Amazon

(AWS Certified Data Engineer - Associate DEA-C01)

AWS Certified Data Engineer - Associate DEA-C01

Total: **241 Questions**

Link:

Question: 1

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 Conceptshttps://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-sort-dist.html

Question: 68

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

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

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

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

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

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.