.

Why other options are incorrect:

A. Create views of the tables that need to be shared. Include only the required columns. While views are useful for restricting column access, creating views in the data producer's Redshift cluster/Serverless and then finding a mechanism to grant access to those views from the consumer accounts doesn't align with the data mesh principles. You'd have to either give direct access to Redshift, which is not ideal, or create a data lake process to copy the data to another location. The intent is to make the producer responsible for the data, which data sharing achieves.

C. Create an Amazon Redshift managed VPC endpoint in the marketing team’s account. Grant the marketing team access to the views. VPC endpoints provide private connectivity, but are not part of the core mechanism for sharing and managing access in a data mesh, especially considering Lake Formation's involvement for governance. Also, this depends on option A which makes this incorrect by default.

E. Share the Amazon Redshift data share to the Amazon Redshift Serverless workgroup in the marketing team's account. While you can share Redshift data shares between Redshift clusters or Serverless instances,

doing it directly to the marketing team's account bypasses the central governance provided by Lake Formation. The governance account must be the one managing data sharing and permissions for all data products. Also, column level access cannot be granted through the redshift console..

Supporting Links:

Amazon Redshift data sharing
AWS Lake Formation documentation

Question: 158 CertyIQ

A company has a data lake in Amazon S3. The company uses AWS Glue to catalog data and AWS Glue Studio to implement data extract, transform, and load (ETL) pipelines.

The company needs to ensure that data quality issues are checked every time the pipelines run. A data engineer must enhance the existing pipelines to evaluate data quality rules based on predefined thresholds.

Which solution will meet these requirements with the LEAST implementation effort?

A.Add a new transform that is defined by a SQL query to each Glue ETL job. Use the SQL query to implement a ruleset that includes the data quality rules that need to be evaluated.

B.Add a new Evaluate Data Quality transform to each Glue ETL job. Use Data Quality Definition Language (DQDL) to implement a ruleset that includes the data quality rules that need to be evaluated.

C.Add a new custom transform to each Glue ETL job. Use the PyDeequ library to implement a ruleset that includes the data quality rules that need to be evaluated.

D.Add a new custom transform to each Glue ETL job. Use the Great Expectations library to implement a ruleset that includes the data quality rules that need to be evaluated.

Answer: B

Explanation:

The correct answer is **B. Add a new Evaluate Data Quality transform to each Glue ETL job. Use Data Quality Definition Language (DQDL) to implement a ruleset that includes the data quality rules that need to be evaluated.**

Here's a detailed justification:

AWS Glue's "Evaluate Data Quality" transform is specifically designed for implementing data quality checks within Glue ETL pipelines. This approach offers the least implementation effort because it leverages a built-in Glue functionality tailored for the task. Data Quality Definition Language (DQDL) is the language provided by AWS Glue for defining data quality rules.

Option A suggests using SQL queries. While SQL can perform some data quality checks, it would require significantly more complex SQL and custom logic to implement a comprehensive ruleset, increasing development time and complexity. This introduces extra overhead and may not be as efficient as a dedicated data quality transform.

Options C and D propose using custom transforms with PyDeequ and Great Expectations, respectively. While both PyDeequ and Great Expectations are powerful data quality libraries, integrating them as custom transforms in Glue ETL jobs demands considerable coding and dependency management. This involves packaging the library, configuring the environment, and writing custom Python code to interact with the library, resulting in a significantly more complex setup and maintenance burden compared to the native Glue functionality. The integration complexity is high compared to the direct integration using the "Evaluate Data Quality" transform.

The "Evaluate Data Quality" transform with DQDL offers a declarative way to define data quality rules,

simplifying the process. It's the most direct, integrated, and therefore the solution requiring the least implementation effort to meet the data quality monitoring requirements. Using Glue's built-in functionality optimizes for maintainability and reduces custom code dependencies, making it the preferred solution for the described scenario.

For further research, refer to the following AWS documentation:

AWS Glue Data Quality: https://docs.aws.amazon.com/glue/latest/dg/glue_data_quality.html
Data Quality Definition Language (DQDL): https://docs.aws.amazon.com/glue/latest/dg/data_quality_rules-dqdl.html

Question: 159 CertyIQ

A company has an application that uses a microservice architecture. The company hosts the application on an Amazon Elastic Kubernetes Services (Amazon EKS) cluster.

The company wants to set up a robust monitoring system for the application. The company needs to analyze the logs from the EKS cluster and the application. The company needs to correlate the cluster's logs with the application's traces to identify points of failure in the whole application request flow.

Which combination of steps will meet these requirements with the LEAST development effort? (Choose two.)

A.Use FluentBit to collect logs. Use OpenTelemetry to collect traces.

B.Use Amazon CloudWatch to collect logs. Use Amazon Kinesis to collect traces.

C.Use Amazon CloudWatch to collect logs. Use Amazon Managed Streaming for Apache Kafka (Amazon MSK) to collect traces.

D.Use Amazon OpenSearch to correlate the logs and traces.

E.Use AWS Glue to correlate the logs and traces.

Answer: AD

Explanation:

Here's a detailed justification for why options A and D are the best choices for correlating EKS cluster logs and application traces with minimal development effort:

Option A: Use FluentBit to collect logs. Use OpenTelemetry to collect traces.

FluentBit: FluentBit is a lightweight, open-source log processor and forwarder specifically designed for containerized environments like EKS. It efficiently collects logs from various sources within the cluster (pods, containers, system logs) with minimal resource consumption. Its Kubernetes integration makes log collection straightforward. https://www.fluentbit.io/
OpenTelemetry: OpenTelemetry is an open-source observability framework that provides standardized APIs, SDKs, and tools for collecting and exporting traces, metrics, and logs. Using OpenTelemetry simplifies instrumentation across different microservices and programming languages. It avoids vendor lock-in, promotes interoperability, and provides the necessary context for tracing requests across the entire application. https://opentelemetry.io/
By using these tools, the company will minimize the amount of development to implement this solution. They are designed to be containerized, which makes them compatible with the current architecture.

Option D: Use Amazon OpenSearch to correlate the logs and traces.

Amazon OpenSearch: Amazon OpenSearch Service (successor to Elasticsearch Service) is a fully managed search and analytics service that makes it easy to ingest, secure, search, analyze, and visualize data in real-time. OpenSearch is a good choice for log aggregation and analysis because of its ability to index and search

large volumes of data quickly. OpenSearch supports ingest pipeline processing, which allows for the enrichment of logs with trace IDs captured by OpenTelemetry. It also enables powerful querying and visualization of both logs and traces, which is important for identifying points of failure in the request flow.
https://aws.amazon.com/opensearch-service/

Why other options are less suitable:

B & C: While CloudWatch can collect logs, it is not as efficient or well-suited for the dynamic and high-volume nature of Kubernetes logs as FluentBit. Kinesis and MSK are better suited for real-time data streaming use cases, and are not the most cost effective choices for collecting traces.
E: AWS Glue is primarily for ETL (extract, transform, load) operations and data cataloging, not for real-time correlation of logs and traces.
Additionally, options B, C, and E require more development effort.

Therefore, the combination of FluentBit for log collection, OpenTelemetry for trace collection, and Amazon OpenSearch for correlation provides a robust and scalable solution for monitoring the EKS cluster and correlating logs with application traces with the least development effort.

Question: 160 CertyIQ

A company has a gaming application that stores data in Amazon DynamoDB tables. A data engineer needs to ingest the game data into an Amazon OpenSearch Service cluster. Data updates must occur in near real time.

Which solution will meet these requirements?

A.Use AWS Step Functions to periodically export data from the Amazon DynamoDB tables to an Amazon S3 bucket. Use an AWS Lambda function to load the data into Amazon OpenSearch Service.

B.Configure an AWS Glue job to have a source of Amazon DynamoDB and a destination of Amazon OpenSearch Service to transfer data in near real time.

C.Use Amazon DynamoDB Streams to capture table changes. Use an AWS Lambda function to process and update the data in Amazon OpenSearch Service.

D.Use a custom OpenSearch plugin to sync data from the Amazon DynamoDB tables.

Answer: C

Explanation:

Here's a detailed justification for why option C is the best solution:

The primary requirement is near real-time data updates from DynamoDB to OpenSearch Service. Option C directly addresses this by leveraging DynamoDB Streams. DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table, akin to a change data capture (CDC) mechanism. This is crucial for real-time data synchronization.

An AWS Lambda function is then used to subscribe to this DynamoDB stream. Each time a record is modified in DynamoDB (insert, update, or delete), the Lambda function is triggered. The Lambda function processes this change record and updates the corresponding document in the OpenSearch Service cluster. This creates an automated pipeline ensuring that the OpenSearch Service cluster reflects the latest data in DynamoDB with minimal latency.

Options A and B are not suitable for near real-time updates. Option A involves periodic exports to S3 and loading via Lambda, which introduces significant latency. Option B utilizes AWS Glue, which is more appropriate for batch processing and ETL tasks, not continuous near real-time data ingestion. Option D, using a custom OpenSearch plugin, introduces unnecessary complexity and potential maintenance overhead compared to the managed services offered by AWS. Using a custom plugin also requires deep expertise in

OpenSearch internals.

Option C utilizes native AWS services that are optimized for this type of integration. It's serverless, scalable, and cost-effective because you only pay for the Lambda function execution time when changes occur. The integration between DynamoDB Streams and Lambda functions provides a robust and reliable near real-time data pipeline. The "push" model enabled by DynamoDB streams is far more appropriate for real-time updates than the periodic "pull" model of option A and the batch processing approach of option B.

For further research, refer to the following documentation:

Amazon DynamoDB Streams:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html
AWS Lambda: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
Amazon OpenSearch Service: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/what-is.html

Question: 161 CertyIQ

A company uses Amazon Redshift as its data warehouse service. A data engineer needs to design a physical data model.

The data engineer encounters a de-normalized table that is growing in size. The table does not have a suitable column to use as the distribution key.

Which distribution style should the data engineer use to meet these requirements with the LEAST maintenance overhead?

A.ALL distribution
B.EVEN distribution
C.AUTO distribution
D.KEY distribution

Answer: C

Explanation:

The correct answer is **C. AUTO distribution**. Let's break down why:

Problem: The scenario presents a large, denormalized table in Amazon Redshift lacking an ideal distribution key. The goal is to choose a distribution style that minimizes maintenance overhead.

Understanding Distribution Styles:

KEY distribution: This requires selecting a column as the distribution key. Since the table lacks a suitable column, using KEY distribution would force a suboptimal choice, potentially leading to skewed data distribution and performance issues. It also requires manual intervention if data patterns change.
EVEN distribution: This distributes data evenly across all nodes in the cluster in a round-robin fashion. While simple, it often leads to data needing to be moved across nodes during queries (data redistribution), impacting performance, especially with large tables. It doesn't require initial key selection but doesn't adapt to data patterns.
ALL distribution: This replicates the entire table to every node in the cluster. It's suitable for small, frequently joined tables but highly inefficient for large, growing tables due to storage overhead and data consistency requirements. It's not appropriate here.
AUTO distribution: This allows Amazon Redshift to automatically choose and manage the distribution style. Initially, it will use EVEN distribution. It then monitors query patterns and data characteristics and can

automatically switch to KEY distribution if a suitable key is identified or maintain EVEN distribution if that's optimal. It handles data skew more gracefully than manual distribution.

Why AUTO is the best choice: AUTO distribution minimizes maintenance overhead because it dynamically adapts to changes in data volume, data skew, and query patterns. It starts with EVEN and evolves based on the workload, potentially switching to KEY or staying with EVEN as needed. This avoids the need for a data engineer to manually select, test, and optimize the distribution key and re-distribute the data as the table grows and usage changes. Since the prompt focuses on minimizing maintenance overhead, AUTO is the clear winner. Using EVEN distribution (without AUTO) will result in having to manually monitor and manage the data skew and manually change it to KEY when/if a suitable distribution key is found later.

Links for Further Research:

Amazon Redshift Distribution Styles
Amazon Redshift Table Design Best Practices
Amazon Redshift AUTO distribution

Question: 162 CertyIQ

A retail company is expanding its operations globally. The company needs to use Amazon QuickSight to accurately calculate currency exchange rates for financial reports. The company has an existing dashboard that includes a visual that is based on an analysis of a dataset that contains global currency values and exchange rates.

A data engineer needs to ensure that exchange rates are calculated with a precision of four decimal places. The calculations must be precomputed. The data engineer must materialize results in QuickSight super-fast, parallel, in-memory calculation engine (SPICE).

Which solution will meet these requirements?

A.Define and create the calculated field in the dataset.
B.Define and create the calculated field in the analysis.
C.Define and create the calculated field in the visual.
D.Define and create the calculated field in the dashboard.

Answer: A

Explanation:

The correct answer is A: Define and create the calculated field in the dataset. Here's why:

The requirement to precompute exchange rates and materialize them in SPICE dictates that the calculation should be done at the dataset level. SPICE, being QuickSight's in-memory engine, excels at fast data retrieval and analysis when data is pre-processed and stored efficiently. Defining the calculated field within the dataset ensures that the exchange rate calculation is performed during data ingestion and is stored as part of the dataset itself. This precomputed value is then readily available to the visuals without requiring runtime calculation, optimizing performance.

Calculating the exchange rates directly within the dataset allows for the desired precision (four decimal places) to be enforced during the calculation process. This ensures consistency and accuracy across all visuals using this dataset. Furthermore, defining it at the dataset level makes the calculation reusable across multiple analyses and dashboards, promoting maintainability and reducing redundancy. Options B, C, and D are incorrect because they perform the calculation at the analysis, visual, or dashboard level, respectively. These approaches would compute the exchange rates at runtime, preventing materialization in SPICE and potentially impacting dashboard performance. They also make it harder to ensure consistency across the board. Only defining the calculation at the data ingestion level guarantees it will happen during ingestion and will be accessible in spice and ensure a higher quality output with better performance.

https://docs.aws.amazon.com/quicksight/latest/user/working-with-calculated-fields.htmlhttps://docs.aws.amazon.com/quicksight/latest/user/spice.html

Question: 163 CertyIQ

A company has three subsidiaries. Each subsidiary uses a different data warehousing solution. The first subsidiary hosts its data warehouse in Amazon Redshift. The second subsidiary uses Teradata Vantage on AWS. The third subsidiary uses Google BigQuery.

The company wants to aggregate all the data into a central Amazon S3 data lake. The company wants to use Apache Iceberg as the table format.

A data engineer needs to build a new pipeline to connect to all the data sources, run transformations by using each source engine, join the data, and write the data to Iceberg.

Which solution will meet these requirements with the LEAST operational effort?

A.Use native Amazon Redshift, Teradata, and BigQuery connectors to build the pipeline in AWS Glue. Use native AWS Glue transforms to join the data. Run a Merge operation on the data lake Iceberg table.

B.Use the Amazon Athena federated query connectors for Amazon Redshift, Teradata, and BigQuery to build the pipeline in Athena. Write a SQL query to read from all the data sources, join the data, and run a Merge operation on the data lake Iceberg table.

C.Use the native Amazon Redshift connector, the Java Database Connectivity (JDBC) connector for Teradata, and the open source Apache Spark BigQuery connector to build the pipeline in Amazon EMR. Write code in PySpark to join the data. Run a Merge operation on the data lake Iceberg table.

D.Use the native Amazon Redshift, Teradata, and BigQuery connectors in Amazon Appflow to write data to Amazon S3 and AWS Glue Data Catalog. Use Amazon Athena to join the data. Run a Merge operation on the data lake Iceberg table.

Answer: A

Explanation:

The best solution for aggregating data from Redshift, Teradata, and BigQuery into an S3 data lake using Iceberg with the least operational effort is option A. AWS Glue provides native connectors for these data warehouses, simplifying the connection process. Glue's built-in transforms are efficient for data joining operations. The Merge operation within Glue allows for streamlined updates to the Iceberg table, ensuring data consistency and minimal code development. This approach minimizes the need for custom coding and infrastructure management, which is a key factor in reducing operational overhead.

Option B requires using Athena federated queries, but Athena is more suited for ad-hoc querying and might not be the most efficient for complex ETL pipelines. Also, relying solely on SQL for joining and merging data can become complex and less maintainable than using Glue's visual interface and transforms.

Option C involves using Amazon EMR with Spark. While EMR is powerful, it requires more operational overhead for cluster management, configuration, and scaling compared to Glue. Writing PySpark code adds complexity and maintenance efforts.

Option D uses AppFlow to initially write data to S3 and then Athena for joining. This introduces an unnecessary step of data staging and increases the complexity of the pipeline. AppFlow is designed for SaaS data transfer, not complex data integration and transformation, making Glue a better fit.

Therefore, AWS Glue provides the best balance of ease of use, native connector support, and efficient data transformation capabilities for this scenario, resulting in the least operational effort.

Relevant Links:

AWS Glue: https://aws.amazon.com/glue/
Apache Iceberg: https://iceberg.apache.org/
Amazon Athena: https://aws.amazon.com/athena/
Amazon EMR: https://aws.amazon.com/emr/
Amazon AppFlow: https://aws.amazon.com/appflow/

Question: 164 CertyIQ

A company is building a data stream processing application. The application runs in an Amazon Elastic Kubernetes Service (Amazon EKS) cluster. The application stores processed data in an Amazon DynamoDB table.

The company needs the application containers in the EKS cluster to have secure access to the DynamoDB table. The company does not want to embed AWS credentials in the containers.

Which solution will meet these requirements?

A.Store the AWS credentials in an Amazon S3 bucket. Grant the EKS containers access to the S3 bucket to retrieve the credentials.

B.Attach an IAM role to the EKS worker nodes, Grant the IAM role access to DynamoDUse the IAM role to set up IAM roles service accounts (IRSA) functionality.

C.Create an IAM user that has an access key to access the DynamoDB table. Use environment variables in the EKS containers to store the IAM user access key data.

D.Create an IAM user that has an access key to access the DynamoDB table. Use Kubernetes secrets that are mounted in a volume of the EKS duster nodes to store the user access key data.

Answer: B

Explanation:

The correct answer is B because it leverages IAM Roles for Service Accounts (IRSA) to grant secure access to DynamoDB without embedding credentials directly within the containers.

Here's why:

IAM Roles for Service Accounts (IRSA): This approach associates an IAM role with a Kubernetes service account. When a pod uses that service account, AWS credentials are automatically fetched and provided to the application inside the pod. This eliminates the need to manage and rotate credentials within the container itself, significantly improving security.
IAM Role Attachment to EKS Worker Nodes: EKS worker nodes need an IAM role to function, and attaching this role provides base permissions. However, it is best practice to grant the least privilege necessary to applications running within the cluster and not solely rely on the worker node's IAM role. The combination of this worker node role and IRSA facilitates the desired granular access control.
Security Best Practices: Embedding or storing credentials within containers (options A, C, and D) is a security risk. If a container is compromised, the credentials are exposed. IRSA eliminates this risk by dynamically providing temporary credentials managed by AWS IAM.

Why other options are incorrect:

Option A (S3 Bucket for Credentials): While storing credentials in S3 is better than embedding them in the code, it still requires managing access to the S3 bucket itself and poses a security risk if the bucket is compromised.
Option C and D (IAM User Access Key in Environment Variables/Secrets): Storing IAM user access keys in environment variables or Kubernetes secrets is a bad practice. Environment variables can be inadvertently logged or exposed, and Kubernetes secrets, while encrypted at rest, require careful management and can still be compromised if the cluster is. IAM users should generally be reserved for human access and not used

within applications.

In summary, IRSA is the preferred method for securely granting AWS permissions to applications running in EKS because it avoids storing long-term credentials within the cluster and leverages IAM roles for temporary credential vending.

Supporting Links:

IAM Roles for Service Accounts (IRSA): https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html
Security Best Practices in EKS: https://aws.github.io/aws-eks-best-practices/security/docs/iam/

Question: 165 CertyIQ

A data engineer needs to onboard a new data producer into AWS. The data producer needs to migrate data products to AWS.

The data producer maintains many data pipelines that support a business application. Each pipeline must have service accounts and their corresponding credentials. The data engineer must establish a secure connection from the data producer's on-premises data center to AWS. The data engineer must not use the public internet to transfer data from an on-premises data center to AWS.

Which solution will meet these requirements?

A.Instruct the new data producer to create Amazon Machine Images (AMIs) on Amazon Elastic Container Service (Amazon ECS) to store the code base of the application. Create security groups in a public subnet that allow connections only to the on-premises data center.

B.Create an AWS Direct Connect connection to the on-premises data center. Store the service account credentials in AWS Secrets manager.

C.Create a security group in a public subnet. Configure the security group to allow only connections from the CIDR blocks that correspond to the data producer. Create Amazon S3 buckets than contain presigned URLS that have one-day expiration dates.

D.Create an AWS Direct Connect connection to the on-premises data center. Store the application keys in AWS Secrets Manager. Create Amazon S3 buckets that contain presigned URLS that have one-day expiration dates.

Answer: B

Explanation:

The correct answer is **B**. Here's why:

Secure Connection (AWS Direct Connect): The requirement to avoid using the public internet for data transfer is directly addressed by AWS Direct Connect. Direct Connect establishes a dedicated network connection from the on-premises data center to AWS, providing a private, secure, and consistent connection. This bypasses the public internet, enhancing security and potentially improving performance. (See: https://aws.amazon.com/directconnect/)

Credential Management (AWS Secrets Manager): Storing service account credentials (application keys) in AWS Secrets Manager provides a secure and centralized location for managing sensitive information. Secrets Manager enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. This reduces the risk of hardcoding credentials in applications and improves overall security posture. (See: https://aws.amazon.com/secrets-manager/)

Why other options are not optimal:

Option A: Using Amazon ECS and AMIs to store application code is irrelevant to the requirements of secure connection and credential management. While ECS is useful for containerized application deployment, it

doesn't address secure data transfer from on-premises or secure credential storage. The public subnet also violates the security aspect.
Option C: Security groups and presigned URLs, while useful security measures, do not replace the necessity of a Direct Connect connection for avoiding the public internet. Presigned URLs provide temporary access to S3 buckets, but the data transfer itself could still occur over the internet if Direct Connect is not used.
Option D: This option is partially correct as it implements AWS Direct Connect and Secrets Manager. But it mixes credential storage for application keys with presigned urls that only have one day expirations. This does not fulfill the requirement for secure service account credentials. Option B is better and more directly addresses the complete security requirements for both data transfer and credential management.

Therefore, Option B is the most complete solution as it provides both a secure dedicated network connection and secure credential management, fulfilling all the requirements in the prompt.

Question: 166 CertyIQ

A data engineer configured an AWS Glue Data Catalog for data that is stored in Amazon S3 buckets. The data engineer needs to configure the Data Catalog to receive incremental updates.

The data engineer sets up event notifications for the S3 bucket and creates an Amazon Simple Queue Service (Amazon SQS) queue to receive the S3 events.

Which combination of steps should the data engineer take to meet these requirements with LEAST operational overhead? (Choose two.)

A.Create an S3 event-based AWS Glue crawler to consume events from the SQS queue.

B.Define a time-based schedule to run the AWS Glue crawler, and perform incremental updates to the Data Catalog.

C.Use an AWS Lambda function to directly update the Data Catalog based on S3 events that the SQS queue receives.

D.Manually initiate the AWS Glue crawler to perform updates to the Data Catalog when there is a change in the S3 bucket.

E.Use AWS Step Functions to orchestrate the process of updating the Data Catalog based on S3 events that the SQS queue receives.

Answer: BC

Explanation:

The correct answer is **B and C**. Here's a detailed justification:

Option B Justification:

Scheduling a Glue crawler for incremental updates is a standard approach for managing data catalogs. A time-based schedule, like running the crawler hourly or daily, allows the crawler to automatically discover new partitions and schema changes in the S3 data lake. This reduces operational overhead because the data engineer doesn't need to manually trigger the crawler every time data is updated. Incremental crawls are more efficient than full crawls, as they only process changes since the last crawl, minimizing resource usage.

Incremental Crawls: https://docs.aws.amazon.com/glue/latest/dg/incremental-crawls.html
Scheduling Crawlers: https://docs.aws.amazon.com/glue/latest/dg/console-crawlers.html

Option C Justification:

Using a Lambda function triggered by the SQS queue offers a real-time update mechanism to the Data Catalog. When an S3 event (like object creation or deletion) arrives in the SQS queue, the Lambda function is invoked. The Lambda function can then directly call the AWS Glue Data Catalog APIs to update the metadata,

adding a new table partition or updating existing schema information. This approach provides near real-time updates and minimal operational overhead as Lambda is serverless and scales automatically. This approach is especially useful if near real-time catalog updates are needed.

AWS Lambda with SQS: https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html
AWS Glue Data Catalog API: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api.html

Why other options are incorrect:

A: S3 event-based Glue Crawlers are a viable option, however, you can't directly consume SQS events with the event-based Glue crawler. Glue crawlers natively support event triggers, but not direct consumption from an SQS queue that already filters and processes those events. Implementing this would likely require a more complex setup.
D: Manual triggering defeats the purpose of automating incremental updates and increases operational overhead.
E: Using Step Functions for this scenario introduces unnecessary complexity. Step Functions are useful for orchestrating multiple steps of a complex workflow, which is not required for directly updating the Data Catalog based on S3 events. A Lambda function is a simpler and more appropriate solution.

In summary, a time-based scheduled Glue crawler (B) efficiently manages periodic updates, while a Lambda function triggered by SQS (C) handles near real-time updates based on specific S3 events. This combination addresses the requirements with the least operational overhead.

Question: 167 CertyIQ

A company uses AWS Glue Data Catalog to index data that is uploaded to an Amazon S3 bucket every day. The company uses a daily batch processes in an extract, transform, and load (ETL) pipeline to upload data from external sources into the S3 bucket.

The company runs a daily report on the S3 data. Some days, the company runs the report before all the daily data has been uploaded to the S3 bucket. A data engineer must be able to send a message that identifies any incomplete data to an existing Amazon Simple Notification Service (Amazon SNS) topic.

Which solution will meet this requirement with the LEAST operational overhead?

A.Create data quality checks for the source datasets that the daily reports use. Create a new AWS managed Apache Airflow cluster. Run the data quality checks by using Airflow tasks that run data quality queries on the columns data type and the presence of null values. Configure Airflow Directed Acyclic Graphs (DAGs) to send an email notification that informs the data engineer about the incomplete datasets to the SNS topic.

B.Create data quality checks on the source datasets that the daily reports use. Create a new Amazon EMR cluster. Use Apache Spark SQL to create Apache Spark jobs in the EMR cluster that run data quality queries on the columns data type and the presence of null values. Orchestrate the ETL pipeline by using an AWS Step Functions workflow. Configure the workflow to send an email notification that informs the data engineer about the incomplete datasets to the SNS topic.

C.Create data quality checks on the source datasets that the daily reports use. Create data quality actions by using AWS Glue workflows to confirm the completeness and consistency of the datasets. Configure the data quality actions to create an event in Amazon EventBridge if a dataset is incomplete. Configure EventBridge to send the event that informs the data engineer about the incomplete datasets to the Amazon SNS topic.

D.Create AWS Lambda functions that run data quality queries on the columns data type and the presence of null values. Orchestrate the ETL pipeline by using an AWS Step Functions workflow that runs the Lambda functions. Configure the Step Functions workflow to send an email notification that informs the data engineer about the incomplete datasets to the SNS topic.

Answer: C

Explanation:

The most efficient solution (least operational overhead) is option C, leveraging AWS Glue's data quality features and EventBridge for notification.

Here's why:

AWS Glue Data Quality Actions: Glue offers built-in data quality checks directly within its workflows. Using these actions avoids the need for custom scripting and separate infrastructure (like EMR or Airflow). It also tightly integrates with the Glue Data Catalog. https://docs.aws.amazon.com/glue/latest/dg/data-quality.html

EventBridge: EventBridge is a serverless event bus that allows you to react to changes in your AWS environment. By configuring Glue data quality actions to emit events upon detecting incomplete data, you can trigger a notification mechanism without writing complex orchestration logic. This minimizes operational overhead compared to Step Functions or Airflow workflows. https://aws.amazon.com/eventbridge/

SNS Integration: EventBridge can directly integrate with SNS to publish messages when specific events occur. This makes it a straightforward solution for alerting the data engineer about incomplete datasets.

Alternatives (Why they are less optimal):

A (Airflow): Requires setting up and managing an Airflow cluster, which adds significant operational overhead. Developing and maintaining Airflow DAGs specifically for data quality checks is also more complex than using Glue's built-in features.
B (EMR/Spark): Similarly, setting up and managing an EMR cluster for data quality checks is an over-engineered solution. Spark SQL is powerful, but unnecessary when Glue provides equivalent data quality functionality with less operational overhead.
D (Lambda/Step Functions): While serverless, this approach requires writing custom Lambda functions for data quality checks, which is more work than utilizing Glue's built-in features. The Step Functions workflow is also more complex than the simple EventBridge rule.

In summary, option C provides the simplest, most integrated, and most cost-effective way to monitor data quality within your existing AWS Glue-based ETL pipeline and notify stakeholders about incomplete datasets using SNS. Glue's data quality features are designed precisely for this type of use case and require minimal custom coding or infrastructure management.

Question: 168 CertyIQ

A company stores customer data that contains personally identifiable information (PII) in an Amazon Redshift cluster. The company's marketing, claims, and analytics teams need to be able to access the customer data.

The marketing team should have access to obfuscated claim information but should have full access to customer contact information. The claims team should have access to customer information for each claim that the team processes. The analytics team should have access only to obfuscated PII data.

Which solution will enforce these data access requirements with the LEAST administrative overhead?

A.Create a separate Redshift cluster for each team. Load only the required data for each team. Restrict access to clusters based on the teams.

B.Create views that include required fields for each of the data requirements. Grant the teams access only to the view that each team requires.

C.Create a separate Amazon Redshift database role for each team. Define masking policies that apply for each team separately. Attach appropriate masking policies to each team role.

D.Move the customer data to an Amazon S3 bucket. Use AWS Lake Formation to create a data lake. Use fine-grained security capabilities to grant each team appropriate permissions to access the data.

Answer: C

Explanation:

The best solution is **C: Create a separate Amazon Redshift database role for each team. Define masking policies that apply for each team separately. Attach appropriate masking policies to each team role.**

Here's why:

Redshift Roles & Masking Policies: Redshift offers database roles and masking policies designed precisely for this type of data access control. Roles allow you to group users and assign permissions collectively. Masking policies dynamically alter the data displayed to users based on their role, providing the necessary obfuscation (e.g., masking PII for the marketing and analytics teams).

Least Administrative Overhead: This approach centralizes data governance within the Redshift cluster. Creating separate clusters (option A) involves significantly more infrastructure management, data duplication, and cost. Moving to S3 and Lake Formation (option D) is an overkill involving unnecessary data migration and increased complexity if the data already resides in Redshift. Creating views (option B) can achieve granular access but managing multiple views and potential performance impacts is more overhead compared to the dynamic masking capabilities.

Flexibility & Scalability: Redshift roles and masking policies can be easily modified as team needs evolve or new data requirements emerge. It allows you to modify security configurations without requiring changes to the underlying data or cluster architecture.

Authorization and Auditing: Redshift's role-based access control facilitates detailed auditing of data access, enhancing security and compliance efforts. You can track which users or roles accessed what data and how masking policies were applied.

The other options are less efficient:

Option A: Isolating data into separate clusters is highly redundant, resource-intensive, and difficult to manage.
Option B: Managing numerous views can become complex and introduce maintenance challenges.
Option D: Moving data to S3 and using Lake Formation adds unnecessary complexity and cost, especially since the data is already in Redshift. Lake Formation is more suitable for heterogeneous data sources and data lakes spanning multiple services.

References:

Amazon Redshift Database Roles: https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_ROLE.html
Amazon Redshift Data Masking Policies: https://docs.aws.amazon.com/redshift/latest/dg/data-masking-policies.html

Question: 169 CertyIQ

A financial company recently added more features to its mobile app. The new features required the company to create a new topic in an existing Amazon Managed Streaming for Apache Kafka (Amazon MSK) cluster.

A few days after the company added the new topic, Amazon CloudWatch raised an alarm on the RootDiskUsed metric for the MSK cluster.

How should the company address the CloudWatch alarm?

A. Expand the storage of the MSK broker. Configure the MSK cluster storage to expand automatically.
B. Expand the storage of the Apache ZooKeeper nodes.
C. Update the MSK broker instance to a larger instance type. Restart the MSK cluster.
D. Specify the Target Volume-in-GiB parameter for the existing topic.

Answer: A

Explanation:

The CloudWatch alarm on RootDiskUsed indicates that the storage space allocated to the MSK broker instances is nearing its capacity. Since new features and a new topic were added to the Kafka cluster, it is highly probable that the data volume being stored on the brokers has increased significantly, leading to the storage capacity issue.

Option A, expanding the storage of the MSK broker, directly addresses the problem of insufficient disk space. Expanding the storage ensures that the Kafka brokers have enough space to store the incoming data, thereby resolving the CloudWatch alarm. Furthermore, configuring the MSK cluster storage to expand automatically avoids similar issues in the future as data volume grows. MSK supports storage auto-scaling feature.

Option B, expanding the storage of Apache ZooKeeper nodes, is incorrect. ZooKeeper is primarily responsible for metadata management and cluster coordination, not data storage. The data resides on the Kafka brokers. Increasing the storage for ZooKeeper will not alleviate the root disk utilization issue of the brokers.

Option C, updating the MSK broker instance to a larger instance type, might provide more storage depending on the instance type, but it's an indirect and potentially more expensive solution. While a larger instance type could increase compute and network resources as well, the primary problem identified by the CloudWatch alarm is storage capacity. It's best to directly address the specific issue rather than resorting to a more comprehensive (and likely pricier) upgrade of the instance type, unless CPU or Network are bottlenecking. Also, restarting the MSK cluster introduces unnecessary downtime.

Option D, specifying the Target Volume-in-GiB parameter for the existing topic, is also incorrect. The Target Volume-in-GiB parameter is used when creating or updating a tiered storage topic, indicating how much data to keep on the primary tier (broker storage) before tiering it to the secondary tier (S3). This parameter does not directly affect the total storage available to the brokers. This is the tiered storage feature, not an expansion of the overall broker storage. It helps in managing storage costs when dealing with data archiving to reduce the storage costs but is not applicable to the described scenario.

Therefore, expanding the MSK broker storage and configuring it to expand automatically is the most direct and efficient solution to address the CloudWatch alarm on RootDiskUsed.

Supporting Documentation:

Amazon MSK Storage Expansion:
Amazon MSK Monitoring with CloudWatch:
Amazon MSK Tiered Storage:

Question: 170 CertyIQ

A data engineer needs to build an enterprise data catalog based on the company's Amazon S3 buckets and Amazon RDS databases. The data catalog must include storage format metadata for the data in the catalog.

Which solution will meet these requirements with the LEAST effort?

A.Use an AWS Glue crawler to scan the S3 buckets and RDS databases and build a data catalog. Use data stewards to inspect the data and update the data catalog with the data format.

B.Use an AWS Glue crawler to build a data catalog. Use AWS Glue crawler classifiers to recognize the format of data and store the format in the catalog.

C.Use Amazon Macie to build a data catalog and to identify sensitive data elements. Collect the data format information from Macie.

D.Use scripts to scan data elements and to assign data classifications based on the format of the data.

Answer: B

Explanation:

Here's a detailed justification for why option B is the best solution, along with supporting concepts and links:

The requirement is to create an enterprise data catalog with storage format metadata for data residing in S3 buckets and RDS databases, minimizing manual effort.

Option B, using AWS Glue crawlers and classifiers, directly addresses these needs efficiently. AWS Glue crawlers can automatically scan data sources like S3 and RDS.
[https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html]

Crucially, Glue crawlers utilize classifiers to infer the schema and data format (e.g., CSV, JSON, Parquet) of the data. These classifiers are pre-built to recognize common formats, and custom classifiers can be added for less standard formats. [https://docs.aws.amazon.com/glue/latest/dg/add-classifier.html] This automated format recognition significantly reduces manual effort. The metadata, including the data format, is then stored within the AWS Glue Data Catalog.

Option A suggests manual inspection and updates by data stewards. This approach is time-consuming and prone to errors, directly contradicting the "least effort" requirement.

Option C proposes using Amazon Macie. While Macie is valuable for identifying sensitive data, it is not designed for building comprehensive data catalogs or focusing on storage format metadata. Macie's primary concern is data security and compliance, not general metadata management.
[https://aws.amazon.com/macie/]

Option D suggests using custom scripts to scan data elements and assign classifications. This involves significant development and maintenance overhead, violating the "least effort" requirement. It also duplicates functionality already provided by AWS Glue crawlers and classifiers.

In summary, AWS Glue crawlers with classifiers provide an automated and efficient way to discover, classify, and catalog data, including its storage format. This eliminates the need for manual inspection or custom scripting, making it the optimal solution for meeting the requirements with the least amount of effort.

Question: 171 CertyIQ

A company analyzes data in a data lake every quarter to perform inventory assessments. A data engineer uses AWS Glue DataBrew to detect any personally identifiable formation (PII) about customers within the data. The company's privacy policy considers some custom categories of information to be PII. However, the categories are not included in standard DataBrew data quality rules.

The data engineer needs to modify the current process to scan for the custom PII categories across multiple datasets within the data lake.

Which solution will meet these requirements with the LEAST operational overhead?

A.Manually review the data for custom PII categories.

B.Implement custom data quality rules in DataBrew. Apply the custom rules across datasets.

C.Develop custom Python scripts to detect the custom PII categories. Call the scripts from DataBrew.

D.Implement regex patterns to extract PII information from fields during extract transform, and load (ETL) operations into the data lake.

Answer: B

Explanation:

The best solution is **B. Implement custom data quality rules in DataBrew. Apply the custom rules across datasets.**

Here's why:

DataBrew's Custom Rules: AWS Glue DataBrew allows users to create and apply custom data quality rules. This means the data engineer can define rules specifically tailored to identify the custom PII categories defined in the company's privacy policy. This aligns directly with the requirement to scan for these specific categories.

Scalability and Automation: Once the custom rules are defined in DataBrew, they can be easily applied across multiple datasets within the data lake. This ensures a consistent and automated approach to PII detection, reducing manual effort and improving efficiency for the quarterly inventory assessments.

Least Operational Overhead: Creating and managing custom rules within DataBrew is a low-overhead approach. DataBrew provides a user-friendly interface and pre-built functionalities that minimize the need for extensive coding or complex infrastructure management.

Why other options are less ideal:

A. Manually review the data: Manual review is not scalable or efficient, especially when dealing with large datasets in a data lake. It's also prone to human error.
C. Develop custom Python scripts: While this approach is feasible, it introduces more operational overhead. It requires developing, testing, and maintaining custom scripts, as well as integrating them with DataBrew. Furthermore, DataBrew already provides the mechanism to define custom rules, making the scripts redundant.
D. Implement regex patterns during ETL: While this can be part of an overall data governance strategy, it's less flexible for ongoing monitoring of existing data in the lake. It addresses PII during ingestion but does not readily apply to datasets already present. Additionally, DataBrew's UI and rule-based approach offer easier management compared to embedding complex regex patterns in ETL pipelines.

Authoritative Links:

AWS Glue DataBrew Features: This provides an overview of DataBrew's capabilities, including data quality rules and PII detection.
AWS Glue DataBrew Documentation: Detailed documentation on using DataBrew, including creating and applying custom rules.

Question: 172 CertyIQ

A company receives a data file from a partner each day in an Amazon S3 bucket. The company uses a daily AWS Glue extract, transform, and load (ETL) pipeline to clean and transform each data file. The output of the ETL pipeline is written to a CSV file named Daily.csv in a second S3 bucket.

Occasionally, the daily data file is empty or is missing values for required fields. When the file is missing data, the company can use the previous day's CSV file.

A data engineer needs to ensure that the previous day's data file is overwritten only if the new daily file is complete and valid.

Which solution will meet these requirements with the LEAST effort?

A.Invoke an AWS Lambda function to check the file for missing data and to fill in missing values in required fields.

B.Configure the AWS Glue ETL pipeline to use AWS Glue Data Quality rules. Develop rules in Data Quality Definition Language (DQDL) to check for missing values in required fields and empty files.

C.Use AWS Glue Studio to change the code in the ETL pipeline to fill in any missing values in the required fields

with the most common values for each field.

D.Run a SQL query in Amazon Athena to read the CSV file and drop missing rows. Copy the corrected CSV file to the second S3 bucket.

Answer: B

Explanation:

The correct answer is B because it provides the most efficient and automated way to validate data integrity before overwriting existing files. AWS Glue Data Quality rules, defined using DQDL, allow for declarative data quality checks within the ETL pipeline itself. This eliminates the need for separate Lambda functions or manual SQL queries. Glue Data Quality can detect missing values and empty files, failing the ETL job if the data doesn't meet the defined rules. The job failure prevents the overwriting of the previous day's data with invalid information.

Option A, using Lambda, introduces additional operational overhead and complexity because it requires managing a separate function, deployment, and error handling. Option C is not robust; filling missing values with the "most common" value can introduce inaccuracies. Option D involves running a SQL query with Athena, which is a separate step and not integrated into the ETL pipeline, making it less automated and efficient. Therefore, using Glue Data Quality rules (option B) is the most streamlined, cost-effective, and reliable approach.

Here are some authoritative links to support this justification:

AWS Glue Data Quality: https://docs.aws.amazon.com/glue/latest/dg/data-quality-intro.html
DQDL Reference: https://docs.aws.amazon.com/glue/latest/dg/dqdl.html
AWS Glue ETL: https://aws.amazon.com/glue/features/

Question: 173 CertyIQ

A marketing company uses Amazon S3 to store marketing data. The company uses versioning in some buckets. The company runs several jobs to read and load data into the buckets.

To help cost-optimize its storage, the company wants to gather information about incomplete multipart uploads and outdated versions that are present in the S3 buckets.

Which solution will meet these requirements with the LEAST operational effort?

A.Use AWS CLI to gather the information.

B.Use Amazon S3 Inventory configurations reports to gather the information.

C.Use the Amazon S3 Storage Lens dashboard to gather the information.

D.Use AWS usage reports for Amazon S3 to gather the information.

Answer: C

Explanation:

The optimal solution is to use Amazon S3 Storage Lens because it provides a comprehensive, organization-wide view of object storage usage and activity trends. S3 Storage Lens offers built-in dashboards that automatically generate insights into storage usage, data tiering, activity, and data protection, including information about incomplete multipart uploads and noncurrent versions.

Option A, using the AWS CLI, would require custom scripting and manual aggregation of data across buckets, which increases operational overhead. Option B, using Amazon S3 Inventory reports, can provide detailed information about objects and their versions, but requires configuring separate inventory reports for each bucket and then aggregating the reports to get a comprehensive view, increasing complexity and effort.

Option D, using AWS Usage Reports, focuses primarily on billing and cost data, and does not provide granular insights into incomplete multipart uploads or outdated versions within S3 buckets.

S3 Storage Lens, on the other hand, offers a single pane of glass to monitor storage across the entire organization, identify cost optimization opportunities, and track data protection trends with minimal configuration. Its pre-built dashboards automatically surface insights about incomplete multipart uploads (which consume storage space unnecessarily) and noncurrent versions (which incur storage costs without offering immediate value). This aligns perfectly with the company's goal of cost optimization with the least operational effort. S3 Storage Lens allows you to view the size and count of incomplete multipart uploads and noncurrent versions, helping to identify and address inefficiencies in storage utilization. It provides a centralized and automated approach to storage analysis.

https://aws.amazon.com/s3/storage-lens/https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage_lens.html

Question: 174 CertyIQ

A gaming company uses Amazon Kinesis Data Streams to collect clickstream data. The company uses Amazon Data Firehose delivery streams to store the data in JSON format in Amazon S3. Data scientists at the company use Amazon Athena to query the most recent data to obtain business insights.

The company wants to reduce Athena costs but does not want to recreate the data pipeline.

Which solution will meet these requirements with the LEAST management effort?

A.Change the Firehose output format to Apache Parquet. Provide a custom S3 object YYYYMMDD prefix expression and specify a large buffer size. For the existing data, create an AWS Glue extract, transform, and load (ETL) job. Configure the ETL job to combine small JSON files, convert the JSON files to large Parquet files, and add the YYYYMMDD prefix. Use the ALTER TABLE ADD PARTITION statement to reflect the partition on the existing Athena table.

B.Create an Apache Spark job that combines JSON files and converts the JSON files to Apache Parquet files. Launch an Amazon EMR ephemeral cluster every day to run the Spark job to create new Parquet files in a different S3 location. Use the ALTER TABLE SET LOCATION statement to reflect the new S3 location on the existing Athena table.

C.Create a Kinesis data stream as a delivery destination for Firehose. Use Amazon Managed Service for Apache Flink (previously known as Amazon Kinesis Data Analytics) to run Apache Flink on the Kinesis data stream. Use Flink to aggregate the data and save the data to Amazon S3 in Apache Parquet format with a custom S3 object YYYYMMDD prefix. Use the ALTER TABLE ADD PARTITION statement to reflect the partition on the existing Athena table.

D.Integrate an AWS Lambda function with Firehose to convert source records to Apache Parquet and write them to Amazon S3. In parallel, run an AWS Glue extract, transform, and load (ETL) job to combine the JSON files and convert the JSON files to large Parquet files. Create a custom S3 object YYYYMMDD prefix. Use the ALTER TABLE ADD PARTITION statement to reflect the partition on the existing Athena table.

Answer: A

Explanation:

Option A is the most suitable solution because it efficiently addresses the company's requirements for reducing Athena costs with minimal management overhead. Switching Firehose's output format to Apache Parquet directly improves Athena query performance. Parquet is a columnar storage format, which allows Athena to read only the necessary columns for a query, significantly reducing data scanned and thus costs. Configuring a custom S3 object YYYYMMDD prefix enables partitioning, further optimizing query performance by allowing Athena to target specific date ranges. Specifying a large buffer size in Firehose reduces the number of small files written to S3, improving Parquet file efficiency.

For existing data, the AWS Glue ETL job is a straightforward way to convert the historical JSON data into

Parquet format, combine small files, and add the partition prefix. AWS Glue is a fully managed ETL service, reducing the management overhead compared to self-managed solutions. The ALTER TABLE ADD PARTITION statement then updates the Athena table metadata to reflect the new partitioned data. This approach requires minimal changes to the existing data pipeline while optimizing Athena query performance and costs.

Option B involves a more complex and resource-intensive approach using Amazon EMR and Spark, which adds operational overhead. Option C introduces a Kinesis Data Analytics application, which increases complexity for a task that can be accomplished with Firehose and Glue. Option D involves both Lambda and Glue ETL jobs which adds complexity. Lambda's role in Firehose is limited as it should be as thin as possible, Glue ETL handles bigger data transformations. Therefore, compared to the others, option A has the least management effort.

Relevant documentation:

Apache Parquet: https://parquet.apache.org/
Amazon Athena Cost Optimization: https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/
Amazon Kinesis Data Firehose Data Format Conversion: https://docs.aws.amazon.com/firehose/latest/dev/data-transformation.html
AWS Glue: https://aws.amazon.com/glue/

Question: 175 CertyIQ

A company needs a solution to manage costs for an existing Amazon DynamoDB table. The company also needs to control the size of the table. The solution must not disrupt any ongoing read or write operations. The company wants to use a solution that automatically deletes data from the table after 1 month.

Which solution will meet these requirements with the LEAST ongoing maintenance?

A.Use the DynamoDB TTL feature to automatically expire data based on timestamps.

B.Configure a scheduled Amazon EventBridge rule to invoke an AWS Lambda function to check for data that is older than 1 month. Configure the Lambda function to delete old data.

C.Configure a stream on the DynamoDB table to invoke an AWS Lambda function. Configure the Lambda function to delete data in the table that is older than 1 month.

D.Use an AWS Lambda function to periodically scan the DynamoDB table for data that is older than 1 month. Configure the Lambda function to delete old data.

Answer: A

Explanation:

Here's a detailed justification for why option A is the best solution:

The problem requires automatic deletion of data in a DynamoDB table after one month with minimal maintenance and no disruption to read/write operations.

Option A, using DynamoDB Time To Live (TTL), directly addresses these requirements in the most efficient way. DynamoDB TTL is a built-in feature that allows you to specify a timestamp attribute for each item. After the timestamp elapses, DynamoDB automatically deletes the item. This is a fully managed service, so it requires minimal maintenance. DynamoDB handles the deletion process in the background without impacting read or write performance.

Option B, using EventBridge and Lambda, requires creating and maintaining an EventBridge rule, a Lambda function, and the code to scan and delete data. This adds complexity and maintenance overhead compared to TTL. While functional, it is not the least maintenance option.

Option C, using DynamoDB Streams and Lambda, is triggered by every change to the table. While useful for other purposes (e.g., auditing), using it solely for data deletion is inefficient. The Lambda function would have to process every update, determine the age of the data, and then potentially delete it. This is more resource-intensive than TTL.

Option D, using a Lambda function to periodically scan the table, requires developing and maintaining the Lambda function and configuring its schedule. Scanning a DynamoDB table can be resource-intensive and can impact performance, especially if the table is large. It's also less efficient than TTL, as it involves reading the data to determine its age before potentially deleting it.

DynamoDB TTL provides a native, low-maintenance, and efficient solution for automatically expiring data based on timestamps. The other options involve extra complexity and resource utilization which do not align with the requirement for the LEAST ongoing maintenance. The solution utilizes DynamoDB's built-in capabilities, offering optimal efficiency and simplicity.

Therefore, Option A, using DynamoDB TTL, is the best solution because it is a fully managed service that handles data deletion automatically without impacting performance and requires the least amount of ongoing maintenance.

DynamoDB TTL Documentation

Question: 176 CertyIQ

A company uses Amazon S3 to store data and Amazon QuickSight to create visualizations,

The company has an S3 bucket in an AWS account named Hub-Account. The S3 bucket is encrypted by an AWS Key Management Service (AWS KMS) key. The company's QuickSight instance is in a separate account named BI-Account.

The company updates the S3 bucket policy to grant access to the QuickSight service role. The company wants to enable cross-account access to allow QuickSight to interact with the S3 bucket.

Which combination of steps will meet this requirement? (Choose two.)

A.Use the existing AWS KMS key to encrypt connections from QuickSight to the S3 bucket.

B.Add the S3 bucket as a resource that the QuickSight service role can access.

C.Use AWS Resource Access Manager (AWS RAM) to share the S3 bucket with the BI-Account account.

D.Add an IAM policy to the QuickSight service role to give QuickSight access to the KMS key that encrypts the S3 bucket.

E.Add the KMS key as a resource that the QuickSight service role can access.

Answer: BE

Explanation:

Let's break down why options B and E are the correct choices for enabling cross-account QuickSight access to an S3 bucket encrypted with KMS.

The core problem is that the S3 bucket resides in Hub-Account, the QuickSight instance in BI-Account, and the S3 bucket is encrypted with a KMS key in Hub-Account. For QuickSight to access the data, it needs both permission to access the S3 bucket and permission to decrypt the data using the KMS key.

Option B is correct because it directly addresses the S3 bucket access control. Adding the S3 bucket as a resource that the QuickSight service role can access is crucial. This involves modifying the S3 bucket policy in Hub-Account to explicitly grant the QuickSight service role (in BI-Account) permission to perform actions like

s3:GetObject. Without this, QuickSight has no authorization to even attempt to retrieve data from the bucket. Refer to https://docs.aws.amazon.com/quicksight/latest/user/iam-policy-examples.html#s3-bucket-permissions for examples of S3 bucket policies for QuickSight access.

Option E is correct because it addresses the KMS key access control. The QuickSight service role also needs permission to use the KMS key to decrypt the data. Adding the KMS key as a resource that the QuickSight service role can access is accomplished by modifying the KMS key policy in Hub-Account. This involves granting the QuickSight service role (in BI-Account) permission to perform actions like kms:Decrypt on the key. Without this, even if QuickSight can access the S3 bucket, it will be unable to decrypt the encrypted objects. See https://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html for information on KMS key policies.

Option A is incorrect because it suggests using the existing KMS key to encrypt connections. KMS keys are used for data encryption/decryption, not for encrypting connections. TLS/SSL handles the encryption of the connections.

Option C is incorrect because AWS RAM primarily shares resources within an organization or across accounts within the same organization. While RAM can share some S3 resources, it's not the typical or recommended approach for granting access to QuickSight. Explicit bucket and key policies are the standard. Also, RAM doesn't solve the KMS key access issue.

Option D is incorrect because while technically feasible, it is not the best practice. Adding an IAM policy to the QuickSight service role within the BI-Account is less secure and harder to manage in this scenario. The KMS key is in the Hub-Account, and the best practice is to control access to the KMS key at the KMS key in the Hub-Account using its key policy.

In summary, the correct combination allows the QuickSight service role in BI-Account to first, access the S3 bucket in Hub-Account (through the S3 bucket policy), and second, use the KMS key in Hub-Account to decrypt the data (through the KMS key policy).

Question: 177 CertyIQ

A car sales company maintains data about cars that are listed for sale in an area. The company receives data about new car listings from vendors who upload the data daily as compressed files into Amazon S3. The compressed files are up to 5 KB in size. The company wants to see the most up-to-date listings as soon as the data is uploaded to Amazon S3.

A data engineer must automate and orchestrate the data processing workflow of the listings to feed a dashboard. The data engineer must also provide the ability to perform one-time queries and analytical reporting. The query solution must be scalable.

Which solution will meet these requirements MOST cost-effectively?

A.Use an Amazon EMR cluster to process incoming data. Use AWS Step Functions to orchestrate workflows. Use Apache Hive for one-time queries and analytical reporting. Use Amazon OpenSearch Service to bulk ingest the data into compute optimized instances. Use OpenSearch Dashboards in OpenSearch Service for the dashboard.

B.Use a provisioned Amazon EMR cluster to process incoming data. Use AWS Step Functions to orchestrate workflows. Use Amazon Athena for one-time queries and analytical reporting. Use Amazon QuickSight for the dashboard.

C.Use AWS Glue to process incoming data. Use AWS Step Functions to orchestrate workflows. Use Amazon Redshift Spectrum for one-time queries and analytical reporting. Use OpenSearch Dashboards in Amazon OpenSearch Service for the dashboard.

D.Use AWS Glue to process incoming data. Use AWS Lambda and S3 Event Notifications to orchestrate workflows. Use Amazon Athena for one-time queries and analytical reporting. Use Amazon QuickSight for the dashboard.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most cost-effective solution for the car sales company's data processing workflow, along with supporting concepts and authoritative links:

Justification for Option D:

Option D utilizes a serverless architecture, making it the most cost-effective and scalable solution. AWS Glue can handle the data processing of small compressed files (up to 5 KB) efficiently. Glue is a fully managed ETL (Extract, Transform, Load) service that charges only for the time the ETL job runs, optimizing cost for intermittent workloads.

To orchestrate the workflow, AWS Lambda and S3 Event Notifications are used. S3 Event Notifications trigger a Lambda function when a new compressed file is uploaded to S3. The Lambda function then initiates the Glue job to process the data. This serverless event-driven approach eliminates the need for continuously running servers or clusters, further reducing costs. Lambda's pay-per-request model is ideal for handling infrequent or sporadic uploads.

For one-time queries and analytical reporting, Amazon Athena is the optimal choice. Athena allows you to query data directly from S3 using standard SQL without the need to load data into a database. Athena is serverless and pay-per-query, providing a cost-effective solution for ad-hoc analysis.

Amazon QuickSight is used for creating the dashboard. QuickSight offers interactive dashboards and visualizations with pay-per-session pricing, making it a cost-efficient BI tool.

Why other options are less optimal:

Option A (EMR, Step Functions, Hive, OpenSearch): EMR clusters are expensive to run, especially for small data volumes. Provisioning and managing an EMR cluster for processing 5 KB files is overkill and not cost-effective. OpenSearch can be costly to maintain, especially with compute-optimized instances, when the query volume is not consistent.
Option B (EMR, Step Functions, Athena, QuickSight): Similar to Option A, using a provisioned EMR cluster is excessive for small files and incurs unnecessary costs.
Option C (Glue, Step Functions, Redshift Spectrum, OpenSearch): While AWS Glue is a good choice for processing, using Step Functions as the sole orchestrator is unnecessary. Step Functions is more suited for complex workflows involving multiple services. Redshift Spectrum, while powerful, is often more expensive than Athena for one-time queries against data in S3. OpenSearch is an unnecessary complexity for direct dashboards.

Supporting Concepts & Authoritative Links:

Serverless Architecture: https://aws.amazon.com/serverless/
AWS Glue: https://aws.amazon.com/glue/
AWS Lambda: https://aws.amazon.com/lambda/
Amazon S3 Event Notifications:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html
Amazon Athena: https://aws.amazon.com/athena/
Amazon QuickSight: https://aws.amazon.com/quicksight/

In summary, the combination of AWS Glue, Lambda with S3 event triggers, Amazon Athena, and Amazon QuickSight offers a fully managed, serverless, and cost-effective solution for the given data processing workflow, analytical reporting, and dashboarding requirements.

Question: 178 CertyIQ

A company has AWS resources in multiple AWS Regions. The company has an Amazon EFS file system in each Region where the company operates. The company's data science team operates within only a single Region. The data that the data science team works with must remain within the team's Region.

A data engineer needs to create a single dataset by processing files that are in each of the company's Regional EFS file systems. The data engineer wants to use an AWS Step Functions state machine to orchestrate AWS Lambda functions to process the data.

Which solution will meet these requirements with the LEAST effort?

A.Peer the VPCs that host the EFS file systems in each Region with the VPC that is in the data science team's Region. Enable EFS file locking. Configure the Lambda functions in the data science team's Region to mount each of the Region specific file systems. Use the Lambda functions to process the data.

B.Configure each of the Regional EFS file systems to replicate data to the data science team's Region. In the data science team's Region, configure the Lambda functions to mount the replica file systems. Use the Lambda functions to process the data.

C.Deploy the Lambda functions to each Region. Mount the Regional EFS file systems to the Lambda functions. Use the Lambda functions to process the data. Store the output in an Amazon S3 bucket in the data science team's Region.

D.Use AWS DataSync to transfer files from each of the Regional EFS files systems to the file system that is in the data science team's Region. Configure the Lambda functions in the data science team's Region to mount the file system that is in the same Region. Use the Lambda functions to process the data.

Answer: A

Explanation:

The correct answer is A. Here's why:

Least Effort: The requirement emphasizes minimizing effort. VPC peering is a relatively straightforward process for connecting VPCs across regions.
Data Locality: The question specifies that data must remain within the data science team's Region. Option A allows the data science team's Lambda functions, running within their Region, to access data residing in the other Regional EFS file systems without moving it. The other options involve data transfer or replication.
Direct Access: By peering VPCs, the Lambda functions in the data science team's region can directly mount the remote EFS file systems and access the data as if it were local, eliminating the need for data transfer steps.
EFS File Locking: Enabling EFS file locking ensures data consistency and prevents data corruption when multiple Lambda functions concurrently access and process files across different regions.

Why other options are less suitable:

B: EFS Replication requires more configuration and management effort compared to VPC peering. Also, the goal is to process the data centrally, not to replicate it.

C: Deploying Lambda functions to each Region significantly increases operational complexity (more code deployments, logging, monitoring, etc.) and doesn't centralize processing in the data science team's Region as required. It also requires transferring output to the data science team's region, adding complexity.

D: Using DataSync introduces an unnecessary data transfer step, increasing complexity and cost. DataSync is suitable for data migration and synchronization, but not for direct processing in this scenario.

Authoritative Links:

VPC Peering: https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html
Amazon EFS: https://aws.amazon.com/efs/
AWS Lambda: https://aws.amazon.com/lambda/

AWS Step Functions: https://aws.amazon.com/step-functions/

Question: 179 CertyIQ

A company hosts its applications on Amazon EC2 instances. The company must use SSL/TLS connections that encrypt data in transit to communicate securely with AWS infrastructure that is managed by a customer.

A data engineer needs to implement a solution to simplify the generation, distribution, and rotation of digital certificates. The solution must automatically renew and deploy SSL/TLS certificates.

Which solution will meet these requirements with the LEAST operational overhead?

A.Store self-managed certificates on the EC2 instances.

B.Use AWS Certificate Manager (ACM).

C.Implement custom automation scripts in AWS Secrets Manager.

D.Use Amazon Elastic Container Service (Amazon ECS) Service Connect.

Answer: B

Explanation:

The best solution to simplify SSL/TLS certificate management with the least operational overhead is **B. Use AWS Certificate Manager (ACM).**

ACM is a service designed specifically for managing SSL/TLS certificates. It handles the complexities of certificate generation, deployment, and renewal, significantly reducing the burden on the data engineer. ACM offers both public and private certificate authorities (CAs). For securing communication with AWS infrastructure managed by a customer, a private CA could be more appropriate if the customer uses only internally trusted certificates.

ACM can automatically renew SSL/TLS certificates that are used with ACM-integrated services like Elastic Load Balancing (ELB), Amazon CloudFront, Amazon API Gateway, and others. It integrates seamlessly with these services, making the deployment process straightforward. ACM also provides certificate lifecycle management, including automated renewal, which minimizes the risk of expired certificates disrupting operations.

Option A, storing self-managed certificates on EC2 instances, introduces significant operational overhead. The data engineer would need to manually generate, distribute, renew, and deploy the certificates, a process that is prone to errors and requires constant monitoring.

Option C, implementing custom automation scripts in AWS Secrets Manager, could automate some aspects of certificate management, but it's more complex than using ACM directly. Secrets Manager is primarily for storing secrets, not a dedicated certificate management service. Building and maintaining custom scripts adds operational overhead.

Option D, Amazon ECS Service Connect, is designed for service-to-service communication within an ECS cluster. It does handle traffic encryption, but isn't directly used to issue/manage certificates for external/customer managed infrastructure; thus isn't the best option in this context.

ACM directly addresses the need for simplified generation, distribution, and rotation of SSL/TLS certificates while minimizing operational overhead through its automation and integration capabilities.

Relevant Links:

AWS Certificate Manager (ACM)
ACM Documentation

Question: 180 CertyIQ

A company saves customer data to an Amazon S3 bucket. The company uses server-side encryption with AWS KMS keys (SSE-KMS) to encrypt the bucket. The dataset includes personally identifiable information (PII) such as social security numbers and account details.

Data that is tagged as PII must be masked before the company uses customer data for analysis. Some users must have secure access to the PII data during the pre-processing phase. The company needs a low-maintenance solution to mask and secure the PII data throughout the entire engineering pipeline.

Which combination of solutions will meet these requirements? (Choose two.)

A.Use AWS Glue DataBrew to perform extract, transform, and load (ETL) tasks that mask the PII data before analysis.

B.Use Amazon GuardDuty to monitor access patterns for the PII data that is used in the engineering pipeline.

C.Configure an Amazon Macie discovery job for the S3 bucket.

D.Use AWS Identity and Access Management (IAM) to manage permissions and to control access to the PII data.

E.Write custom scripts in an application to mask the PII data and to control access.

Answer: AD

Explanation:

Here's a detailed justification for why options A and D are the correct solutions, and why the others are not, in the context of securely masking PII data during an engineering pipeline within AWS.

Why A is Correct: AWS Glue DataBrew for ETL and Masking

AWS Glue DataBrew is a visual data preparation tool that allows you to clean and normalize data without writing code. It's ideal for masking PII because it provides built-in recipes and transformations specifically designed for data masking, such as tokenization, redaction, and encryption. DataBrew integrates directly with S3, allowing you to access the PII data, apply the masking transformations during the ETL process, and then write the masked data back to S3 or another data store. Because the masking happens within a managed service, it's a low-maintenance solution. This enables you to maintain a single point of masking, which is beneficial for compliance. https://aws.amazon.com/glue/databrew/

Why D is Correct: IAM for Access Control

AWS Identity and Access Management (IAM) is fundamental for controlling access to AWS resources. IAM allows you to define fine-grained policies that specify which users or groups have access to the S3 bucket containing the PII data, and what actions they are permitted to perform (e.g., read, write, delete). IAM policies can be used to grant specific users access to the original, unmasked PII data for pre-processing while restricting access to the raw data for other users. This ensures that only authorized personnel have access to the sensitive data during the necessary phases, enhancing security and compliance.
https://aws.amazon.com/iam/

Why B is Incorrect: Amazon GuardDuty for Threat Detection (Not Masking)

Amazon GuardDuty is a threat detection service. It monitors your AWS environment for malicious activity and unauthorized behavior. While it's helpful for security, it doesn't directly address the requirement of masking PII data. GuardDuty would be useful for detecting unusual access to the PII data, it does not mask or control data access as granularly as IAM.

Why C is Incorrect: Amazon Macie for PII Discovery (Not Masking)

Amazon Macie helps discover and classify sensitive data in S3 buckets. While it identifies PII, it doesn't

provide the functionality to mask or transform the data. Macie is useful for finding the data that needs masking, but it doesn't perform the masking itself.

Why E is Incorrect: Custom Scripts (High Maintenance)

Writing custom scripts to mask PII data can be complex and prone to errors. It also introduces a significant maintenance overhead, as you're responsible for managing, updating, and securing the scripts. This solution is not low-maintenance. Furthermore, managing access control through custom scripts can be less secure and harder to audit compared to using IAM.

Question: 181 CertyIQ

A data engineer is launching an Amazon EMR cluster. The data that the data engineer needs to load into the new cluster is currently in an Amazon S3 bucket. The data engineer needs to ensure that data is encrypted both at rest and in transit.

The data that is in the S3 bucket is encrypted by an AWS Key Management Service (AWS KMS) key. The data engineer has an Amazon S3 path that has a Privacy Enhanced Mail (PEM) file.

Which solution will meet these requirements?

A.Create an Amazon EMR security configuration. Specify the appropriate AWS KMS key for at-rest encryption for the S3 bucket. Create a second security configuration. Specify the Amazon S3 path of the PEM file for in-transit encryption. Create the EMR cluster, and attach both security configurations to the cluster.

B.Create an Amazon EMR security configuration. Specify the appropriate AWS KMS key for local disk encryption for the S3 bucket. Specify the Amazon S3 path of the PEM file for in-transit encryption. Use the security configuration during EMR cluster creation.

C.Create an Amazon EMR security configuration. Specify the appropriate AWS KMS key for at-rest encryption for the S3 bucket. Specify the Amazon S3 path of the PEM file for in-transit encryption. Use the security configuration during EMR cluster creation.

D.Create an Amazon EMR security configuration. Specify the appropriate AWS KMS key for at-rest encryption for the S3 bucket. Specify the Amazon S3 path of the PEM file for in-transit encryption. Create the EMR cluster, and attach the security configuration to the cluster.

Answer: C

Explanation:

The correct solution is **C** because it leverages EMR's security configurations to centrally manage both at-rest and in-transit encryption.

Here's a breakdown of why:

EMR Security Configurations: Amazon EMR security configurations provide a consolidated way to specify encryption settings, authentication, and authorization configurations for an EMR cluster. https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-security-configurations.html
At-Rest Encryption with KMS: EMR supports using AWS KMS keys to encrypt data at rest in S3, as well as local disks attached to the EMR cluster nodes. Since the data is already encrypted at rest in S3 with a KMS key, the security configuration will use this key.
In-Transit Encryption: EMR enables encryption of data in transit between nodes within the cluster. A common method for this is to use Transport Layer Security (TLS) with a certificate. The PEM file provided contains the certificate needed to establish secure TLS connections. The security configuration in EMR can be used to specify the S3 path to this certificate. https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-configure-data-in-transit-encryption.html
Single Configuration: Option C uses a single security configuration for both, providing a more organized and manageable approach.

Why other options are incorrect:

Option A: Creating two separate security configurations is unnecessarily complex and not the intended use of EMR's security configurations. The whole point of security configurations is to have a single source of truth for security settings.

Option B: The KMS key is used for S3 at-rest encryption, not just local disk encryption. Although you can configure the KMS key for local disk encryption, this scenario is for S3 at-rest encryption

Option D: This option correctly creates a single security configuration, but then attempts to "attach" it to the cluster. In reality, the security configuration is specified during cluster creation, not attached afterwards.

Question: 182 CertyIQ

A retail company is using an Amazon Redshift cluster to support real-time inventory management. The company has deployed an ML model on a real-time endpoint in Amazon SageMaker.

The company wants to make real-time inventory recommendations. The company also wants to make predictions about future inventory needs.

Which solutions will meet these requirements? (Choose two.)

A.Use Amazon Redshift ML to generate inventory recommendations.
B.Use SQL to invoke a remote SageMaker endpoint for prediction.
C.Use Amazon Redshift ML to schedule regular data exports for offline model training.
D.Use SageMaker Autopilot to create inventory management dashboards in Amazon Redshift.
E.Use Amazon Redshift as a file storage system to archive old inventory management reports.

Answer: AB

Explanation:

The question asks for solutions to generate real-time inventory recommendations and predict future inventory needs using an Amazon Redshift cluster and a SageMaker endpoint.

Option A is correct because Amazon Redshift ML allows users to leverage existing ML models deployed in SageMaker directly from Redshift using SQL. This enables the company to generate real-time inventory recommendations based on the live data in Redshift by querying the SageMaker endpoint through Redshift ML. https://docs.aws.amazon.com/redshift/latest/dg/machine-learning.html

Option B is correct because SQL can be used to invoke a remote SageMaker endpoint for prediction. Redshift ML allows users to create SQL functions that call a SageMaker endpoint. This is a direct way to obtain real-time predictions from the ML model deployed in SageMaker, which is necessary for inventory recommendations. The function takes input data from Redshift, sends it to SageMaker, and returns the prediction result.

Option C is incorrect because while exporting data for offline model training is relevant for improving model accuracy, it doesn't directly address the requirement of real-time inventory recommendations. It's more related to model maintenance and updates, not real-time prediction.

Option D is incorrect because SageMaker Autopilot is used for automated model building and tuning. It doesn't directly create dashboards inside Amazon Redshift. While Autopilot could help build a better model that's used for inventory management, it doesn't create dashboards directly in Redshift, nor does it provide real-time recommendations.

Option E is incorrect because using Amazon Redshift as a file storage system for archiving reports is not a primary function of Redshift. Redshift is optimized for analytical workloads, not general-purpose file storage. S3 or a similar object storage solution would be more suitable for archiving. Furthermore, archiving reports doesn't address the need for real-time recommendations or future inventory predictions.

Question: 183 CertyIQ

A company stores CSV files in an Amazon S3 bucket. A data engineer needs to process the data in the CSV files and store the processed data in a new S3 bucket.

The process needs to rename a column, remove specific columns, ignore the second row of each file, create a new column based on the values of the first row of the data, and filter the results by a numeric value of a column.

Which solution will meet these requirements with the LEAST development effort?

A.Use AWS Glue Python jobs to read and transform the CSV files.
B.Use an AWS Glue custom crawler to read and transform the CSV files.
C.Use an AWS Glue workflow to build a set of jobs to crawl and transform the CSV files.
D.Use AWS Glue DataBrew recipes to read and transform the CSV files.

Answer: D

Explanation:

The correct answer is D: Use AWS Glue DataBrew recipes to read and transform the CSV files. Let's break down why this is the best approach with the least development effort.

AWS Glue DataBrew is specifically designed for data preparation tasks without requiring code. Its visual interface and pre-built transformations make it ideal for the required transformations: renaming columns, removing columns, ignoring rows, creating new columns based on existing data, and filtering. Each of these can be accomplished using DataBrew's built-in recipe steps. The "Ignore the second row" requirement can be achieved by using filtering based on row number or by explicitly dropping it using a custom formula. Creating a new column based on the first row involves extracting the first row, defining a formula using its value, and applying it to subsequent rows.

Option A, using AWS Glue Python jobs, requires writing and debugging Python code using Spark or Pandas DataFrames, which is a more complex development task. Option B, employing a Glue custom crawler, is focused on metadata discovery, not data transformation. While a crawler can identify the schema, it wouldn't directly perform the transformation steps. Option C, utilizing a Glue workflow with crawler and transform jobs, is also more complex than necessary, as it involves managing multiple components and writing code for the transformation job.

DataBrew significantly reduces development time and complexity by offering a visual, no-code/low-code approach. It aligns directly with the problem statement's requirements, making it the solution with the least development effort. This is because DataBrew is designed for business users and data analysts who might not have extensive coding experience.

AWS Glue DataBrew Documentation provides further information on its capabilities.

Question: 184 CertyIQ

A company uses Amazon Redshift as its data warehouse. Data encoding is applied to the existing tables of the data warehouse. A data engineer discovers that the compression encoding applied to some of the tables is not the

best fit for the data.

The data engineer needs to improve the data encoding for the tables that have sub-optimal encoding.

Which solution will meet this requirement?

A.Run the ANALYZE command against the identified tables. Manually update the compression encoding of columns based on the output of the command.

B.Run the ANALYZE COMPRESSION command against the identified tables. Manually update the compression encoding of columns based on the output of the command.

C.Run the VACUUM REINDEX command against the identified tables.

D.Run the VACUUM RECLUSTER command against the identified tables.

Answer: B

Explanation:

The correct answer is **B. Run the ANALYZE COMPRESSION command against the identified tables. Manually update the compression encoding of columns based on the output of the command.** Here's why:

Amazon Redshift's data encoding significantly impacts storage efficiency and query performance. Selecting appropriate encoding helps reduce storage space and improve query speed. When encoding is suboptimal, queries become slower, and storage costs increase.

The ANALYZE COMPRESSION command is specifically designed to analyze existing table data and suggest optimal compression encodings for each column in a Redshift table. This command samples the data in each column and generates recommendations based on the data type, cardinality, and distribution of values. This analysis helps identify columns where the current encoding isn't the most efficient.

After running ANALYZE COMPRESSION, the data engineer needs to review the suggested encoding and then manually alter the table schema using the ALTER TABLE command to apply the new encoding for each column.

Option A is incorrect because while ANALYZE gathers statistics about the table, it doesn't provide specific compression encoding recommendations. Option C VACUUM REINDEX command only helps with index defragmentation, which doesn't alter or optimize the data encodings. Option D VACUUM RECLUSTER sorts the data based on the sort key, improving the effectiveness of zone maps, but does not directly address suboptimal encoding choices. The VACUUM command's primary use is to reclaim space and sort data rather than analyzing compression effectiveness. Therefore, only ANALYZE COMPRESSION helps identify the problematic encodings, fulfilling the requirement.

Authoritative Links:

Amazon Redshift ANALYZE COMPRESSION:
https://docs.aws.amazon.com/redshift/latest/dg/r_ANALYZE_COMPRESSION.html
Amazon Redshift Data Encoding: https://docs.aws.amazon.com/redshift/latest/dg/c_compression.html

Question: 185 CertyIQ

The company stores a large volume of customer records in Amazon S3. To comply with regulations, the company must be able to access new customer records immediately for the first 30 days after the records are created. The company accesses records that are older than 30 days infrequently.

The company needs to cost-optimize its Amazon S3 storage.

Which solution will meet these requirements MOST cost-effectively?

A.Apply a lifecycle policy to transition records to S3 Standard Infrequent-Access (S3 Standard-IA) storage after 30 days.

B.Use S3 Intelligent-Tiering storage.

C.Transition records to S3 Glacier Deep Archive storage after 30 days.

D.Use S3 Standard-Infrequent Access (S3 Standard-IA) storage for all customer records.

Answer: A

Explanation:

The most cost-effective solution is to use a lifecycle policy to transition records to S3 Standard-IA after 30 days. This directly addresses the requirement of infrequent access for records older than 30 days while keeping the recent records (within 30 days) readily available in S3 Standard for immediate access.

Option A leverages S3 lifecycle policies to automatically move data between storage classes. This eliminates the need for manual intervention, improving operational efficiency. S3 Standard-IA provides lower storage costs than S3 Standard for data that is accessed less frequently, making it a suitable choice for records older than 30 days. Because records less than 30 days old are accessed immediately, S3 Standard is a good choice.

Option B, S3 Intelligent-Tiering, automatically moves data between frequent, infrequent, and archive access tiers based on access patterns. While this could work, it isn't most cost-effective because there's a monitoring overhead. Since the access pattern is pre-defined (immediate for 30 days, then infrequent), a lifecycle policy is a simpler and less expensive solution. S3 Intelligent-Tiering can also have a small per-object monitoring and automation charge.

Option C, S3 Glacier Deep Archive, is designed for long-term archival and retrieval is measured in hours. This makes it unsuitable for accessing records older than 30 days if immediate access is needed at any point, even infrequently. Transitioning data into Glacier incurs retrieval costs when accessing data.

Option D, using S3 Standard-IA for all records, would mean that records accessed immediately (within the first 30 days) are stored in a less performant tier. S3 Standard-IA has a per-GB retrieval fee, making it more expensive than S3 Standard for records accessed immediately and frequently. While it addresses the infrequent access requirement post 30 days, it's not optimal from a cost perspective for the initial 30 days where frequent access is needed.

In summary, the lifecycle policy offers the most targeted and cost-efficient way to balance immediate access for new records with cost-effective storage for older, infrequently accessed records. It aligns precisely with the stated requirements and constraints.

Further reading:

S3 Lifecycle Policies
S3 Storage Classes

Question: 186 CertyIQ

A data engineer is using Amazon QuickSight to build a dashboard to report a company's revenue in multiple AWS Regions. The data engineer wants the dashboard to display the total revenue for a Region, regardless of the drill-down levels shown in the visual.

Which solution will meet these requirements?

A.Create a table calculation.

B.Create a simple calculated field.

C.Create a level-aware calculation - aggregate (LAC-A) function.

D.Create a level-aware calculation - window (LAC-W) function.

Answer: C

Explanation:

The correct answer is C, creating a level-aware calculation - aggregate (LAC-A) function. Level-aware calculations in Amazon QuickSight allow you to control how aggregations are performed based on the levels of granularity present in your visual. In this scenario, the data engineer needs to calculate the total revenue for a Region, irrespective of any drill-down levels.

LAC-A functions are specifically designed for scenarios where you want an aggregation to consider a particular level (in this case, Region) while the visual might be showing data at a more granular level (e.g., by product or by day within a region). The LAC-A function computes the aggregate at the specified level, and the result is the same even when the visualization is drilled down further. This means the total revenue for a Region will always be displayed, regardless of how the visual is filtered or expanded.

A simple calculated field (option B) performs calculations at the level of the data in the dataset and won't necessarily preserve the aggregate for the Region when drilling down. A table calculation (option A) performs calculations on the visual table itself, and its result is heavily dependent on the current drill-down level. It's not suited for maintaining a Region-level total. LAC-W functions (option D) calculate values over a window of rows, primarily useful for moving averages or running totals, not maintaining a constant aggregate across drill-down levels. Therefore, LAC-A is the correct approach for calculating and displaying regional totals consistently, even during drill-downs.

For further reading on level-aware calculations in QuickSight, refer to the AWS documentation:

Understanding Level-Aware Calculations in Amazon QuickSight:
https://docs.aws.amazon.com/quicksight/latest/user/level-aware-calculations.html

Question: 187 CertyIQ

A retail company stores customer data in an Amazon S3 bucket. Some of the customer data contains personally identifiable information (PII) about customers. The company must not share PII data with business partners.

A data engineer must determine whether a dataset contains PII before making objects in the dataset available to business partners.

Which solution will meet this requirement with the LEAST manual intervention?

A.Configure the S3 bucket and S3 objects to allow access to Amazon Macie. Use automated sensitive data discovery in Macie.

B.Configure AWS CloudTrail to monitor S3 PUT operations. Inspect the CloudTrail trails to identify operations that save PII.

C.Create an AWS Lambda function to identify PII in S3 objects. Schedule the function to run periodically.

D.Create a table in AWS Glue Data Catalog. Write custom SQL queries to identify PII in the table. Use Amazon Athena to run the queries.

Answer: A

Explanation:

Here's a detailed justification for why option A is the best solution for automatically identifying PII in an S3 bucket with the least manual intervention:

Option A leverages Amazon Macie's built-in capabilities for sensitive data discovery. Macie is specifically

designed for this purpose, offering automated scanning of S3 buckets to identify PII and other sensitive data types. By configuring the S3 bucket and objects to allow access to Macie, the company can utilize Macie's pre-built managed data identifiers and/or create custom data identifiers to accurately locate PII based on predefined patterns and keywords. This automated discovery significantly reduces manual effort compared to other options.

The alternative approaches necessitate more manual configuration and ongoing maintenance. CloudTrail (option B) primarily focuses on auditing API calls, not the content of objects. While PUT operations can be monitored, inspecting trails to determine if PII was saved would require significant manual effort and expertise in analyzing audit logs and understanding data patterns, making it less practical and efficient.

Creating a Lambda function (option C) demands development and maintenance of custom code to identify PII. This introduces complexity and potential for errors, especially as data formats evolve. Furthermore, scheduling and monitoring the Lambda function adds operational overhead.

Using AWS Glue and Athena (option D) involves creating tables in the Glue Data Catalog and writing custom SQL queries. While powerful, this approach requires significant upfront investment in data modeling, query development, and ongoing maintenance of these queries as data characteristics change. It's less flexible and requires more specialized skills than using Macie.

Macie's automated sensitive data discovery reduces operational overhead, ensures consistent and accurate PII identification, and simplifies compliance efforts, making it the most efficient and least manual solution.

Authoritative Links:

Amazon Macie: https://aws.amazon.com/macie/
Macie documentation: https://docs.aws.amazon.com/macie/latest/userguide/what-is-macie.html

Question: 188 CertyIQ

A data engineer needs to create an empty copy of an existing table in Amazon Athena to perform data processing tasks. The existing table in Athena contains 1,000 rows.

Which query will meet this requirement?

A.CREATE TABLE new_table -
LIKE old_table;

B.CREATE TABLE new_table -
AS SELECT *

FROM old_table -
WITH NO DATA;

C.CREATE TABLE new_table -
AS SELECT *
FROM old_table;

D.CREATE TABLE new_table -
as SELECT *

FROM old_cable -
WHERE 1=1;

Answer: B

Explanation:

The objective is to create an empty copy of an existing Athena table. The correct approach is to use the CREATE TABLE AS SELECT (CTAS) statement with the WITH NO DATA clause. This method creates a new

table with the same schema (column definitions, data types) as the existing table but without copying any of the data rows.

Option A, CREATE TABLE new_table LIKE old_table;, only copies the table metadata, such as schema, comments, and table properties. Crucially, it does not copy table properties such as partitioning or bucketed information, which might be important. It is useful but insufficient for a "true" empty copy intended for processing that relies on the original table's structure and properties related to data layout.

Option B, CREATE TABLE new_table AS SELECT * FROM old_table WITH NO DATA;, accurately creates a new table with the same schema and table properties as the old_table, but importantly, does not populate it with data from the old table, leaving it empty as required. The WITH NO DATA clause is the key element. This satisfies the complete requirement: schema replication and an empty table.

Option C, CREATE TABLE new_table AS SELECT * FROM old_table;, would create a new table and populate it with all 1,000 rows from the existing table. This defeats the requirement of creating an empty copy.

Option D, CREATE TABLE new_table AS SELECT * FROM old_table WHERE 1=1;, creates a new table and populates it with data from the existing table because the WHERE 1=1 condition is always true; consequently, all rows are selected. Like option C, this results in a copy of the data, violating the "empty" requirement.

Therefore, the only statement that creates a new, empty table with the schema of the old table is Option B.

Refer to the AWS Athena documentation for detailed information on CTAS statements:
https://docs.aws.amazon.com/athena/latest/ug/create-table-as.html

Question: 189 CertyIQ

A company has a data lake in Amazon S3. The company collects AWS CloudTrail logs for multiple applications. The company stores the logs in the data lake, catalogs the logs in AWS Glue, and partitions the logs based on the year. The company uses Amazon Athena to analyze the logs.

Recently, customers reported that a query on one of the Athena tables did not return any data. A data engineer must resolve the issue.

Which combination of troubleshooting steps should the data engineer take? (Choose two.)

A.Confirm that Athena is pointing to the correct Amazon S3 location.
B.Increase the query timeout duration.
C.Use the MSCK REPAIR TABLE command.
D.Restart Athena.
E.Delete and recreate the problematic Athena table.

Answer: AC

Explanation:

The correct answer is A and C. Here's why:

A. Confirm that Athena is pointing to the correct Amazon S3 location:

Athena relies on the AWS Glue Data Catalog to understand the schema and location of the data stored in S3. If Athena is pointing to the wrong S3 location, it won't be able to find the data, resulting in an empty result set. This is a fundamental step in troubleshooting any Athena query issue. It verifies that Athena is looking at the intended data source. The Glue Data Catalog stores the metadata of the data residing in S3, guiding Athena's query engine. A mismatch between the location specified in the Glue table and the actual data location in S3

is a common cause of such problems.

C. Use the MSCK REPAIR TABLE command:

MSCK REPAIR TABLE is crucial for synchronizing the partition information in the AWS Glue Data Catalog with the actual directory structure in S3. If new partitions (e.g., a new year's worth of CloudTrail logs) are added to S3 without updating the Glue Data Catalog, Athena won't be aware of these new partitions. This leads to Athena querying only the partitions it knows about, potentially missing data and returning an empty result set if the query targets the missing partitions. MSCK REPAIR TABLE automatically discovers and registers these new partitions within the Glue Data Catalog, allowing Athena to correctly query all data.

Why the other options are incorrect:

B. Increase the query timeout duration: Increasing the timeout duration might help if the query is taking a long time to process, but it won't resolve the issue if Athena cannot find the data in the first place. A missing dataset due to incorrect data location or un-crawled partitions isn't related to query timeout.
D. Restart Athena: Athena is a serverless query service. Restarting it has no effect on data visibility or query results. It's not a stateful service that caches data locally.
E. Delete and recreate the problematic Athena table: While recreating the table might resolve the issue if there's a problem with the table definition, it's a more drastic step than MSCK REPAIR TABLE. MSCK REPAIR TABLE updates the existing table's partition information and should be attempted first. Moreover, recreating the table will not fix the problem if the location or underlying data is incorrect. It's best practice to troubleshoot before resorting to more disruptive solutions.

Authoritative Links:

AWS Glue Data Catalog: https://docs.aws.amazon.com/glue/latest/dg/datacatalog-overview.html
Athena MSCK REPAIR TABLE command: https://docs.aws.amazon.com/athena/latest/ug/msck-repair-table.html
Troubleshooting Athena Queries: https://aws.amazon.com/premiumsupport/knowledge-center/athena-query-fails/

Question: 190 CertyIQ

A data engineer wants to orchestrate a set of extract, transform, and load (ETL) jobs that run on AWS. The ETL jobs contain tasks that must run Apache Spark jobs on Amazon EMR, make API calls to Salesforce, and load data into Amazon Redshift.

The ETL jobs need to handle failures and retries automatically. The data engineer needs to use Python to orchestrate the jobs.

Which service will meet these requirements?

A.Amazon Managed Workflows for Apache Airflow (Amazon MWAA)
B.AWS Step Functions
C.AWS Glue
D.Amazon EventBridge

Answer: A

Explanation:

The correct answer is A: Amazon Managed Workflows for Apache Airflow (Amazon MWAA). Here's why:

Amazon MWAA excels at orchestrating complex workflows involving various services. Airflow is a popular

open-source platform specifically designed for programmatically authoring, scheduling, and monitoring workflows. MWAA simplifies running Airflow by managing the underlying infrastructure, including scaling, security, and updates. In the described scenario, the data engineer requires a solution to manage a set of ETL jobs that include diverse tasks: running Spark jobs on EMR, making API calls to Salesforce, and loading data into Redshift. Airflow provides operators and integrations to seamlessly interact with each of these services. Airflow also provides built-in support for failure handling and retries, as specified in the requirements. The data engineer's preference for using Python aligns perfectly with Airflow, as Airflow workflows (DAGs) are defined using Python code.

While AWS Step Functions (Option B) can also orchestrate workflows, it's often better suited for simpler state-based workflows and may require more custom coding to integrate with EMR and external services like Salesforce. AWS Glue (Option C) primarily focuses on data integration and ETL processes but lacks the broad workflow orchestration capabilities of Airflow. AWS EventBridge (Option D) is primarily for event-driven architectures and doesn't provide the necessary workflow management and orchestration features needed for this complex ETL process. Therefore, Amazon MWAA is the most suitable choice due to its specialized capabilities for scheduling and monitoring complex workflows, its support for Python, and its seamless integration with the required AWS services and external APIs.

Further Reading:

Amazon MWAA: https://aws.amazon.com/managed-workflows-for-apache-airflow/
Apache Airflow: https://airflow.apache.org/

Question: 191 CertyIQ

A data engineer maintains custom Python scripts that perform a data formatting process that many AWS Lambda functions use. When the data engineer needs to modify the Python scripts, the data engineer must manually update all the Lambda functions.

The data engineer requires a less manual way to update the Lambda functions.

Which solution will meet this requirement?

A.Store the custom Python scripts in a shared Amazon S3 bucket. Store a pointer to the custom scripts in the execution context object.

B.Package the custom Python scripts into Lambda layers. Apply the Lambda layers to the Lambda functions.

C.Store the custom Python scripts in a shared Amazon S3 bucket. Store a pointer to the customer scripts in environment variables.

D.Assign the same alias to each Lambda function. Call each Lambda function by specifying the function's alias.

Answer: B

Explanation:

The optimal solution for centrally managing and updating shared Python scripts across multiple AWS Lambda functions is to utilize Lambda Layers (Option B). Lambda Layers offer a mechanism to package dependencies, such as shared code, libraries, or data, separately from the Lambda function's deployment package. This separation allows multiple Lambda functions to reference and utilize the same shared resources.

When the data engineer updates the Python scripts within the Lambda Layer, only the layer needs to be updated. This update propagates to all Lambda functions that are configured to use that layer. This eliminates the need to individually update each Lambda function deployment package, significantly reducing manual effort and potential errors. Using Lambda Layers promotes code reusability and simplifies the management of shared components within a serverless architecture. The updated layer versions are made available to the functions attached to the layer either immediately or by updating the Layer version referenced by the

function.

Options A and C, which involve storing scripts in S3 and referencing them via execution context objects or environment variables, introduce significant operational overhead. Each Lambda function would still need to fetch the script from S3 during each execution, adding latency and complexity. These solutions also do not inherently offer a streamlined update mechanism. When the script is updated in S3, all the Lambda functions need to redeploy with new environment variables or execution context.

Option D, using Lambda function aliases, primarily serves for version management and traffic routing, and doesn't address the central management of shared code dependencies. It won't help share common code among functions or ease the update process when this shared code changes. Aliases focus on managing versions of a specific Lambda function, not sharing code across multiple independent Lambda functions.

In summary, Lambda Layers provide the most efficient and scalable solution for managing shared code dependencies across multiple AWS Lambda functions. They simplify updates, promote code reuse, and reduce operational overhead compared to other approaches.

Supporting Documentation:

AWS Lambda Layers: https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

Question: 192 CertyIQ

A company stores customer data in an Amazon S3 bucket. Multiple teams in the company want to use the customer data for downstream analysis. The company needs to ensure that the teams do not have access to personally identifiable information (PII) about the customers.

Which solution will meet this requirement with LEAST operational overhead?

A.Use Amazon Macie to create and run a sensitive data discovery job to detect and remove PII.

B.Use S3 Object Lambda to access the data, and use Amazon Comprehend to detect and remove PII.

C.Use Amazon Data Firehose and Amazon Comprehend to detect and remove PII.

D.Use an AWS Glue DataBrew job to store the PII data in a second S3 bucket. Perform analysis on the data that remains in the original S3 bucket.

Answer: A

Explanation:

The question requires a solution that removes PII from customer data stored in S3 before multiple teams use it for analysis, minimizing operational overhead.

Option A (Amazon Macie) is the most suitable choice because Macie is specifically designed to discover, classify, and protect sensitive data in S3. It can be used to identify PII using built-in and custom data identifiers. Once detected, Macie can trigger actions such as removing the PII or redacting it directly within the S3 objects. Macie's automated discovery and remediation capabilities minimize manual intervention, resulting in lower operational overhead compared to other solutions.

Option B (S3 Object Lambda and Amazon Comprehend) involves accessing the data through S3 Object Lambda, which is useful but requires more configurations. Then, Comprehend would have to be integrated to detect and remove PII which adds complexity. S3 Object Lambda is more applicable for real-time transformations than pre-emptive PII removal.

Option C (Amazon Data Firehose and Amazon Comprehend) is less ideal because Data Firehose is primarily for streaming data into destinations like S3, Redshift, and Elasticsearch. While it can integrate with Comprehend, it is designed for near real-time data ingestion, not for performing batch PII removal on existing data in S3.

Using Firehose for this purpose would be inefficient and more complex.

Option D (AWS Glue DataBrew) is incorrect because it focuses on storing the PII data, which directly contradicts the requirement to ensure teams don't have access to it. It aims to move PII, not remove it, and requires creating separate data pipelines and infrastructure, which increases operational overhead.

Therefore, Amazon Macie offers the most streamlined and purpose-built solution to discover and handle PII data directly within S3, thus minimizing operational overhead while meeting the stated requirements.

Supporting links:

Amazon Macie: https://aws.amazon.com/macie/
S3 Object Lambda: https://aws.amazon.com/s3/object-lambda/
Amazon Comprehend: https://aws.amazon.com/comprehend/
Amazon Data Firehose: https://aws.amazon.com/firehose/
AWS Glue DataBrew: https://aws.amazon.com/glue/databrew/

Question: 193 CertyIQ

A company stores its processed data in an S3 bucket. The company has a strict data access policy. The company uses IAM roles to grant teams within the company different levels of access to the S3 bucket.

The company wants to receive notifications when a user violates the data access policy. Each notification must include the username of the user who violated the policy.

Which solution will meet these requirements?

A.Use AWS Config rules to detect violations of the data access policy. Set up compliance alarms.

B.Use Amazon CloudWatch metrics to gather object-level metrics. Set up CloudWatch alarms.

C.Use AWS CloudTrail to track object-level events for the S3 bucket. Forward events to Amazon CloudWatch to set up CloudWatch alarms.

D.Use Amazon S3 server access logs to monitor access to the bucket. Forward the access logs to an Amazon CloudWatch log group. Use metric filters on the log group to set up CloudWatch alarms.

Answer: C

Explanation:

Here's a detailed justification for why option C is the correct solution, along with why the other options are less suitable:

Option C leverages the combined power of AWS CloudTrail and Amazon CloudWatch to achieve the desired outcome of detecting data access policy violations and notifying the company, including the username responsible. CloudTrail is the correct service to use for auditing actions. CloudTrail tracks API calls made to AWS services, including S3 object access attempts. By enabling object-level logging in CloudTrail for the S3 bucket, everyGetObject, PutObject, and other operations are captured as events. These events contain valuable information, including the user identity (IAM user or role) making the request. These CloudTrail logs can then be forwarded to Amazon CloudWatch Logs. In CloudWatch Logs, you can define metric filters that search for specific patterns indicative of policy violations (e.g., unauthorized access attempts based on source IP, user identity, or attempted operation). When a metric filter detects a violation, it triggers a CloudWatch alarm. This alarm can be configured to send a notification to the appropriate team (e.g., security or compliance) via SNS, containing details of the violation, including the username extracted from the CloudTrail event data.

Why the other options are incorrect:

A. Use AWS Config rules: AWS Config is excellent for evaluating the configuration of AWS resources against desired configuration. However, Config rules primarily focus on configuration changes and compliance of resource settings themselves (e.g., is encryption enabled on the S3 bucket?). It is not designed to directly monitor API calls and access attempts to individual objects to identify policy violations in the way CloudTrail can. While you could potentially build complex rules to detect certain access patterns, it's not the intended use case or efficient for capturing user identity and real-time access attempts.

B. Use Amazon CloudWatch metrics: CloudWatch metrics are primarily for performance monitoring (CPU utilization, network traffic, etc.). While S3 exposes some metrics (bucket size, number of objects), it doesn't provide granular object-level metrics that directly indicate policy violations or capture the user identity attempting to access the objects. You can't derive information about individual access attempts or user identities from standard S3 CloudWatch metrics.

D. Use Amazon S3 server access logs: S3 server access logs are a viable way to record access attempts to objects in the bucket. You can configure S3 to write these logs to another bucket. You can then ingest the logs from this bucket to CloudWatch Logs, which allows metric filters to be applied for the logs. This option is viable and provides similar results to Option C. However, CloudTrail provides more comprehensive audit logging because it captures all API calls made to AWS services.

Authoritative Links:

AWS CloudTrail: https://aws.amazon.com/cloudtrail/
Amazon CloudWatch: https://aws.amazon.com/cloudwatch/
Amazon S3 Server Access Logging: https://docs.aws.amazon.com/AmazonS3/latest/userguide/logging-with-S3.html
AWS Config: https://aws.amazon.com/config/

Question: 194 CertyIQ

A company needs to load customer data that comes from a third party into an Amazon Redshift data warehouse. The company stores order data and product data in the same data warehouse. The company wants to use the combined dataset to identify potential new customers.

A data engineer notices that one of the fields in the source data includes values that are in JSON format.

How should the data engineer load the JSON data into the data warehouse with the LEAST effort?

A.Use the SUPER data type to store the data in the Amazon Redshift table.
B.Use AWS Glue to flatten the JSON data and ingest it into the Amazon Redshift table.
C.Use Amazon S3 to store the JSON data. Use Amazon Athena to query the data.
D.Use an AWS Lambda function to flatten the JSON data. Store the data in Amazon S3.

Answer: A

Explanation:

The best approach for loading JSON data with minimal effort into Amazon Redshift when you already have an existing Redshift data warehouse for other data is to leverage Redshift's SUPER data type.

Here's why:

Direct Integration: The SUPER data type is designed specifically for storing semi-structured data like JSON directly within Redshift tables. This eliminates the need for extensive transformations before loading.

Simplified Loading: You can load JSON data into a SUPER column using the standard COPY command, making the process straightforward.

Querying Flexibility: Redshift offers SQL extensions to query nested elements within the SUPER column, allowing you to access and analyze the JSON data directly within your existing Redshift environment. You can extract specific attributes and perform joins with other tables.

Reduced Complexity: Option B (AWS Glue) requires creating, configuring, and maintaining an ETL (Extract, Transform, Load) job, adding complexity and overhead. Similarly, option D (Lambda) requires writing and managing custom code, which adds development and operational burden.

Avoidance of a New Query Engine: Option C (S3 and Athena) introduces a separate query engine and data storage location, complicating the data pipeline and potentially increasing latency when joining with data in Redshift. This also means having to manage separate security configurations, catalogs, and access controls.

Using the SUPER data type allows you to keep your data within the Redshift data warehouse, allowing easy combination of the third-party JSON customer data with your existing order and product data, simplifying analysis for identifying potential new customers. This achieves the company's objectives with minimal effort. Flattening the JSON using Glue or Lambda is unnecessary complexity when Redshift can handle the JSON format natively.

Relevant Links:

Amazon Redshift SUPER Data Type: https://docs.aws.amazon.com/redshift/latest/dg/super-data-type.html
Loading JSON Data in Amazon Redshift: https://aws.amazon.com/blogs/big-data/query-semi-structured-json-data-natively-in-amazon-redshift/

Question: 195 CertyIQ

A company wants to analyze sales records that the company stores in a MySQL database. The company wants to correlate the records with sales opportunities identified by Salesforce.

The company receives 2 GB of sales records every day. The company has 100 GB of identified sales opportunities. A data engineer needs to develop a process that will analyze and correlate sales records and sales opportunities. The process must run once each night.

Which solution will meet these requirements with the LEAST operational overhead?

A.Use Amazon Managed Workflows for Apache Airflow (Amazon MWAA) to fetch both datasets. Use AWS Lambda functions to correlate the datasets. Use AWS Step Functions to orchestrate the process.

B.Use Amazon AppFlow to fetch sales opportunities from Salesforce. Use AWS Glue to fetch sales records from the MySQL database. Correlate the sales records with the sales opportunities. Use Amazon Managed Workflows for Apache Airflow (Amazon MWAA) to orchestrate the process.

C.Use Amazon AppFlow to fetch sales opportunities from Salesforce. Use AWS Glue to fetch sales records from the MySQL database. Correlate the sales records with sales opportunities. Use AWS Step Functions to orchestrate the process.

D.Use Amazon AppFlow to fetch sales opportunities from Salesforce. Use Amazon Kinesis Data Streams to fetch sales records from the MySQL database. Use Amazon Managed Service for Apache Flink to correlate the datasets. Use AWS Step Functions to orchestrate the process.

Answer: C

Explanation:

Here's a detailed justification for why option C is the most suitable solution, along with supporting concepts and links:

Justification for Option C (Correct):

Option C provides a balanced approach, combining specialized tools for data ingestion and transformation

with a robust orchestration service. It uses Amazon AppFlow to efficiently extract sales opportunities from Salesforce, a common use case for this service, which offers pre-built connectors and managed data transfer. AWS Glue is used to pull data from the MySQL database and transform it, making it suitable for batch processing of the 2GB daily sales records. AWS Step Functions then orchestrates the overall process. Since the process needs to run once each night, Step Functions' state management and built-in retry mechanisms provide reliability without requiring custom coding. Option C also boasts the least operational overhead by using fully managed services.

Why other options are less suitable:

Option A: Lambda functions are generally not ideal for processing larger datasets like 100GB of sales opportunities. They have execution time and memory limitations, making this option less efficient and potentially error-prone.
Option B: While using Amazon AppFlow and Glue is appropriate, utilizing Amazon MWAA for orchestration introduces unnecessary complexity and overhead for a daily batch process. MWAA is more suitable for complex workflows.
Option D: Kinesis Data Streams are designed for real-time data ingestion, which is not necessary for a nightly batch process. Flink is optimized for continuous stream processing, making this solution an overkill for analyzing data once per night.

Key Concepts and Considerations:

Amazon AppFlow: A fully managed integration service that enables you to securely transfer data between SaaS applications and AWS services. https://aws.amazon.com/appflow/
AWS Glue: A fully managed ETL (extract, transform, and load) service that simplifies the process of preparing and loading data for analytics. https://aws.amazon.com/glue/
AWS Step Functions: A serverless orchestration service that lets you coordinate multiple AWS services into serverless workflows. https://aws.amazon.com/step-functions/
Operational Overhead: The effort and cost associated with managing and maintaining a system. Using fully managed services minimizes this overhead.

In summary, Option C provides a well-balanced and cost-effective solution with minimal operational overhead by leveraging AppFlow, Glue, and Step Functions for data ingestion, transformation, and orchestration respectively.

Question: 196 CertyIQ

A company stores server logs in an Amazon S3 bucket. The company needs to keep the logs for 1 year. The logs are not required after 1 year.

A data engineer needs a solution to automatically delete logs that are older than 1 year.

Which solution will meet these requirements with the LEAST operational overhead?

A.Define an S3 Lifecycle configuration to delete the logs after 1 year.
B.Create an AWS Lambda function to delete the logs after 1 year.
C.Schedule a cron job on an Amazon EC2 instance to delete the logs after 1 year.
D.Configure an AWS Step Functions state machine to delete the logs after 1 year.

Answer: A

Explanation:

The correct answer is **A. Define an S3 Lifecycle configuration to delete the logs after 1 year.**

Here's a detailed justification:

S3 Lifecycle policies are designed explicitly for managing the lifecycle of objects within S3 buckets, automating actions like transitioning objects to different storage classes or deleting them based on age. These policies operate directly within the S3 service, requiring minimal configuration and maintenance. The problem statement emphasizes "least operational overhead," and S3 Lifecycle policies perfectly align with this requirement due to their built-in nature and automatic execution. The data engineer can easily define a rule to permanently delete objects older than 365 days (1 year), which addresses the specific need of removing old server logs.

Options B, C, and D are less efficient and have higher operational overhead. While AWS Lambda (B) can be used for object deletion, it requires creating, deploying, and managing a Lambda function, along with setting up event triggers or schedules. Cron jobs on EC2 instances (C) similarly require managing an EC2 instance, configuring the cron job, and ensuring the instance's availability and security. AWS Step Functions (D) involves designing and maintaining a state machine, which is overkill for a simple deletion task. All these options introduce additional complexity and potential points of failure compared to the native S3 Lifecycle solution.

Therefore, S3 Lifecycle configuration provides the simplest, most cost-effective, and least operationally intensive way to automatically delete logs older than 1 year. It leverages a built-in S3 feature, reducing the need for custom code or infrastructure management.

Further research:

S3 Lifecycle Management: https://docs.aws.amazon.com/AmazonS3/latest/userguide/lifecycle-configuration-concept.html

Question: 197 CertyIQ

A company is designing a serverless data processing workflow in AWS Step Functions that involves multiple steps. The processing workflow ingests data from an external API, transforms the data by using multiple AWS Lambda functions, and loads the transformed data into Amazon DynamoDB.

The company needs the workflow to perform specific steps based on the content of the incoming data.

Which Step Functions state type should the company use to meet this requirement?

A.Parallel
B.Choice
C.Task
D.Map

Answer: B

Explanation:

The correct answer is **B. Choice**.

The core requirement is to perform specific steps based on the content of the incoming data. This signifies a need for conditional branching within the Step Functions workflow. The Choice state in AWS Step Functions allows you to add branching logic to your state machine. It evaluates a set of rules based on the input data (or the result of a previous state) and transitions to a different state based on which rule matches.

In this scenario, the incoming data from the external API can be analyzed by the Choice state. Based on specific criteria (e.g., a particular field having a certain value, or the data satisfying a particular condition), the workflow will proceed down different paths, invoking different Lambda functions or performing different

DynamoDB operations.

A. Parallel: A Parallel state executes multiple branches of your workflow in parallel. While useful for speeding up processing, it doesn't provide conditional branching based on data content.

C. Task: A Task state represents a single unit of work, such as invoking a Lambda function or calling an AWS service API. It performs a single action but doesn't provide conditional branching.

D. Map: A Map state is used to iterate over a collection of items in an array. While it allows for processing multiple data items, it doesn't facilitate branching based on the content of each item.

Therefore, the Choice state is the most appropriate choice for implementing data-driven conditional logic within the Step Functions workflow to transform and load data into DynamoDB based on the content of the incoming data.

Further research:

AWS Step Functions Choice State
AWS Step Functions State Types

Question: 198 CertyIQ

A data engineer created a table named cloudtrail_logs in Amazon Athena to query AWS CloudTrail logs and prepare data for audits. The data engineer needs to write a query to display errors with error codes that have occurred since the beginning of 2024. The query must return the 10 most recent errors.

Which query will meet these requirements?

A.select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logswhere errorcode is not nulland eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessageorder by TotalEvents desclimit 10;

B.select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logs where eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessage order by TotalEvents desc limit 10;

C.select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logswhere eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessageorder by eventname asc limit 10;

D.select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logs where errorcode is not nulland eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessagelimit 10;

Answer: A

Explanation:

The correct answer is A. Here's a detailed justification:

The requirement is to identify the 10 most recent errors with error codes since the beginning of 2024 from AWS CloudTrail logs stored in an Athena table.

Option A select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logswhere errorcode is not nulland eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessageorder by TotalEvents desclimit 10; correctly addresses all parts of the requirement:

where errorcode is not null: This ensures only errors are included in the result. CloudTrail logs contain many non-error events; this filter focuses on the relevant ones.
eventtime >= '2024-01-01T00:00:00Z': This restricts the results to events that occurred since the beginning of 2024.
group by eventname, errorcode, errormessage: This groups the logs based on the event name, error code, and

error message to count the occurrences of each distinct error.
order by TotalEvents desc: This sorts the results in descending order of the total number of occurrences for each error combination, placing the most frequent errors at the top.
limit 10: This restricts the output to the top 10 most frequent errors. TotalEvents is implicitly providing the number of occurances for each event, which is then ordered to extract the most recent errors that have eventtime as of January 1st, 2024.

Option B select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logs where eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessage order by TotalEvents desc limit 10; is incorrect because it doesn't filter for errors specifically (missing errorcode is not null). It includes all events, not just errors with error codes.

Option C select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logswhere eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessageorder by eventname asc limit 10; is incorrect because it sorts by event name in ascending order (order by eventname asc), not by the frequency of the errors. It also doesn't filter for error events only, failing to include the constraint where errorcode is not null.

Option D select count (*) as TotalEvents, eventname, errorcode, errormessage from cloudtrail_logs where errorcode is not nulland eventtime >= '2024-01-01T00:00:00Z' group by eventname, errorcode, errormessagelimit 10; is incorrect as it retrieves 10 arbitrary rows that satisfy the WHERE clause since the order is not specified. The query does not present the 10 most frequent errors.

In summary, option A correctly filters for errors, limits the timeframe, groups the results to count errors, orders by the total number of occurrences to find the most frequent, and limits the output to the top 10.

Relevant links:

AWS CloudTrail
Amazon Athena
SQL GROUP BY
SQL ORDER BY
SQL WHERE Clause

Question: 199 CertyIQ

An online retailer uses multiple delivery partners to deliver products to customers. The delivery partners send order summaries to the retailer. The retailer stores the order summaries in Amazon S3.

Some of the order summaries contain personally identifiable information (PII) about customers. A data engineer needs to detect PII in the order summaries so the company can redact the PII.

Which solution will meet these requirements with the LEAST operational overhead?

A.Amazon Textract
B.Amazon S3 Storage Lens
C.Amazon Macie
D.Amazon SageMaker Data Wrangler

Answer: C

Explanation:

The correct answer is C, Amazon Macie. Here's why:

Amazon Macie's Purpose: Macie is designed specifically for discovering, classifying, and protecting sensitive data, including PII, stored in Amazon S3.

PII Detection: Macie uses machine learning and pattern matching to automatically identify PII in S3 objects, aligning directly with the requirement to detect PII in order summaries.

Low Operational Overhead: Macie is a fully managed service, meaning the data engineer doesn't need to provision or manage infrastructure. Configuration primarily involves pointing Macie at the S3 bucket.

Alternatives Explanation:

Amazon Textract: Primarily focuses on extracting text from scanned documents and images, not designed for general PII detection across S3 buckets.
Amazon S3 Storage Lens: Provides visibility into storage usage and activity trends but doesn't offer PII detection capabilities.
Amazon SageMaker Data Wrangler: While it can be used for data preparation and feature engineering, setting it up for PII detection would require significant custom coding and model training, leading to higher operational overhead compared to Macie. It's not a direct PII detection service.

Why Macie is Best: Macie provides the most direct, automated, and low-overhead solution for detecting PII in S3, aligning with the problem's need for ease of use and minimal management. It's built for this purpose, unlike the other options, which would require significant customization or aren't applicable.

Relevant Links:

Amazon Macie: https://aws.amazon.com/macie/

Question: 200 CertyIQ

A company has an Amazon Redshift data warehouse that users access by using a variety of IAM roles. More than 100 users access the data warehouse every day.

The company wants to control user access to the objects based on each user's job role, permissions, and how sensitive the data is.

Which solution will meet these requirements?

A.Use the role-based access control (RBAC) feature of Amazon Redshift.
B.Use the row-level security (RLS) feature of Amazon Redshift.
C.Use the column-level security (CLS) feature of Amazon Redshift.
D.Use dynamic data masking policies in Amazon Redshift.

Answer: A

Explanation:

The correct answer is A: Use the role-based access control (RBAC) feature of Amazon Redshift.

RBAC in Amazon Redshift directly addresses the requirement of controlling user access based on job roles and permissions. RBAC allows administrators to create roles representing different job functions within the organization (e.g., data analyst, data scientist, marketing analyst). These roles are then granted specific privileges on database objects (tables, views, etc.) according to the tasks and data sensitivity relevant to each role. Users are then assigned to these roles, inheriting the corresponding access rights. This approach simplifies access management because permissions are managed at the role level, not individually for each user.

Options B, C, and D, while related to data security, do not directly address the central requirement of controlling access based on user roles and permissions in a scalable manner. Row-level security (RLS) limits access to specific rows based on user characteristics or criteria within the data itself, not the user's role. Column-level security (CLS) restricts access to specific columns, but it doesn't inherently connect access to user roles. Dynamic data masking policies mask sensitive data based on various criteria, but again, they don't tie access directly to pre-defined roles.

RBAC provides a structured and scalable approach to managing user access in Amazon Redshift, aligning user permissions with job roles and data sensitivity. This simplifies administration, improves security, and ensures compliance with data governance policies, especially in environments with a large number of users and roles.Implementing RBAC is more efficient for managing access for a large number of users (100+). With RLS, CLS, or data masking, you'd likely have to create more complex and numerous rules to approximate the functionality that RBAC provides natively.

For further research, you can refer to the AWS documentation on Amazon Redshift security:

Amazon Redshift Security Overview: https://docs.aws.amazon.com/redshift/latest/dg/c_security-overview.html
Role-Based Access Control (RBAC) in Amazon Redshift: https://docs.aws.amazon.com/redshift/latest/dg/security-iam-managing-rbac.html
Row-Level Security (RLS) in Amazon Redshift: https://docs.aws.amazon.com/redshift/latest/dg/row-level-security.html
Column-Level Security in Amazon Redshift: https://docs.aws.amazon.com/redshift/latest/dg/column-level-security.html
Dynamic Data Masking in Amazon Redshift: https://docs.aws.amazon.com/redshift/latest/dg/dynamic-data-masking.html

Question: 201 CertyIQ

A company uses Amazon DataZone as a data governance and business catalog solution. The company stores data in an Amazon S3 data lake. The company uses AWS Glue with an AWS Glue Data Catalog.

A data engineer needs to publish AWS Glue Data Quality scores to the Amazon DataZone portal.

Which solution will meet this requirement?

A.Create a data quality ruleset with Data Quality Definition language (DQDL) rules that apply to a specific AWS Glue table. Schedule the ruleset to run daily. Configure the Amazon DataZone project to have an Amazon Redshift data source. Enable the data quality configuration for the data source.

B.Configure AWS Glue ETL jobs to use an Evaluate Data Quality transform. Define a data quality ruleset inside the jobs. Configure the Amazon DataZone project to have an AWS Glue data source. Enable the data quality configuration for the data source.

C.Create a data quality ruleset with Data Quality Definition language (DQDL) rules that apply to a specific AWS Glue table. Schedule the ruleset to run daily. Configure the Amazon DataZone project to have an AWS Glue data source. Enable the data quality configuration for the data source.

D.Configure AWS Glue ETL jobs to use an Evaluate Data Quality transform. Define a data quality ruleset inside the jobs. Configure the Amazon DataZone project to have an Amazon Redshift data source. Enable the data quality configuration for the data source.

Answer: C

Explanation:

The correct solution is **C**. Here's a detailed justification:

Amazon DataZone aims to provide a centralized catalog for data assets along with their quality scores. To

publish AWS Glue Data Quality scores to DataZone, a specific integration path is needed. AWS Glue provides native Data Quality features, including Data Quality Rulesets defined using Data Quality Definition Language (DQDL).

Option C leverages these native AWS Glue features optimally. It proposes creating DQDL rulesets targeted at the specific AWS Glue tables that reside in your Data Lake. Scheduling these rulesets daily ensures continuous monitoring and score updates. The core of the integration lies in configuring an Amazon DataZone project with an AWS Glue Data Source. This configuration explicitly tells DataZone where the data assets and their associated metadata (including the Data Quality scores) reside. Enabling the data quality configuration for the Data Source instructs DataZone to actively pull and display the Glue Data Quality metrics within the DataZone portal.

Option A is incorrect because it suggests configuring Amazon Redshift as a data source, which is not correct because the data resides in the S3 data lake and is cataloged in Glue Data Catalog. Redshift would only be relevant if the data was being loaded into Redshift.

Options B and D suggest incorporating data quality rules within ETL jobs using the Evaluate Data Quality transform. While technically possible, this approach is less optimal for publishing to DataZone because it tightly couples data quality checks with data transformation. Furthermore, DataZone offers a more straightforward integration via a Data Source configuration targeted at AWS Glue. Also, using Redshift in DataZone is not applicable here.

Therefore, Option C is the most direct and recommended approach to achieve the specified goal by directly leveraging AWS Glue Data Quality with Amazon DataZone.

Supporting resources:

AWS Glue Data Quality: https://docs.aws.amazon.com/glue/latest/dg/glue_data_quality.html
Amazon DataZone Integration: (Although specific DataZone integration documentation is evolving rapidly, search for "Amazon DataZone data sources" and "AWS Glue integration with Amazon DataZone" in the AWS documentation to find the most up-to-date information.)

Question: 202 CertyIQ

A company has a data warehouse in Amazon Redshift. To comply with security regulations, the company needs to log and store all user activities and connection activities for the data warehouse.

Which solution will meet these requirements?

A.Create an Amazon S3 bucket. Enable logging for the Amazon Redshift cluster. Specify the S3 bucket in the logging configuration to store the logs.

B.Create an Amazon Elastic File System (Amazon EFS) file system. Enable logging for the Amazon Redshift cluster. Write logs to the EFS file system.

C.Create an Amazon Aurora MySQL database. Enable logging for the Amazon Redshift cluster. Write the logs to a table in the Aurora MySQL database.

D.Create an Amazon Elastic Block Store (Amazon EBS) volume. Enable logging for the Amazon Redshift cluster. Write the logs to the EBS volume.

Answer: A

Explanation:

The correct answer is A because Amazon Redshift natively supports logging user and connection activities directly to an Amazon S3 bucket. This is the recommended and most straightforward approach for meeting the stated requirements.

Here's a detailed justification:

Amazon Redshift Logging: Redshift provides built-in functionality to log connection attempts, SQL queries, and other user activities. These logs are crucial for auditing, security analysis, and compliance. https://docs.aws.amazon.com/redshift/latest/dg/r_CONFIG_AUDIT_LOGGING.html

Amazon S3 as a Log Destination: Amazon S3 is a scalable, durable, and cost-effective object storage service ideal for storing log files. Redshift can be configured to automatically deliver its logs to a specified S3 bucket. This ensures that the logs are securely stored and readily available for analysis.

Why other options are incorrect:

Option B (Amazon EFS): EFS is a file system service primarily designed for shared file storage by EC2 instances. While technically possible to write logs to EFS, it's not the intended use case for Redshift logging and would be less efficient and more complex than using S3.

Option C (Amazon Aurora MySQL): Aurora MySQL is a relational database service. While you could potentially write log data to a database table, it adds unnecessary complexity and overhead compared to using S3. Storing logs in a database generally involves more processing and storage costs.

Option D (Amazon EBS): EBS volumes are block storage devices primarily used for EC2 instances. They are not suitable for direct log storage from Redshift. Furthermore, EBS volumes are tied to a specific availability zone, which could create availability issues.

In summary, configuring Redshift logging to an S3 bucket offers a simple, efficient, and scalable solution for storing user and connection activities, directly addressing the security and compliance requirements. S3's durability, cost-effectiveness, and integration with other AWS services make it the most suitable option for this scenario.

Question: 203 CertyIQ

A company wants to migrate a data warehouse from Teradata to Amazon Redshift.

Which solution will meet this requirement with the LEAST operational effort?

A.Use AWS Database Migration Service (AWS DMS) Schema Conversion to migrate the schema. Use AWS DMS to migrate the data.

B.Use the AWS Schema Conversion Tool (AWS SCT) to migrate the schema. Use AWS Database Migration Service (AWS DMS) to migrate the data.

C.Use AWS Database Migration Service (AWS DMS) to migrate the data. Use automatic schema conversion.

D.Manually export the schema definition from Teradata. Apply the schema to the Amazon Redshift database. Use AWS Database Migration Service (AWS DMS) to migrate the data.

Answer: B

Explanation:

The optimal solution for migrating a data warehouse from Teradata to Amazon Redshift with the least operational effort involves using the AWS Schema Conversion Tool (AWS SCT) for schema migration and AWS Database Migration Service (AWS DMS) for data migration.

AWS SCT automates the process of converting the Teradata schema to a Redshift-compatible schema. It assesses the source database schema, identifies potential conversion challenges, and provides recommendations on how to address them. This drastically reduces manual effort compared to manually exporting and applying the schema definition. AWS SCT handles a significant portion of the complexities

involved in translating database objects, data types, and stored procedures, minimizing manual intervention.

AWS DMS is a managed service specifically designed for database migration. It efficiently and securely transfers data from Teradata to Amazon Redshift. DMS supports heterogeneous database migrations, meaning it can move data between different database engines. The service handles data type conversions and ensures data integrity during the transfer process. Using DMS automates data transfer tasks, minimizing the manual effort and potential errors associated with manual data extraction and loading. DMS allows for continuous data replication, enabling minimal downtime during the migration.

Option A is incorrect because AWS DMS Schema Conversion is primarily designed for migrating schemas between similar database engines. Option C is not ideal because automatic schema conversion in DMS might not be as comprehensive and customized as using AWS SCT, potentially requiring manual intervention. Option D requires the most manual effort as manually exporting and applying schemas is prone to error and time-consuming.

Therefore, AWS SCT and AWS DMS together provide a managed and automated solution that minimizes operational overhead and allows for a smooth migration from Teradata to Amazon Redshift.

Authoritative Links:

AWS Schema Conversion Tool (SCT): https://aws.amazon.com/dms/schema-conversion-tool/
AWS Database Migration Service (DMS): https://aws.amazon.com/dms/

Question: 204 CertyIQ

A company uses a variety of AWS and third-party data stores. The company wants to consolidate all the data into a central data warehouse to perform analytics. Users need fast response times for analytics queries.

The company uses Amazon QuickSight in direct query mode to visualize the data. Users normally run queries during a few hours each day with unpredictable spikes.

Which solution will meet these requirements with the LEAST operational overhead?

A. Use Amazon Redshift Serverless to load all the data into Amazon Redshift managed storage (RMS).
B. Use Amazon Athena to load all the data into Amazon S3 in Apache Parquet format.
C. Use Amazon Redshift provisioned clusters to load all the data into Amazon Redshift managed storage (RMS).
D. Use Amazon Aurora PostgreSQL to load all the data into Aurora.

Answer: A

Explanation:

Here's a detailed justification for why option A, using Amazon Redshift Serverless, is the best solution for the given scenario:

The core requirement is to consolidate data from various sources into a central data warehouse for analytics with fast query response times, especially during peak hours. Amazon QuickSight is used in direct query mode, indicating that QuickSight queries the data warehouse directly rather than importing data. The workload involves unpredictable spikes, making on-demand scaling crucial.

A. Use Amazon Redshift Serverless to load all the data into Amazon Redshift managed storage (RMS).

Scalability: Redshift Serverless automatically scales compute resources based on workload demand. This directly addresses the requirement for handling unpredictable spikes without manual intervention.
Performance: Redshift is a columnar data warehouse designed for fast analytics queries. Its architecture is optimized for large-scale data processing and complex queries.

Direct Query Compatibility: Redshift seamlessly integrates with Amazon QuickSight in direct query mode.
Minimal Operational Overhead: Serverless eliminates the need for capacity planning, cluster sizing, and manual scaling. AWS manages the underlying infrastructure, minimizing operational burden. You only pay for what you use.
Redshift Managed Storage (RMS): This provides a cost-effective storage option specifically designed for Redshift, offering optimized performance.

Why other options are less suitable:

B. Use Amazon Athena to load all the data into Amazon S3 in Apache Parquet format.

While Athena is serverless and cost-effective for querying data in S3, it's generally slower than Redshift for complex analytics queries, especially with unpredictable spikes. Direct querying from QuickSight to S3 using Athena might not consistently meet the fast response time requirement, especially as data grows.

C. Use Amazon Redshift provisioned clusters to load all the data into Amazon Redshift managed storage (RMS).

Redshift provisioned clusters require capacity planning and manual scaling. While this provides fine-grained control, it introduces significant operational overhead, especially for unpredictable workloads. Scaling up or down involves downtime, which could impact query performance during peak times.

D. Use Amazon Aurora PostgreSQL to load all the data into Aurora.

Aurora PostgreSQL is a relational database, not a data warehouse. While it can handle analytics queries, it is not optimized for the scale and complexity of queries typically performed in a data warehouse setting. It also does not scale as effectively or efficiently as Redshift for analytics workloads with unpredictable spikes.

Conclusion:

Amazon Redshift Serverless provides the best balance of performance, scalability, and minimal operational overhead for the given scenario. It allows the company to consolidate data, achieve fast query response times, handle unpredictable spikes, and leverage QuickSight's direct query capabilities without the complexities of managing a provisioned cluster.

Authoritative Links:

Amazon Redshift Serverless: https://aws.amazon.com/redshift/serverless/
Amazon QuickSight: https://aws.amazon.com/quicksight/
Amazon Athena: https://aws.amazon.com/athena/
Amazon Aurora: https://aws.amazon.com/rds/aurora/

Question: 205 CertyIQ

A data engineer uses Amazon Kinesis Data Streams to ingest and process records that contain user behavior data from an application every day.

The data engineer notices that the data stream is experiencing throttling because hot shards receive much more data than other shards in the data stream.

How should the data engineer resolve the throttling issue?

A. Use a random partition key to distribute the ingested records.
B. Increase the number of shards in the data stream. Distribute the records across the shards.
C. Limit the number of records that are sent each second by the producer to match the capacity of the stream.
D. Decrease the size of the records that the producer sends to match the capacity of the stream.

Answer: A

Question: 206 CertyIQ

A company has a data processing pipeline that includes several dozen steps. The data processing pipeline needs to send alerts in real time when a step fails or succeeds. The data processing pipeline uses a combination of Amazon S3 buckets, AWS Lambda functions, and AWS Step Functions state machines.

A data engineer needs to create a solution to monitor the entire pipeline.

Which solution will meet these requirements?

A.Configure the Step Functions state machines to store notifications in an Amazon S3 bucket when the state machines finish running. Enable S3 event notifications on the S3 bucket.

B.Configure the AWS Lambda functions to store notifications in an Amazon S3 bucket when the state machines finish running. Enable S3 event notifications on the S3 bucket.

C.Use AWS CloudTrail to send a message to an Amazon Simple Notification Service (Amazon SNS) topic that sends notifications when a state machine fails to run or succeeds to run.

D.Configure an Amazon EventBridge rule to react when the execution status of a state machine changes. Configure the rule to send a message to an Amazon Simple Notification Service (Amazon SNS) topic that sends notifications.

Answer: D

Question: 207 CertyIQ

A company has an application that uses an Amazon API Gateway REST API and an AWS Lambda function to retrieve data from an Amazon DynamoDB instance. Users recently reported intermittent high latency in the application's response times. A data engineer finds that the Lambda function experiences frequent throttling when the company's other Lambda functions experience increased invocations.

The company wants to ensure the API's Lambda function operate without being affected by other Lambda functions.

Which solution will meet this requirement MOST cost-effectively?

A.Increase the number of read capacity unit (RCU) in DynamoDB.

B.Configure provisioned concurrency for the Lambda function.

C.Configure reserved concurrency for the Lambda function.

D.Increase the Lambda function timeout and allocated memory.

Answer: C

Thank you

Thank you for being so interested in the premium exam material.
I'm glad to hear that you found it informative and helpful.

If you have any feedback or thoughts on the bumps, I would love to hear them.
Your insights can help me improve our writing and better understand our readers.

Best of Luck

You have worked hard to get to this point, and you are well-prepared for the exam
Keep your head up, stay positive, and go show that exam what you're made of!

Feedback

More Papers