Amazon

(AWS Certified Data Analytics - Specialty)

AWS Certified Data Analytics - Specialty (DAS-C01)

Total: **164 Questions**

Link:

Question: 1

A financial services company needs to aggregate daily stock trade data from the exchanges into a data store. The company requires that data be streamed directly into the data store, but also occasionally allows data to be modified using SQL. The solution should integrate complex, analytic queries running with minimal latency. The solution must provide a business intelligence dashboard that enables viewing of the top contributors to anomalies in stock prices.
Which solution meets the company's requirements?

A.Use Amazon Kinesis Data Firehose to stream data to Amazon S3. Use Amazon Athena as a data source for Amazon QuickSight to create a business intelligence dashboard.

B.Use Amazon Kinesis Data Streams to stream data to Amazon Redshift. Use Amazon Redshift as a data source for Amazon QuickSight to create a business intelligence dashboard.

C.Use Amazon Kinesis Data Firehose to stream data to Amazon Redshift. Use Amazon Redshift as a data source for Amazon QuickSight to create a business intelligence dashboard.

D.Use Amazon Kinesis Data Streams to stream data to Amazon S3. Use Amazon Athena as a data source for Amazon QuickSight to create a business intelligence dashboard.

Answer: C

Explanation:

The correct answer is C. Here's why:

The financial services company has specific requirements: streaming data ingestion, SQL-based modification, low-latency analytics, and a business intelligence dashboard.

Streaming Data Ingestion: Kinesis Data Firehose is designed for streaming data into data stores. Kinesis Data Streams provides more granular control but requires more management overhead.

SQL-Based Modification: Amazon Redshift is a fully managed, petabyte-scale data warehouse service that supports SQL. This allows data modifications via SQL queries. Athena supports SQL but is designed for querying data in S3, not modifying it.

Low-Latency Analytics: Redshift is optimized for complex analytic queries and offers low latency performance. S3 with Athena has higher latency, especially for complex queries.

Business Intelligence Dashboard: Amazon QuickSight integrates well with Redshift, allowing you to create interactive dashboards for visualizing anomalies in stock prices.

Therefore, using Kinesis Data Firehose to stream data into Redshift enables efficient streaming and SQL-based modification. Subsequently, Redshift as a data source for QuickSight meets the dashboard requirements.

Options A and D utilize S3 and Athena, which are not suitable for scenarios where low-latency analytics and SQL-based data modification are required. Option B uses Kinesis Data Streams, necessitating more complex implementation. Option C aligns with the demands due to its simplicity and efficiency.

Supporting Links:

Amazon Kinesis Data Firehose:https://aws.amazon.com/kinesis/data-firehose/
Amazon Redshift:https://aws.amazon.com/redshift/
Amazon QuickSight:https://aws.amazon.com/quicksight/

Question: 2

A financial company hosts a data lake in Amazon S3 and a data warehouse on an Amazon Redshift cluster. The

company uses Amazon QuickSight to build dashboards and wants to secure access from its on-premises Active Directory to Amazon QuickSight.
How should the data be secured?

A.Use an Active Directory connector and single sign-on (SSO) in a corporate network environment.

B.Use a VPC endpoint to connect to Amazon S3 from Amazon QuickSight and an IAM role to authenticate Amazon Redshift.

C.Establish a secure connection by creating an S3 endpoint to connect Amazon QuickSight and a VPC endpoint to connect to Amazon Redshift.

D.Place Amazon QuickSight and Amazon Redshift in the security group and use an Amazon S3 endpoint to connect Amazon QuickSight to Amazon S3.

Answer: A

Explanation:

The correct answer is A. Here's why:

Single Sign-On (SSO) and Active Directory Integration: The primary requirement is to secure access to QuickSight using the existing on-premises Active Directory. The most direct and effective way to achieve this is through SSO. AWS supports integrating with on-premises Active Directory using services like AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) or using an AD Connector. This enables users to authenticate with their existing Active Directory credentials to access QuickSight.

Why other options are less suitable:

B: VPC endpoints are relevant for accessing S3 and Redshift privately from within a VPC, but they don't directly address the Active Directory integration requirement for QuickSight user authentication. IAM roles are crucial for allowing QuickSight to access the data in S3 and Redshift after a user has been authenticated, but IAM roles don't handle user authentication against Active Directory.
C: Creating S3 and VPC endpoints establishes secure connections to S3 and Redshift, respectively. However, neither establishes SSO or integration with Active Directory to manage QuickSight user access. This option focuses on network security but misses the authentication requirement.
D: Placing QuickSight and Redshift in a security group helps control network traffic between them. Using an S3 endpoint allows private connectivity to S3. But like options B and C, this doesn't address the core requirement of integrating Active Directory for user authentication in QuickSight.

In summary: Only option A directly addresses the need to integrate with on-premises Active Directory by utilizing SSO. SSO authentication will allow existing on-premise users to use their existing credentials.

Authoritative Links:

AWS Directory Service:https://aws.amazon.com/directoryservice/
AWS Managed Microsoft AD:https://aws.amazon.com/directoryservice/managed-microsoft-ad/ **AD Connector:**https://docs.aws.amazon.com/directoryservice/latest/admin-guide/ad_connector.html **Amazon QuickSight Security:**https://docs.aws.amazon.com/quicksight/latest/user/security-in-quicksight.html

Question: 3

A real estate company has a mission-critical application using Apache HBase in Amazon EMR. Amazon EMR is configured with a single master node. The company has over 5 TB of data stored on an Hadoop Distributed File System (HDFS). The company wants a cost-effective solution to make its HBase data highly available. Which architectural pattern meets company's requirements?

A.Use Spot Instances for core and task nodes and a Reserved Instance for the EMR master node. Configure the

EMR cluster with multiple master nodes. Schedule automated snapshots using Amazon EventBridge.

B.Store the data on an EMR File System (EMRFS) instead of HDFS. Enable EMRFS consistent view. Create an EMR HBase cluster with multiple master nodes. Point the HBase root directory to an Amazon S3 bucket.

C.Store the data on an EMR File System (EMRFS) instead of HDFS and enable EMRFS consistent view. Run two separate EMR clusters in two different Availability Zones. Point both clusters to the same HBase root directory in the same Amazon S3 bucket.

D.Store the data on an EMR File System (EMRFS) instead of HDFS and enable EMRFS consistent view. Create a primary EMR HBase cluster with multiple master nodes. Create a secondary EMR HBase read-replica cluster in a separate Availability Zone. Point both clusters to the same HBase root directory in the same Amazon S3 bucket.

Answer: D

Explanation:

The correct answer is D because it addresses both the high availability requirement and the need for a cost-effective solution for the HBase data. Here's why:

High Availability: Having a primary EMR HBase cluster with multiple master nodes ensures that if one master node fails, the others can take over, preventing downtime.

Disaster Recovery and Redundancy: Creating a secondary read-replica cluster in a separate Availability Zone (AZ) provides disaster recovery. If the primary AZ experiences an outage, the secondary cluster can continue to serve read requests. This design ensures business continuity.

Cost-Effectiveness: While maintaining two clusters incurs some cost, storing data on EMRFS (backed by Amazon S3) and pointing both clusters to the same S3 bucket minimizes storage costs and simplifies data synchronization.

EMRFS Consistent View: Using EMRFS and enabling the consistent view guarantees that all clusters have access to the latest data, regardless of which cluster wrote it. This consistency is vital for maintaining data integrity.

HDFS Limitation: The original configuration with HDFS on a single master node is a single point of failure. Replacing HDFS with EMRFS eliminates this risk.

Why other options are less suitable:

Option A: While using Spot Instances for cost savings is good and having multiple master nodes improves availability, it doesn't address data durability or disaster recovery. Snapshotting doesn't provide real-time redundancy.

Option B: Enabling EMRFS and multiple masters is a good starting point, but it lacks a secondary cluster in a separate AZ for disaster recovery.

Option C: While this option has two clusters, it only points both to the same S3 bucket without creating read-replicas. So, they are both working on the same data, and if one cluster has a write failure, it can impact the other.

Supporting Links:

EMRFS Consistent View:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emrfs-consistency.html
High Availability in EMR:https://aws.amazon.com/blogs/big-data/high-availability-for-your-amazon-emr-clusters/
HBase on EMR:https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase.html

Question: 4

A software company hosts an application on AWS, and new features are released weekly. As part of the application testing process, a solution must be developed that analyzes logs from each Amazon EC2 instance to

ensure that the application is working as expected after each deployment. The collection and analysis solution should be highly available with the ability to display new information with minimal delays.
Which method should the company use to collect and analyze the logs?

A.Enable detailed monitoring on Amazon EC2, use Amazon CloudWatch agent to store logs in Amazon S3, and use Amazon Athena for fast, interactive log analytics.

B.Use the Amazon Kinesis Producer Library (KPL) agent on Amazon EC2 to collect and send data to Kinesis Data Streams to further push the data to Amazon OpenSearch Service (Amazon Elasticsearch Service) and visualize using Amazon QuickSight.

C.Use the Amazon Kinesis Producer Library (KPL) agent on Amazon EC2 to collect and send data to Kinesis Data Firehose to further push the data to Amazon OpenSearch Service (Amazon Elasticsearch Service) and OpenSearch Dashboards (Kibana).

D.Use Amazon CloudWatch subscriptions to get access to a real-time feed of logs and have the logs delivered to Amazon Kinesis Data Streams to further push the data to Amazon OpenSearch Service (Amazon Elasticsearch Service) and OpenSearch Dashboards (Kibana).

Answer: D

Explanation:

The correct answer is **D**, which uses CloudWatch subscriptions, Kinesis Data Streams, and OpenSearch Service for log collection and analysis. Here's why:

Real-time Analysis: The requirement of displaying new information with minimal delays points to a near real-time or real-time analysis requirement. Both Kinesis Data Streams and CloudWatch subscriptions support real-time data processing.

High Availability: Kinesis Data Streams offers high availability and scalability, crucial for a production application that releases features weekly. It can handle a high volume of incoming log data.

Log Aggregation and Delivery: CloudWatch subscriptions provide a mechanism to filter and send specific log events to Kinesis Data Streams. This is preferable to relying solely on the CloudWatch agent writing to S3 (option A) because it's more direct and supports real-time needs. Kinesis Data Firehose (options B & C) is better suited for batch or near-real-time ingestion into destinations like S3 or Redshift, but Data Streams is preferable for immediate processing.

Log Analysis and Visualization: Amazon OpenSearch Service, coupled with OpenSearch Dashboards (Kibana), is well-suited for analyzing and visualizing log data. It provides powerful search capabilities and dashboards for monitoring application performance and identifying issues after deployments.

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

A: Using Amazon CloudWatch agent to store logs in S3 and using Amazon Athena for interactive log analytics does not meet the near real-time requirements. Athena is better suited for batch-oriented queries on data stored in S3, not for live analysis of logs generated during deployments. While Athena is fast, it's not as responsive as a system that can continuously update dashboards as new logs arrive.

B & C: Both of these answers use Kinesis Data Firehose. While Kinesis Data Firehose can deliver data to OpenSearch Service, it's primarily designed for loading data into data lakes or analytics stores like S3 or Redshift. Kinesis Data Streams gives the flexibility needed to build a more complex processing pipeline, while keeping a real-time focus. Also, not utilizing the CloudWatch subscription means more complex management of log capture.

In summary, option D provides a comprehensive, highly available, and scalable solution that addresses the real-time log analysis and visualization requirements, making it the most suitable choice.

Supporting Links:

Amazon CloudWatch:https://aws.amazon.com/cloudwatch/
Amazon Kinesis Data Streams:https://aws.amazon.com/kinesis/data-streams/
Amazon OpenSearch Service:https://aws.amazon.com/opensearch-service/

CloudWatch Subscriptions:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html

Question: 5

A data analyst is using AWS Glue to organize, cleanse, validate, and format a 200 GB dataset. The data analyst triggered the job to run with the Standard worker type. After 3 hours, the AWS Glue job status is still RUNNING.
Logs from the job run show no error codes. The data analyst wants to improve the job execution time without overprovisioning.
Which actions should the data analyst take?

A.Enable job bookmarks in AWS Glue to estimate the number of data processing units (DPUs). Based on the profiled metrics, increase the value of the executor- cores job parameter.

B.Enable job metrics in AWS Glue to estimate the number of data processing units (DPUs). Based on the profiled metrics, increase the value of the maximum capacity job parameter.

C.Enable job metrics in AWS Glue to estimate the number of data processing units (DPUs). Based on the profiled metrics, increase the value of the spark.yarn.executor.memoryOverhead job parameter.

D.Enable job bookmarks in AWS Glue to estimate the number of data processing units (DPUs). Based on the profiled metrics, increase the value of the num- executors job parameter.

Answer: B

Explanation:

The correct answer is **B. Enable job metrics in AWS Glue to estimate the number of data processing units (DPUs). Based on the profiled metrics, increase the value of the maximum capacity job parameter.**

Here's a detailed justification:

AWS Glue job metrics provide valuable insights into resource utilization during job execution. By enabling job metrics, the data analyst can determine if the job is I/O bound, CPU bound, or memory bound. This understanding is crucial for optimizing job performance. Specifically, metrics can highlight how many DPUs are being used, which in turn can help in optimizing the max capacity parameter.

The max capacity parameter in AWS Glue defines the maximum number of DPUs that can be allocated to the job. Increasing this value allows the job to use more DPUs, which can lead to faster processing times, especially for large datasets like the 200 GB dataset in question. If the job is taking a long time and no errors are present, it's likely that the job needs more resources to process the data efficiently.

Job bookmarks (options A and D) are used for incremental processing of data. While helpful for reducing processing time when dealing with frequently updated data, they don't directly address the issue of a slow-running initial job execution.

Options C's spark.yarn.executor.memoryOverhead parameter deals with off-heap memory allocation for Spark executors. While memory management is important, simply increasing this value without understanding the root cause of the slow processing time might not be the most efficient solution. The core issue appears to be a lack of overall processing capacity given the large dataset, which is best addressed by increasing max capacity.

By enabling job metrics and observing DPU utilization, the data analyst can make an informed decision about increasing the max capacity. This allows for efficient scaling of resources without overprovisioning. Monitoring these metrics after the increase helps confirm improved performance.

Further research:

AWS Glue Monitoring: https://docs.aws.amazon.com/glue/latest/dg/monitoring-aws-glue.html

AWS Glue Job Parameters: https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-glue-arguments.html
AWS Glue Worker Types: https://docs.aws.amazon.com/glue/latest/dg/add-job.html

Question: 6

A company has a business unit uploading .csv files to an Amazon S3 bucket. The company's data platform team has set up an AWS Glue crawler to do discovery, and create tables and schemas. An AWS Glue job writes processed data from the created tables to an Amazon Redshift database. The AWS Glue job handles column mapping and creating the Amazon Redshift table appropriately. When the AWS Glue job is rerun for any reason in a day, duplicate records are introduced into the Amazon Redshift table.
Which solution will update the Redshift table without duplicates when jobs are rerun?

A.Modify the AWS Glue job to copy the rows into a staging table. Add SQL commands to replace the existing rows in the main table as postactions in the DynamicFrameWriter class.

B.Load the previously inserted data into a MySQL database in the AWS Glue job. Perform an upsert operation in MySQL, and copy the results to the Amazon Redshift table.

C.Use Apache Spark's DataFrame dropDuplicates() API to eliminate duplicates and then write the data to Amazon Redshift.

D.Use the AWS Glue ResolveChoice built-in transform to select the most recent value of the column.

Answer: A

Explanation:

Here's a detailed justification for why option A is the most suitable solution to prevent duplicate records in Amazon Redshift when rerunning an AWS Glue job processing .csv files from S3:

The core problem is that rerunning the Glue job re-inserts data into Redshift without checking for existing records, leading to duplicates. The most efficient solution involves modifying the Glue job to handle potential duplicate records during the write process to Redshift.

Option A suggests using a staging table within Redshift. This is a common and effective pattern for performing updates or merges in data warehouses. The Glue job initially loads data into a temporary staging table. Then, a SQL command, configured as a post-action, is executed to merge the data from the staging table into the main table. This merge process (using INSERT ... ON CONFLICT DO UPDATE or similar SQL constructs) ensures that only new or updated records are inserted, preventing duplicates. Specifically, the post-action would compare the records in the staging table with those in the main table and only insert new records or update existing ones based on a defined key. The DynamicFrameWriter in Glue provides the functionality to execute these post-actions.

Option B, loading data into MySQL for upserts, introduces unnecessary complexity and latency. It requires setting up and managing an additional database (MySQL), and transferring data multiple times (Glue -> MySQL -> Redshift), making it less efficient. It also adds an additional dependency.

Option C, using Spark's dropDuplicates() API, might seem appealing, but it only removes duplicates within the current batch of data being processed by the Glue job. It doesn't address the issue of duplicates arising from previous job runs. Even if de-duplicated within the job, previous runs will still be re-inserted.

Option D, using ResolveChoice, is intended for handling schema variations, such as different data types within a column. It doesn't inherently prevent duplicate record insertion; it only selects a single representation for potentially conflicting column data types.

Therefore, option A provides a direct and efficient way to manage duplicate records by leveraging the capabilities of Redshift and Glue's post-action feature to perform a proper merge or upsert operation, making it the superior solution. It minimizes data transfer, utilizes Redshift's capabilities and handles the deduplication properly.

Authoritative Links:

AWS Glue DynamicFrameWriter:https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-PaginatorType.html
Amazon Redshift MERGE (Upsert):https://docs.aws.amazon.com/redshift/latest/dg/r_MERGE.html **AWS Glue Best Practices:**https://aws.amazon.com/blogs/big-data/best-practices-for-building-scalable-data-pipelines-with-aws-glue/ (While not directly addressing duplicates, it provides a good overall view on how to approach such operations.)

Question: 7

A streaming application is reading data from Amazon Kinesis Data Streams and immediately writing the data to an Amazon S3 bucket every 10 seconds. The application is reading data from hundreds of shards. The batch interval cannot be changed due to a separate requirement. The data is being accessed by Amazon
Athena. Users are seeing degradation in query performance as time progresses.
Which action can help improve query performance?

A.Merge the files in Amazon S3 to form larger files.

B.Increase the number of shards in Kinesis Data Streams.

C.Add more memory and CPU capacity to the streaming application.

D.Write the files to multiple S3 buckets.

Answer: A

Explanation:

The correct answer is **A. Merge the files in Amazon S3 to form larger files.**

Here's why:

The problem describes a scenario where Athena query performance degrades over time due to a large number of small files in S3. The streaming application is writing data to S3 every 10 seconds from hundreds of shards, leading to numerous small files accumulating rapidly. Athena, being a serverless query service that directly queries data in S3, performs best when dealing with fewer, larger files. Having many small files introduces significant overhead. Athena needs to open and read each of these files individually, which involves numerous metadata requests and I/O operations. This overhead significantly slows down query processing.

Merging the small files into larger, more appropriately sized files reduces this overhead. By decreasing the total number of files that Athena needs to access for each query, you minimize the number of metadata operations and improve data locality, leading to faster query execution. This approach optimizes Athena's performance by reducing the overhead associated with accessing numerous small files. Common file formats that are well-suited for Athena are Parquet and ORC as they are columnar.

Options B, C, and D are incorrect:

B. Increase the number of shards in Kinesis Data Streams: While increasing shards might improve the throughput of the streaming application, it would actually worsen the problem with Athena. More shards mean even more small files being written to S3, further degrading Athena's performance. The problem isn't ingest, it's Athena's performance when querying the data.
C. Add more memory and CPU capacity to the streaming application: Increasing resources for the streaming application addresses potential bottlenecks in data processing. However, this does nothing to alleviate the root cause of the slow Athena queries, which is the large number of small files in S3. It is irrelevant to Athena's query performance.

D. Write the files to multiple S3 buckets: Sharding data across multiple buckets, in and of itself, would not alleviate the problem of numerous small files. Athena would still have to deal with many small files even if they are distributed across multiple S3 buckets. The underlying problem stems from the sheer number of files. While partitioning data can improve query performance if implemented correctly with appropriate filtering, it would add complexity.

Therefore, merging small files into larger files is the most direct and effective solution to address the degradation in Athena query performance.

Supporting Documentation:

Top 10 Performance Tuning Tips for Amazon Athena
Optimizing Amazon Athena Performance

Question: 8

A company uses Amazon OpenSearch Service (Amazon Elasticsearch Service) to store and analyze its website clickstream data. The company ingests 1 TB of data daily using Amazon Kinesis Data Firehose and stores one day's worth of data in an Amazon ES cluster.
The company has very slow query performance on the Amazon ES index and occasionally sees errors from Kinesis Data Firehose when attempting to write to the index. The Amazon ES cluster has 10 nodes running a single index and 3 dedicated master nodes. Each data node has 1.5 TB of Amazon EBS storage attached and the cluster is configured with 1,000 shards. Occasionally, JVMMemoryPressure errors are found in the cluster logs.
Which solution will improve the performance of Amazon ES?

A.Increase the memory of the Amazon ES master nodes.
B.Decrease the number of Amazon ES data nodes.
C.Decrease the number of Amazon ES shards for the index.
D.Increase the number of Amazon ES shards for the index.

Answer: C

Explanation:

The correct answer is **C. Decrease the number of Amazon ES shards for the index.**

Here's a detailed justification:

The primary problem identified is slow query performance, Kinesis Data Firehose errors, and JVMMemoryPressure errors. These symptoms collectively point to excessive sharding. Having 1,000 shards for a 1 TB index means each shard is handling a relatively small amount of data (approximately 1 GB per shard). While sharding is important for horizontal scalability and parallel processing, too many shards can create overhead. Each shard consumes resources (memory, CPU) for management and processing, regardless of the amount of data it holds. This overhead increases query latency and strains the cluster's resources, leading to JVMMemoryPressure errors as the cluster struggles to manage a large number of shards. Kinesis Data Firehose errors can arise when the cluster is overloaded, and it struggles to ingest the data quickly.

Option A (Increase the memory of the Amazon ES master nodes) is not ideal because the master nodes are responsible for cluster management, not data processing. While they do require sufficient memory, increasing master node memory will not directly alleviate the sharding problem. The master nodes are unlikely to be the primary bottleneck in this scenario given the described symptoms.

Option B (Decrease the number of Amazon ES data nodes) would worsen the problem, as the same amount of data would be spread across fewer nodes, increasing the load on each node and exacerbating the memory pressure.

Option D (Increase the number of Amazon ES shards for the index) would dramatically worsen the problem.

Even more shards would lead to even greater overhead, increasing query latency, and more frequent JVMMemoryPressure errors. A good rule of thumb is to aim for shards between 10-50 GB, although this can depend on the use case. 1 GB shards are almost certainly too small.

Reducing the number of shards will reduce the management overhead, improve query performance, and decrease JVMMemoryPressure errors by optimizing resource utilization. It allows the cluster to process data more efficiently by decreasing the number of metadata requests and allowing nodes to work on larger data chunks simultaneously.Therefore, decreasing the number of Amazon ES shards is the most effective solution.Reference links:

Amazon OpenSearch Service Shards
Amazon OpenSearch Service Performance Tuning

Question: 9

A manufacturing company has been collecting IoT sensor data from devices on its factory floor for a year and is storing the data in Amazon Redshift for daily analysis. A data analyst has determined that, at an expected ingestion rate of about 2 TB per day, the cluster will be undersized in less than 4 months. A long-term solution is needed. The data analyst has indicated that most queries only reference the most recent 13 months of data, yet there are also quarterly reports that need to query all the data generated from the past 7 years. The chief technology officer (CTO) is concerned about the costs, administrative effort, and performance of a long-term solution.
Which solution should the data analyst use to meet these requirements?

A.Create a daily job in AWS Glue to UNLOAD records older than 13 months to Amazon S3 and delete those records from Amazon Redshift. Create an external table in Amazon Redshift to point to the S3 location. Use Amazon Redshift Spectrum to join to data that is older than 13 months.

B.Take a snapshot of the Amazon Redshift cluster. Restore the cluster to a new cluster using dense storage nodes with additional storage capacity.

C.Execute a CREATE TABLE AS SELECT (CTAS) statement to move records that are older than 13 months to quarterly partitioned data in Amazon Redshift Spectrum backed by Amazon S3.

D.Unload all the tables in Amazon Redshift to an Amazon S3 bucket using S3 Intelligent-Tiering. Use AWS Glue to crawl the S3 bucket location to create external tables in an AWS Glue Data Catalog. Create an Amazon EMR cluster using Auto Scaling for any daily analytics needs, and use Amazon Athena for the quarterly reports, with both using the same AWS Glue Data Catalog.

Answer: A

Explanation:

The most suitable solution is **A**. Here's why:

Cost-Effectiveness: S3 provides significantly cheaper storage compared to Amazon Redshift. Offloading older data to S3 reduces Redshift storage costs. https://aws.amazon.com/s3/pricing/ and https://aws.amazon.com/redshift/pricing/
Scalability: S3 offers virtually unlimited storage, addressing the long-term data growth concerns.

Performance: Redshift Spectrum allows querying data directly in S3 as if it were in Redshift, enabling seamless access to both recent (in Redshift) and older (in S3) data. This satisfies the need for quarterly reports covering 7 years of data. https://aws.amazon.com/redshift/spectrum/
Administrative Overhead: AWS Glue simplifies the ETL process of unloading and cleaning data before storing it in S3. Creating an external table is straightforward.

Query Optimization: By keeping recent data (13 months) in Redshift, most daily queries will be fast and efficient. The external table using Spectrum only comes into play for less frequent quarterly reports.

Why other options are less ideal:

B: Simply increasing Redshift cluster size delays the problem but doesn't address the core issue of cost-effectively storing infrequently accessed data. It's a short-term fix.

C: Moving data to Redshift Spectrum via CTAS still requires Redshift compute and storage while creating potentially redundant copies of data, affecting both cost and administrative overhead.

D: Using EMR and Athena adds complexity and potentially higher costs for daily analytics compared to directly querying Redshift. Athena is ideal for ad-hoc querying, not necessarily for consistently run daily reports. Also, managing an EMR cluster adds significant operational overhead compared to using Redshift directly.

Question: 10

An insurance company has raw data in JSON format that is sent without a predefined schedule through an Amazon Kinesis Data Firehose delivery stream to an
Amazon S3 bucket. An AWS Glue crawler is scheduled to run every 8 hours to update the schema in the data catalog of the tables stored in the S3 bucket. Data analysts analyze the data using Apache Spark SQL on Amazon EMR set up with AWS Glue Data Catalog as the metastore. Data analysts say that, occasionally, the data they receive is stale. A data engineer needs to provide access to the most up-to-date data.
Which solution meets these requirements?

A.Create an external schema based on the AWS Glue Data Catalog on the existing Amazon Redshift cluster to query new data in Amazon S3 with Amazon Redshift Spectrum.

B.Use Amazon CloudWatch Events with the rate (1 hour) expression to execute the AWS Glue crawler every hour.

C.Using the AWS CLI, modify the execution schedule of the AWS Glue crawler from 8 hours to 1 minute.

D.Run the AWS Glue crawler from an AWS Lambda function triggered by an S3:ObjectCreated:* event notification on the S3 bucket.

Answer: D

Explanation:

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

The problem states that data analysts are experiencing stale data and need access to the most up-to-date information. The current AWS Glue crawler schedule of 8 hours is insufficient to provide near real-time data freshness.

Option D addresses this directly by using an S3 event trigger. When a new object (data) is created in the S3 bucket (S3:ObjectCreated:*), it triggers an AWS Lambda function, which in turn executes the AWS Glue crawler. This means the crawler updates the AWS Glue Data Catalog metadata almost immediately after new data arrives in S3, ensuring the data analysts have access to the most current schema and therefore the latest data. This approach provides an event-driven, near real-time update mechanism, which is more efficient and responsive than scheduled crawling.

Option A, using Amazon Redshift Spectrum, introduces a new component (Redshift) to the architecture and might not be the most cost-effective or efficient solution for simply updating the metadata in AWS Glue Data Catalog. It does not directly address the staleness issue caused by infrequent crawling. Redshift Spectrum is excellent for querying data directly in S3 but is more suited to scenarios needing extensive analytics on the raw data rather than simply cataloging it.

Option B attempts to increase crawler frequency through CloudWatch Events but still relies on a scheduled approach. Even with a 1-hour rate, there's still a potential delay of up to an hour before the schema is updated after new data arrives. This is less responsive than an event-driven solution.

Option C, while increasing the crawler frequency to 1 minute, is an overly aggressive approach. It can lead to unnecessary crawling and increased costs, especially if data is not frequently updated. Moreover, constantly running crawlers can place unnecessary load on both S3 and the AWS Glue service. The S3 event trigger approach is more efficient as it crawls only when there is new data.

Therefore, option D is the best solution as it's event-driven, cost-effective (crawls only when needed), and ensures that the AWS Glue Data Catalog is updated with the most recent schema information close to when the data lands in S3. This provides the data analysts with access to the freshest data available.

Supporting Links:

AWS Glue Crawlers: https://docs.aws.amazon.com/glue/latest/dg/add-crawler.html
AWS Lambda Triggers: https://docs.aws.amazon.com/lambda/latest/dg/services-s3-tutorial.html
Amazon S3 Event Notifications:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html

Question: 11

A company that produces network devices has millions of users. Data is collected from the devices on an hourly basis and stored in an Amazon S3 data lake.
The company runs analyses on the last 24 hours of data flow logs for abnormality detection and to troubleshoot and resolve user issues. The company also analyzes historical logs dating back 2 years to discover patterns and look for improvement opportunities.
The data flow logs contain many metrics, such as date, timestamp, source IP, and target IP. There are about 10 billion events every day.
How should this data be stored for optimal performance?

A.In Apache ORC partitioned by date and sorted by source IP
B.In compressed .csv partitioned by date and sorted by source IP
C.In Apache Parquet partitioned by source IP and sorted by date
D.In compressed nested JSON partitioned by source IP and sorted by date

Answer: A

Explanation:

The optimal data storage strategy for the company's network device logs, given the requirements for both recent and historical analysis, is option A: **In Apache ORC partitioned by date and sorted by source IP.**

Here's why:

1. **Data Lake and S3 Suitability:** Amazon S3 is well-suited for data lakes due to its scalability, durability, and cost-effectiveness.

2. **Partitioning for Performance:** Partitioning by date is crucial. The primary use case is analyzing the last 24 hours of data. Partitioning by date allows queries to target only the relevant partitions, significantly reducing the amount of data scanned. This dramatically improves query performance for real-time analysis and troubleshooting.

3. **ORC File Format Benefits:** Apache ORC (Optimized Row Columnar) is a columnar storage format. Columnar formats excel in analytical workloads because they allow querying only the columns needed for a specific analysis. This minimizes I/O and CPU usage. ORC provides excellent compression and efficient data encoding, further reducing storage costs and improving query speeds. (https://orc.apache.org/)

4. **Sorting for Enhanced Filtering:** Sorting by source IP within each date partition allows for faster filtering and retrieval of logs based on specific source IPs. This is beneficial when troubleshooting user issues or investigating network anomalies originating from particular sources.

5. **Historical Analysis Considerations:** While real-time analysis is prioritized, historical analysis also requires efficient data retrieval. Date partitioning helps limit the scope of queries across large datasets (2 years of data).

6. **Why other options are less suitable:**

   B (Compressed .csv): CSV (Comma Separated Values) is a row-oriented format, which is inefficient for analytical queries. It lacks the compression and encoding capabilities of ORC, leading to higher storage costs and slower query performance.

   C (Apache Parquet partitioned by source IP): While Parquet is also a columnar format, partitioning by source IP is less effective. The primary use case is analyzing the last 24 hours. This option forces queries to scan many source IP partitions even if they are looking for data of only the last 24 hours. Date partitioning is the key to optimize for the recent data access pattern.

   D (Compressed nested JSON): JSON is a semi-structured format. Querying JSON requires parsing the entire document, which is significantly slower than querying columnar formats. Although the nested nature might suit certain data complexities, its impact on performance makes it unsuitable for large-scale analytical workloads.

7. **Cost Optimization:** ORC's compression capabilities reduce storage costs in S3. Efficient querying minimizes processing time and costs associated with services like AWS Athena or Redshift Spectrum.

In summary, the combination of partitioning by date and using the ORC columnar format, sorted by source IP provides the best balance of storage efficiency, query performance, and cost-effectiveness for the company's data analysis requirements. This approach is aligned with best practices for building data lakes on AWS.

Question: 12

A banking company is currently using an Amazon Redshift cluster with dense storage (DS) nodes to store sensitive data. An audit found that the cluster is unencrypted. Compliance requirements state that a database with sensitive data must be encrypted through a hardware security module (HSM) with automated key rotation.
Which combination of steps is required to achieve compliance? (Choose two.)

A.Set up a trusted connection with HSM using a client and server certificate with automatic key rotation.

B.Modify the cluster with an HSM encryption option and automatic key rotation.

C.Create a new HSM-encrypted Amazon Redshift cluster and migrate the data to the new cluster.

D.Enable HSM with key rotation through the AWS CLI.

E.Enable Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) encryption in the HSM.

Answer: AC

Explanation:

Let's break down why options A and C are the correct choices for enabling HSM encryption with automated key rotation for the Amazon Redshift cluster.

Option C is necessary because you cannot directly convert an unencrypted Redshift cluster to an HSM-encrypted cluster. Redshift encryption (including HSM) is configured at cluster creation. Therefore, the only way to achieve HSM encryption is to create a new, HSM-encrypted cluster and migrate the data from the existing unencrypted cluster. This is explicitly mentioned in the AWS documentation.
https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-HSM.html

Option A is crucial because once the new, HSM-encrypted Redshift cluster is created, it needs to establish a secure and trusted connection with the designated HSM (Hardware Security Module). This connection allows Redshift to securely manage encryption keys through the HSM. The AWS documentation outlines the process of configuring this trusted connection, which involves setting up client and server certificates for authentication between Redshift and the HSM. Automatic key rotation is a feature of the HSM itself, ensuring the keys are periodically changed without manual intervention to enhance security.

https://docs.aws.amazon.com/redshift/latest/mgmt/redshift-security.html &
https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-HSM.html

Option B is incorrect because you cannot modify an existing Redshift cluster to use HSM encryption. Encryption must be enabled at cluster creation.

Option D is not sufficient. While the AWS CLI is used to create and configure Redshift clusters, it doesn't enable HSM key rotation directly. Key rotation is configured within the HSM itself.

Option E is incorrect. ECDHE is an encryption key exchange protocol. While it's generally a secure protocol, it is not directly related to HSM integration for Redshift. The focus is on using the HSM to manage the encryption keys, regardless of the specific encryption algorithms or key exchange protocols used by the client applications. The Redshift documentation does not recommend enabling specific types of encryption algorithms or key exchange protocol when using an HSM.

Question: 13

A company is planning to do a proof of concept for a machine learning (ML) project using Amazon SageMaker with a subset of existing on-premises data hosted in the company's 3 TB data warehouse. For part of the project, AWS Direct Connect is established and tested. To prepare the data for ML, data analysts are performing data curation.
The data analysts want to perform multiple step, including mapping, dropping null fields, resolving choice, and splitting fields. The company needs the fastest solution to curate the data for this project.
Which solution meets these requirements?

A.Ingest data into Amazon S3 using AWS DataSync and use Apache Spark scrips to curate the data in an Amazon EMR cluster. Store the curated data in Amazon S3 for ML processing.

B.Create custom ETL jobs on-premises to curate the data. Use AWS DMS to ingest data into Amazon S3 for ML processing.

C.Ingest data into Amazon S3 using AWS DMS. Use AWS Glue to perform data curation and store the data in Amazon S3 for ML processing.

D.Take a full backup of the data store and ship the backup files using AWS Snowball. Upload Snowball data into Amazon S3 and schedule data curation jobs using AWS Batch to prepare the data for ML.

Answer: C

Explanation:

The correct answer is **C. Ingest data into Amazon S3 using AWS DMS. Use AWS Glue to perform data curation and store the data in Amazon S3 for ML processing.**

Here's why:

AWS DMS (Database Migration Service): AWS DMS efficiently moves data from on-premises data warehouses to AWS services like Amazon S3. The project already uses AWS Direct Connect, making DMS a suitable choice for transferring the data.

AWS Glue: AWS Glue is a serverless ETL (Extract, Transform, Load) service designed for data discovery, data transformation, and job scheduling. The data analysts' requirements, including mapping, dropping null fields, resolving choice, and splitting fields, fall under the purview of ETL operations, which Glue can handle effectively. Glue supports multiple transformation types natively.

Amazon S3: Amazon S3 is a highly scalable and durable object storage service ideal for storing data used in ML workflows. Storing the curated data in S3 makes it readily available for SageMaker.

Speed and Cost: This solution strikes a balance between speed and cost. DMS for initial data transfer leverages the existing Direct Connect. Glue's serverless nature means you only pay for what you use, and it is optimized for data curation tasks.

Here's why the other options are less suitable:

A (AWS DataSync and EMR): While AWS DataSync is good for transferring large datasets, using Amazon EMR with Spark for data curation is more complex and potentially slower for the stated transformations compared to Glue. EMR also has operational overhead.

B (Custom ETL on-premises and AWS DMS): Performing ETL on-premises increases the complexity and load on the existing infrastructure, and the question asks for the fastest way. The goal is to leverage AWS services for the data curation process.

D (AWS Snowball and AWS Batch): AWS Snowball is an option for very large initial data transfers if bandwidth is limited. However, since Direct Connect is established, DMS is a faster solution for 3TB. AWS Batch is more suited for batch processing jobs and not designed for interactive data curation.

Authoritative Links:

AWS Database Migration Service (DMS):https://aws.amazon.com/dms/
AWS Glue:https://aws.amazon.com/glue/
Amazon S3:https://aws.amazon.com/s3/

Question: 14

A US-based sneaker retail company launched its global website. All the transaction data is stored in Amazon RDS and curated historic transaction data is stored in Amazon Redshift in the us-east-1 Region. The business intelligence (BI) team wants to enhance the user experience by providing a dashboard for sneaker trends. The BI team decides to use Amazon QuickSight to render the website dashboards. During development, a team in Japan provisioned Amazon QuickSight in ap-northeast-1. The team is having difficulty connecting Amazon QuickSight from ap-northeast-1 to Amazon Redshift in us-east-1.
Which solution will solve this issue and meet the requirements?

A.In the Amazon Redshift console, choose to configure cross-Region snapshots and set the destination Region as ap-northeast-1. Restore the Amazon Redshift Cluster from the snapshot and connect to Amazon QuickSight launched in ap-northeast-1.

B.Create a VPC endpoint from the Amazon QuickSight VPC to the Amazon Redshift VPC so Amazon QuickSight can access data from Amazon Redshift.

C.Create an Amazon Redshift endpoint connection string with Region information in the string and use this connection string in Amazon QuickSight to connect to Amazon Redshift.

D.Create a new security group for Amazon Redshift in us-east-1 with an inbound rule authorizing access from the appropriate IP address range for the Amazon QuickSight servers in ap-northeast-1.

Answer: D

Explanation:

The correct answer is **D**. Here's why:

The primary issue is that Amazon QuickSight in ap-northeast-1 cannot directly access the Amazon Redshift cluster in us-east-1 due to network and security boundaries between AWS Regions. The most direct way to enable this access is to configure network access rules to allow communication.

Option A, replicating the Redshift cluster to ap-northeast-1 is overkill. While it would solve the connectivity issue, it introduces significant operational overhead and cost with data replication, storage, and cluster management. It's unnecessary for the dashboard requirement.

Option B, creating a VPC endpoint, is generally used to allow services within a VPC to connect to other AWS services without going over the public internet. VPC endpoints do not inherently bridge cross-Region network connectivity without additional configurations like VPC peering or Transit Gateway, which are not mentioned as being present.

Option C, modifying the connection string, will not solve the connectivity issue. The connection string contains addressing information, but it does not override the underlying network and security rules that prevent cross-region access.

Option D addresses the security aspect directly. It focuses on configuring the security group associated with the Amazon Redshift cluster in us-east-1. By adding an inbound rule that allows traffic from the IP address range used by Amazon QuickSight servers in ap-northeast-1, we explicitly grant the necessary network access. Each AWS Region has a distinct IP address range used for QuickSight. This is the leanest, most direct method to enabling communication.

Here are some authoritative links for more information:

Amazon QuickSight Security:https://docs.aws.amazon.com/quicksight/latest/user/security-iam.html
Amazon Redshift Security Groups:https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-security-groups.html
AWS Regions and Endpoints:https://docs.aws.amazon.com/general/latest/gr/rande.html

In summary, granting access using the correct IP address range of the QuickSight servers in the security group of Redshift enables access while avoiding unnecessary replication or complex networking solutions.

Question: 15

An airline has .csv-formatted data stored in Amazon S3 with an AWS Glue Data Catalog. Data analysts want to join this data with call center data stored in
Amazon Redshift as part of a daily batch process. The Amazon Redshift cluster is already under a heavy load. The solution must be managed, serverless, well- functioning, and minimize the load on the existing Amazon Redshift cluster. The solution should also require minimal effort and development activity.
Which solution meets these requirements?

A.Unload the call center data from Amazon Redshift to Amazon S3 using an AWS Lambda function. Perform the join with AWS Glue ETL scripts.

B.Export the call center data from Amazon Redshift using a Python shell in AWS Glue. Perform the join with AWS Glue ETL scripts.

C.Create an external table using Amazon Redshift Spectrum for the call center data and perform the join with Amazon Redshift.

D.Export the call center data from Amazon Redshift to Amazon EMR using Apache Sqoop. Perform the join with Apache Hive.

Answer: C

Explanation:

The correct answer is C: Create an external table using Amazon Redshift Spectrum for the call center data and perform the join with Amazon Redshift.

Here's a breakdown of why this is the best solution and why the others aren't ideal:

Redshift Spectrum (Option C): Redshift Spectrum directly addresses the core requirements. It allows Redshift to query data stored in Amazon S3 without loading it into the Redshift cluster itself. This minimizes load on the heavily utilized Redshift cluster. It's serverless (no infrastructure to manage), and well-functioning as Spectrum is designed for querying S3-based data. Minimal development effort is needed to define external tables and write the join query. By leveraging Redshift Spectrum, you're utilizing the Redshift query engine to perform the join, while offloading the storage and processing of the call center data to S3 and Spectrum, respectively.

AWS Glue ETL scripts (Options A & B): While Glue is a serverless ETL service, using it to move data out of Redshift is less efficient and more complex than simply querying it in place using Spectrum. Unloading data and then joining it in Glue introduces unnecessary data movement and ETL logic. Option A mentions Lambda, implying even more orchestration needed. Furthermore, exporting the call center data from Redshift, as suggested in Options A and B, potentially increases the load on the Redshift cluster during the extraction process, contradicting the requirement to minimize the load.

Amazon EMR and Apache Hive (Option D): EMR introduces significant operational overhead. It requires managing an EMR cluster. The goal is a managed and serverless solution, which EMR does not provide. Also, unloading data from Redshift to EMR adds load to the Redshift cluster and requires more significant development efforts than leveraging Redshift Spectrum's capabilities.

In summary, Redshift Spectrum allows the join to happen within the Redshift ecosystem while offloading the data storage and processing related to the external data to a scalable service, aligning perfectly with the requirements of minimizing Redshift load, being managed, serverless, well-functioning, and requiring minimal effort.

Authoritative Links for further research:

Amazon Redshift Spectrum:https://aws.amazon.com/redshift/spectrum/
Using External Tables with Amazon Redshift Spectrum:https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html

Question: 16

A data analyst is using Amazon QuickSight for data visualization across multiple datasets generated by applications. Each application stores files within a separate Amazon S3 bucket. AWS Glue Data Catalog is used as a central catalog across all application data in Amazon S3. A new application stores its data within a separate S3 bucket. After updating the catalog to include the new application data source, the data analyst created a new Amazon QuickSight data source from an Amazon Athena table, but the import into SPICE failed.
How should the data analyst resolve the issue?

A.Edit the permissions for the AWS Glue Data Catalog from within the Amazon QuickSight console.
B.Edit the permissions for the new S3 bucket from within the Amazon QuickSight console.
C.Edit the permissions for the AWS Glue Data Catalog from within the AWS Glue console.
D.Edit the permissions for the new S3 bucket from within the S3 console.

Answer: B

Explanation:

The correct answer is B: Edit the permissions for the new S3 bucket from within the Amazon QuickSight console.

Here's why: The problem is that QuickSight can't access the data in the newly added S3 bucket, which is why the SPICE import is failing. QuickSight requires explicit permissions to access data sources, even if those data sources are cataloged by AWS Glue. Option A and C, editing permissions in Glue console won't resolve the issue. Glue manages the metadata but doesn't grant QuickSight the authorization to directly read data from S3. Editing permissions for the Glue catalog only allows QuickSight to discover the data schema, not access the underlying data. Option D, editing permissions within the S3 console is partially correct. However, the most straightforward and recommended approach is to grant QuickSight access from within the QuickSight console. This provides a centralized location for managing QuickSight's data source permissions and ensures the correct IAM role is being used by QuickSight to access the S3 bucket. Specifically, within QuickSight, when creating or editing a data source connected to Athena, you will see an option to specify which S3 buckets the data source has access to. The data analyst should add the new S3 bucket to the list of accessible buckets for the relevant QuickSight data source. This process ensures that QuickSight's IAM role has the necessary read permissions (s3:GetObject, s3:ListBucket, etc.) to access the data within the new bucket. Therefore, modifying the S3 bucket permissions within the QuickSight console is the most direct and appropriate solution.

Further research:

Connecting to Amazon S3 buckets
Managing Permissions for Amazon S3 Data Sources

Question: 17

A team of data scientists plans to analyze market trend data for their company's new investment strategy. The trend data comes from five different data sources in large volumes. The team wants to utilize Amazon Kinesis to support their use case. The team uses SQL-like queries to analyze trends and wants to send notifications based on certain significant patterns in the trends. Additionally, the data scientists want to save the data to Amazon S3 for archival and historical re- processing, and use AWS managed services wherever possible. The team wants to implement the lowest-cost solution.
Which solution meets these requirements?

A.Publish data to one Kinesis data stream. Deploy a custom application using the Kinesis Client Library (KCL) for analyzing trends, and send notifications using Amazon SNS. Configure Kinesis Data Firehose on the Kinesis data stream to persist data to an S3 bucket.

B.Publish data to one Kinesis data stream. Deploy Kinesis Data Analytic to the stream for analyzing trends, and configure an AWS Lambda function as an output to send notifications using Amazon SNS. Configure Kinesis Data Firehose on the Kinesis data stream to persist data to an S3 bucket.

C.Publish data to two Kinesis data streams. Deploy Kinesis Data Analytics to the first stream for analyzing trends, and configure an AWS Lambda function as an output to send notifications using Amazon SNS.
Configure Kinesis Data Firehose on the second Kinesis data stream to persist data to an S3 bucket.

D.Publish data to two Kinesis data streams. Deploy a custom application using the Kinesis Client Library (KCL) to the first stream for analyzing trends, and send notifications using Amazon SNS. Configure Kinesis Data Firehose on the second Kinesis data stream to persist data to an S3 bucket.

Answer: B

Explanation:

The correct answer is B. Here's why:

The scenario requires a solution that ingests data from multiple sources via Kinesis, allows SQL-like queries for analysis, sends notifications based on patterns, archives data to S3, and leverages AWS managed services for cost optimization.

Kinesis Data Stream: Publishing data to a single Kinesis data stream centralizes the ingestion process, which simplifies the initial data flow from multiple sources. https://aws.amazon.com/kinesis/data-streams/

Kinesis Data Analytics: Using Kinesis Data Analytics fulfills the requirement of performing SQL-like queries on the streaming data. It provides a serverless environment for real-time analytics, reducing operational overhead.
https://aws.amazon.com/kinesis/data-analytics/

AWS Lambda with SNS: The Lambda function, triggered by Kinesis Data Analytics, acts as an output to send notifications using Amazon SNS. This allows for dynamic and event-driven notifications based on the analyzed trends.
https://aws.amazon.com/lambda/ , https://aws.amazon.com/sns/

Kinesis Data Firehose to S3: Configuring Kinesis Data Firehose on the data stream efficiently persists data to an S3 bucket for archival and historical reprocessing. Firehose handles batching, compression, and encryption, reducing the need for custom data management. https://aws.amazon.com/kinesis/data-firehose/

Option A is incorrect because it uses a custom application (KCL), increasing the operational burden and potentially the cost.

Option C is incorrect because it uses two Kinesis Data Streams, adding unnecessary complexity. Using a single stream allows the same data to be consumed by both Kinesis Data Analytics and Firehose.

Option D is incorrect for the same reasons as option A and C (custom application and two Kinesis Data Streams).

Using a single Kinesis Data Stream, Kinesis Data Analytics, Lambda with SNS, and Kinesis Data Firehose is the most cost-effective and manageable solution using AWS managed services.

Question: 18

A company currently uses Amazon Athena to query its global datasets. The regional data is stored in Amazon S3 in the us-east-1 and us-west-2 Regions. The data is not encrypted. To simplify the query process and manage it centrally, the company wants to use Athena in us-west-2 to query data from Amazon S3 in both
Regions. The solution should be as low-cost as possible.
What should the company do to achieve this goal?

A.Use AWS DMS to migrate the AWS Glue Data Catalog from us-east-1 to us-west-2. Run Athena queries in us-west-2.

B.Run the AWS Glue crawler in us-west-2 to catalog datasets in all Regions. Once the data is crawled, run Athena queries in us-west-2.

C.Enable cross-Region replication for the S3 buckets in us-east-1 to replicate data in us-west-2. Once the data is replicated in us-west-2, run the AWS Glue crawler there to update the AWS Glue Data Catalog in us-west-2 and run Athena queries.

D.Update AWS Glue resource policies to provide us-east-1 AWS Glue Data Catalog access to us-west-2. Once the catalog in us-west-2 has access to the catalog in us-east-1, run Athena queries in us-west-2.

Answer: B

Explanation:

The correct answer is B. Let's break down why:

The goal is to query data in S3 buckets located in both us-east-1 and us-west-2 from a central Athena instance in us-west-2, while minimizing cost.

Option B achieves this by using AWS Glue crawlers in us-west-2 to catalog the datasets in both regions. Glue crawlers infer the schema of your data and create/update table metadata in the AWS Glue Data Catalog. This means Athena can query data in us-east-1 and us-west-2 without needing to move or replicate the data. Athena can directly query data residing in other regions if it has metadata about it.

Why other options are incorrect:

A: AWS DMS migration: Migrating the entire Glue Data Catalog from us-east-1 to us-west-2 wouldn't automatically catalog the S3 data in us-west-2. It also doesn't address the data in us-east-1 effectively after the migration. DMS is more suited for database migrations, and it would be overkill and costly for just metadata.

C: Cross-Region Replication: This approach involves replicating all the data from us-east-1 to us-west-2.

Replicating the entire dataset is an expensive and unnecessary data movement operation. It would double the storage costs and increase data transfer costs.

D: Updating Glue Resource Policies: While Glue Resource policies enable cross-account and cross-organization access to Glue resources, the problem specifically requires that the company use Athena in us-west-2. Simply allowing us-west-2 access to us-east-1's catalog doesn't resolve the underlying requirement that Athena in us-west-2 be the primary query engine for both regions. Using crawlers in us-west-2 to create the data catalog in us-west-2 is simpler and more centralized.

Glue crawlers are a cost-effective way to catalog data in S3, and they automatically detect schema changes. By running the crawler in us-west-2, the company maintains a central point of control for its metadata and querying, without incurring the cost of data replication or unnecessary migration.

For further information on Athena cross-region queries and Glue crawlers, refer to these links:

Querying Data in Remote AWS Regions using Athena - AWS Big Data Blog
Populating the AWS Glue Data Catalog - AWS Glue Documentation

Question: 19

A large company receives files from external parties in Amazon EC2 throughout the day. At the end of the day, the files are combined into a single file, compressed into a gzip file, and uploaded to Amazon S3. The total size of all the files is close to 100 GB daily. Once the files are uploaded to Amazon S3, an
AWS Batch program executes a COPY command to load the files into an Amazon Redshift cluster.
Which program modification will accelerate the COPY process?

A.Upload the individual files to Amazon S3 and run the COPY command as soon as the files become available.

B.Split the number of files so they are equal to a multiple of the number of slices in the Amazon Redshift cluster. Gzip and upload the files to Amazon S3. Run the COPY command on the files.

C.Split the number of files so they are equal to a multiple of the number of compute nodes in the Amazon Redshift cluster. Gzip and upload the files to Amazon S3. Run the COPY command on the files.

D.Apply sharding by breaking up the files so the distkey columns with the same values go to the same file. Gzip and upload the sharded files to Amazon S3. Run the COPY command on the files.

Answer: B

Explanation:

The most effective way to accelerate the Amazon Redshift COPY process in this scenario is to split the files into a number equal to a multiple of the slices within the Redshift cluster, then gzip and upload to S3, and finally run the COPY command. (Option B).

Here's a breakdown:

Amazon Redshift is a massively parallel processing (MPP) data warehouse. Its performance is heavily reliant on distributing the workload evenly across all compute nodes and their slices. Each compute node is further divided into slices. A slice processes a portion of the data in parallel. The COPY command leverages this parallelism to load data quickly.

By splitting the input files into a multiple of the number of slices, you maximize parallel loading. For example, if your cluster has 8 slices and you have 16 files, each slice will process two files. If you only had one large file, only one slice would initially work while the others wait, thus underutilizing the Redshift's parallel processing capability. Having a file count that is a multiple of the number of slices ensures more uniform distribution of the workload.

Gzipping the files before uploading to S3 reduces the amount of data that needs to be transferred over the network, further speeding up the process. Redshift efficiently handles gzipped files with the COPY command.

Option A is suboptimal because issuing a COPY command each time a file becomes available creates overhead in establishing connections and parsing the data. The collective overhead of many small COPY operations would likely outweigh the benefits.

Option C is less optimal because it focuses on the number of compute nodes, not the slices within each node, which are the actual processing units. The distribution should be done considering the number of slices.

Option D focuses on sharding data based on the distribution key. While this is beneficial for query performance after the data is loaded (by co-locating related data on the same slice), it doesn't directly optimize the COPY process itself. The problem statement specifically targets accelerating the COPY process, and while a good distribution key is important overall, file splitting is more effective at COPY time parallelization.

Further research:

Amazon Redshift COPY command: https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html Best Practices for Loading Data: https://docs.aws.amazon.com/redshift/latest/dg/loading-data-best-practices.html

Question: 20

A large ride-sharing company has thousands of drivers globally serving millions of unique customers every day. The company has decided to migrate an existing data mart to Amazon Redshift. The existing schema includes the following tables.
➮ A trips fact table for information on completed rides.
➮ A drivers dimension table for driver profiles.
➮ A customers fact table holding customer profile information.
The company analyzes trip details by date and destination to examine profitability by region. The drivers data rarely changes. The customers data frequently changes.
What table design provides optimal query performance?

A.Use DISTSTYLE KEY (destination) for the trips table and sort by date. Use DISTSTYLE ALL for the drivers and customers tables.

B.Use DISTSTYLE EVEN for the trips table and sort by date. Use DISTSTYLE ALL for the drivers table. Use DISTSTYLE EVEN for the customers table.

C.Use DISTSTYLE KEY (destination) for the trips table and sort by date. Use DISTSTYLE ALL for the drivers table. Use DISTSTYLE EVEN for the customers table.

D.Use DISTSTYLE EVEN for the drivers table and sort by date. Use DISTSTYLE ALL for both fact tables.

Answer: C

Explanation:

Here's a detailed justification for why option C is the optimal table design for the given Amazon Redshift scenario, along with supporting concepts and authoritative links:

Justification:

The primary goal is to optimize query performance for analyzing trip details by date and destination, particularly for profitability analysis by region. The most effective strategy involves distributing and sorting data in a way that minimizes data movement across nodes during query execution.

Trips Table (Fact Table): DISTSTYLE KEY (destination) and Sort by Date: Using DISTSTYLE KEY with destination as the distribution key is crucial because the analysis focuses on trip details by destination (region).
This ensures that rows with the same destination are stored on the same compute node, minimizing data

redistribution during queries that filter or aggregate by destination. Sorting by date enables efficient range-based queries when analyzing trends over time.

Drivers Table (Dimension Table - Rarely Changes): DISTSTYLE ALL: The drivers table is relatively small and rarely changes. Using DISTSTYLE ALL replicates the entire table across all nodes. This allows for efficient joins with the trips table because the driver information is readily available on every node, eliminating the need for data movement during joins.

Customers Table (Fact Table - Frequently Changes): DISTSTYLE EVEN: The customers table is large and frequently updated. DISTSTYLE EVEN distributes rows evenly across the compute nodes. This is because of the frequent changes of customers profile information and we do not have any specific column to distribute the rows of customers table. Since data isn't skewed to specific nodes, data updates and queries aren't bottlenecked. While joins with this table might require data redistribution in some cases, the even distribution mitigates potential hot spots and optimizes overall performance, especially given the frequent updates.

Why other options are suboptimal:

Option A: Using DISTSTYLE ALL for the customers table, which changes frequently, will increase the load and cost of updates across all nodes.

Option B:DISTSTYLE EVEN for the trips table wouldn't be optimal since the goal is to analyze trip details by destination. DISTSTYLE EVEN would lead to data redistribution during queries involving destination-based filtering or aggregation.

Option D:DISTSTYLE EVEN for drivers table wouldn't be optimal since the goal is to allow the queries to run faster between trips and drivers table. Using DISTSTYLE ALL for both fact tables would be optimal.

Key Concepts and AWS Redshift Optimization Principles:

Data Distribution Styles: Redshift offers EVEN, KEY, and ALL distribution styles to control how data is distributed across compute nodes.

Sort Keys: Sort keys define the order in which data is stored within each node, enabling efficient range scans and filtering.

Minimize Data Movement: Redshift optimizes query performance by minimizing the need to move data between nodes. Choosing appropriate distribution keys is crucial for this.

Dimension Table Optimization: Small, infrequently changing dimension tables benefit from replication (DISTSTYLE ALL) for faster joins.

Fact Table Distribution: Large fact tables should be distributed based on columns frequently used in joins, filters, or aggregations (DISTSTYLE KEY).

Balanced Distribution:DISTSTYLE EVEN is a good default when no specific join or filter patterns exist, especially when updates are frequent.

Authoritative Links:

Amazon Redshift Documentation - Choosing a Data Distribution Style:
https://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_data.html
Amazon Redshift Documentation - Working with Distribution Styles:
https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_a_dist_key.html
Amazon Redshift Engineering Best Practices:https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/

Question: 21

Three teams of data analysts use Apache Hive on an Amazon EMR cluster with the EMR File System (EMRFS) to

query data stored within each teams Amazon
S3 bucket. The EMR cluster has Kerberos enabled and is configured to authenticate users from the corporate Active Directory. The data is highly sensitive, so access must be limited to the members of each team. Which steps will satisfy the security requirements?

A.For the EMR cluster Amazon EC2 instances, create a service role that grants no access to Amazon S3. Create three additional IAM roles, each granting access to each team's specific bucket. Add the additional IAM roles to the cluster's EMR role for the EC2 trust policy. Create a security configuration mapping for the additional IAM roles to Active Directory user groups for each team.

B.For the EMR cluster Amazon EC2 instances, create a service role that grants no access to Amazon S3. Create three additional IAM roles, each granting access to each team's specific bucket. Add the service role for the EMR cluster EC2 instances to the trust policies for the additional IAM roles. Create a security configuration mapping for the additional IAM roles to Active Directory user groups for each team.

C.For the EMR cluster Amazon EC2 instances, create a service role that grants full access to Amazon S3. Create three additional IAM roles, each granting access to each team's specific bucket. Add the service role for the EMR cluster EC2 instances to the trust polices for the additional IAM roles. Create a security configuration mapping for the additional IAM roles to Active Directory user groups for each team.

D.For the EMR cluster Amazon EC2 instances, create a service role that grants full access to Amazon S3. Create three additional IAM roles, each granting access to each team's specific bucket. Add the service role for the EMR cluster EC2 instances to the trust polices for the base IAM roles. Create a security configuration mapping for the additional IAM roles to Active Directory user groups for each team.

Answer: B

Explanation:

Here's a detailed justification for why option B is the correct solution for securing data access in the given scenario:

The core requirement is to restrict access to S3 data based on team membership, using Active Directory (AD) groups to manage permissions. Option B achieves this through a combination of IAM roles, trust policies, and EMR security configurations.

1. **Restricting Base EMR Role:** Creating an EMR service role with no S3 access is critical. This implements the principle of least privilege, preventing the EMR cluster's base role from accessing any S3 bucket without explicit permission. This improves the overall security posture.

2. **Team-Specific IAM Roles:** Three additional IAM roles are created, each granting access only to the specific S3 bucket for each team. This isolates data access.

3. **Trust Relationship:** The service role for the EMR cluster EC2 instances is added to the trust policies of the team-specific IAM roles. This is the crucial step that allows the EMR cluster instances (acting under the EMR service role) to assume these more restrictive, team-specific IAM roles. It establishes the link between the compute and access to the data. Without this trust policy, the EMR cluster would not be able to use these roles.

4. **Security Configuration Mapping:** An EMR security configuration then maps these team-specific IAM roles to the corresponding Active Directory user groups. This allows EMR to dynamically grant users the appropriate IAM role based on their AD group membership.

5. **Kerberos and AD Integration:** Because Kerberos and AD are integrated, EMR can authenticate users against the AD and determine their group memberships. This enables the security configuration to accurately assign the correct IAM role to the user when they query data using Hive.

In contrast, Option A's approach of adding IAM roles to the EC2 trust policy is incorrect. The EC2 trust policy determines which services can assume the EC2 instance's role. We want the opposite: the EC2 instance (acting as the EMR cluster) to be able to assume the more restricted roles.

Options C and D are incorrect because they grant the EMR service role full access to S3, which violates the principle of least privilege and defeats the purpose of restricting data access.

In summary: Option B sets up a secure system where access to team-specific data is strictly controlled through a combination of restricted IAM roles, trust policies, and security configuration that integrates with Active Directory, fulfilling the key requirement of the problem.

Authoritative Links:

IAM Roles:https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
EMR Security Configurations:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-security-configurations.html
EMRFS authorization using IAM roles:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-fs-iam.html

Question: 22

A company is planning to create a data lake in Amazon S3. The company wants to create tiered storage based on access patterns and cost objectives. The solution must include support for JDBC connections from legacy clients, metadata management that allows federation for access control, and batch-based ETL using PySpark and Scala. Operational management should be limited.
Which combination of components can meet these requirements? (Choose three.)

A.AWS Glue Data Catalog for metadata management
B.Amazon EMR with Apache Spark for ETL
C.AWS Glue for Scala-based ETL
D.Amazon EMR with Apache Hive for JDBC clients
E.Amazon Athena for querying data in Amazon S3 using JDBC drivers
F.Amazon EMR with Apache Hive, using an Amazon RDS with MySQL-compatible backed metastore

Answer: ACE

Explanation:

Here's a breakdown of why the answer is ACE, and why the other options are less suitable, focusing on meeting all the requirements:

A. AWS Glue Data Catalog for metadata management: The Glue Data Catalog is a fully managed, serverless metadata repository that integrates well with other AWS services like Athena and EMR. It's perfect for managing the metadata of your S3 data lake and supports federation for access control. This directly addresses the metadata management requirement, allowing different tools to understand the structure and schema of the data.

C. Amazon Glue for Scala-based ETL: AWS Glue is suitable for performing batch ETL tasks. It supports writing custom ETL jobs using Scala or Python (PySpark). Glue handles the underlying infrastructure, so operational management is limited, which aligns with the requirement. Glue's ability to connect to various data sources and transform data makes it a good fit for batch-based ETL.

E. Amazon Athena for querying data in Amazon S3 using JDBC drivers: Amazon Athena is a serverless query service that enables you to analyze data in S3 using standard SQL. Athena integrates with the Glue Data Catalog, allowing you to query the data lake tables you've defined. Importantly, Athena provides JDBC drivers, which enables legacy clients to connect and query the data lake. This directly addresses the JDBC connection requirement.

Why other options are incorrect:

B. Amazon EMR with Apache Spark for ETL: While EMR with Spark is powerful for ETL, AWS Glue is generally preferred for scenarios emphasizing limited operational management. EMR requires more configuration and management overhead compared to Glue.
D. Amazon EMR with Apache Hive for JDBC clients: While Hive can serve JDBC clients, Athena is a better choice for querying data in S3 directly with JDBC, due to its serverless nature and closer integration with the Glue Data Catalog. Using EMR with Hive introduces unnecessary overhead.
F. Amazon EMR with Apache Hive, using an Amazon RDS with MySQL-compatible backed metastore: This option duplicates the functionality covered more efficiently by the combination of AWS Glue Data Catalog and Athena. Managing an external metastore adds operational complexity, going against the requirement for limited management. It also doesn't natively integrate as seamlessly with PySpark ETL processes.

In Summary: ACE provides a robust, managed solution: Glue Catalog for metadata, Glue for Scala ETL, and Athena with JDBC support for querying. This minimizes operational overhead and satisfies all requirements.

Authoritative Links:

AWS Glue: https://aws.amazon.com/glue/
Amazon Athena: https://aws.amazon.com/athena/
Amazon S3: https://aws.amazon.com/s3/

Question: 23

A company wants to optimize the cost of its data and analytics platform. The company is ingesting a number of .csv and JSON files in Amazon S3 from various data sources. Incoming data is expected to be 50 GB each day. The company is using Amazon Athena to query the raw data in Amazon S3 directly. Most queries aggregate data from the past 12 months, and data that is older than 5 years is infrequently queried. The typical query scans about 500 MB of data and is expected to return results in less than 1 minute. The raw data must be retained indefinitely for compliance requirements.
Which solution meets the company's requirements?

A.Use an AWS Glue ETL job to compress, partition, and convert the data into a columnar data format. Use Athena to query the processed dataset. Configure a lifecycle policy to move the processed data into the Amazon S3 Standard-Infrequent Access (S3 Standard-IA) storage class 5 years after object creation. Configure a second lifecycle policy to move the raw data into Amazon S3 Glacier for long-term archival 7 days after object creation.

B.Use an AWS Glue ETL job to partition and convert the data into a row-based data format. Use Athena to query the processed dataset. Configure a lifecycle policy to move the data into the Amazon S3 Standard-Infrequent Access (S3 Standard-IA) storage class 5 years after object creation. Configure a second lifecycle policy to move the raw data into Amazon S3 Glacier for long-term archival 7 days after object creation.

C.Use an AWS Glue ETL job to compress, partition, and convert the data into a columnar data format. Use Athena to query the processed dataset. Configure a lifecycle policy to move the processed data into the Amazon S3 Standard-Infrequent Access (S3 Standard-IA) storage class 5 years after the object was last accessed. Configure a second lifecycle policy to move the raw data into Amazon S3 Glacier for long-term archival 7 days after the last date the object was accessed.

D.Use an AWS Glue ETL job to partition and convert the data into a row-based data format. Use Athena to query the processed dataset. Configure a lifecycle policy to move the data into the Amazon S3 Standard-Infrequent Access (S3 Standard-IA) storage class 5 years after the object was last accessed. Configure a second lifecycle policy to move the raw data into Amazon S3 Glacier for long-term archival 7 days after the last date the object was accessed.

Answer: A

Explanation:

Here's a detailed justification for why option A is the best solution, along with relevant cloud computing concepts and authoritative links.

Justification:

Option A optimally addresses the company's requirements for cost optimization, query performance, data retention, and data access patterns. The key to cost savings lies in efficient data processing and storage tiering.

1. **Data Transformation (Columnar Format):** Using AWS Glue to transform data into a columnar format (like Parquet or ORC) is crucial. Columnar formats store data by columns instead of rows. Athena benefits significantly from columnar storage because it only reads the necessary columns for a query, greatly reducing the amount of data scanned and therefore lowering costs. This directly addresses the "cost optimization" requirement. The Amazon Athena documentation emphasizes the cost benefits of columnar data formats. https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/

2. **Data Partitioning:** Partitioning the data (e.g., by date) allows Athena to scan only the relevant partitions, further reducing the amount of data processed for queries focused on specific time periods. This directly addresses the "cost optimization" requirement and query performance.

3. **Data Compression:** Compressing the data (e.g., using gzip or Snappy) reduces storage costs and improves I/O performance during query execution.

4. **Athena Integration:** Using Athena to query the transformed data allows the company to leverage its SQL skills without needing to manage a separate data warehouse. This supports the "typical query scans about 500MB of data and is expected to return results in less than 1 minute" requirement.

5. **Lifecycle Policies:** The S3 lifecycle policies effectively manage data storage costs based on access frequency:

   S3 Standard-IA: Moving processed data older than 5 years to S3 Standard-IA provides cost savings for infrequently accessed data while maintaining relatively quick access.

   S3 Glacier: Moving the original raw data to S3 Glacier after 7 days provides the most cost-effective storage for long-term archival, meeting the "raw data must be retained indefinitely for compliance requirements." Glacier is designed for infrequent access and has retrieval time considerations, which are acceptable for archival data. The S3 Storage Classes documentation details the cost and performance tradeoffs. https://aws.amazon.com/s3/storage-classes/

Why other options are less ideal:

Options B & D (Row-based format): Using a row-based format (like CSV) would be significantly less efficient for Athena queries, leading to higher costs and slower performance. Athena is designed to work with columnar formats.

Options C & D (Lifecycle Policy based on Last Access): Using "last access" as the trigger for lifecycle policies is less predictable and potentially more expensive than using "object creation" date. Infrequent access doesn't guarantee the data is no longer needed. The cost would increase if the data that is older than 5 years is accessed again. Also, this method would lead to increased S3 costs as each S3 object requires associated metadata that must be stored and is billed.

In summary, option A provides the most cost-effective solution while meeting all performance, retention, and accessibility requirements. The combination of columnar data format, partitioning, compression, and intelligent storage tiering with S3 lifecycle policies makes it the best choice.

Question: 24

An energy company collects voltage data in real time from sensors that are attached to buildings. The company wants to receive notifications when a sequence of two voltage drops is detected within 10 minutes of a sudden voltage increase at the same building. All notifications must be delivered as quickly as possible. The system must be highly available. The company needs a solution that will automatically scale when this monitoring feature is

implemented in other cities. The notification system is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic for remediation.
Which solution will meet these requirements?

A.Create an Amazon Managed Streaming for Apache Kafka cluster to ingest the data. Use an Apache Spark Streaming with Apache Kafka consumer API in an automatically scaled Amazon EMR cluster to process the incoming data. Use the Spark Streaming application to detect the known event sequence and send the SNS message.

B.Create a REST-based web service by using Amazon API Gateway in front of an AWS Lambda function. Create an Amazon RDS for PostgreSQL database with sufficient Provisioned IOPS to meet current demand. Configure the Lambda function to store incoming events in the RDS for PostgreSQL database, query the latest data to detect the known event sequence, and send the SNS message.

C.Create an Amazon Kinesis Data Firehose delivery stream to capture the incoming sensor data. Use an AWS Lambda transformation function to detect the known event sequence and send the SNS message.

D.Create an Amazon Kinesis data stream to capture the incoming sensor data. Create another stream for notifications. Set up AWS Application Auto Scaling on both streams. Create an Amazon Kinesis Data Analytics for Java application to detect the known event sequence, and add a message to the message stream Configure an AWS Lambda function to poll the message stream and publish to the SNS topic.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most suitable solution, along with explanations of why the other options are less optimal, drawing on cloud computing concepts.

Justification for Option D (Amazon Kinesis Data Streams, Kinesis Data Analytics, Lambda, SNS):

Option D leverages a robust, scalable, and real-time processing pipeline using Kinesis services, ideal for handling high-velocity data streams from sensors. Here's a breakdown:

1. **Kinesis Data Streams for Ingestion:** Kinesis Data Streams is designed for real-time streaming data ingestion. It allows high throughput and can handle the continuous flow of voltage data from numerous sensors across different cities. The use of two streams facilitates separation of concern -one for ingesting data and another specifically for notifications, allowing scalability and efficiency.
https://aws.amazon.com/kinesis/data-streams/

2. **AWS Application Auto Scaling:** Automatically scaling both data streams ensures the system adapts to fluctuating data volumes as the company expands to other cities, meeting the scalability requirement.

3. **Kinesis Data Analytics for Java Application:** Kinesis Data Analytics enables real-time processing of streaming data using SQL or Java. The Java application can analyze the incoming voltage data, detect the specific sequence of events (voltage increase followed by two drops within 10 minutes), and output a message indicating the detection of the event sequence. It ensures low latency processing needed to address the "as quickly as possible" requirement. https://aws.amazon.com/kinesis/data-analytics/

4. **AWS Lambda for SNS Integration:** A Lambda function triggered by the notification stream polls the message stream and then publishes to the SNS topic. This decouples the data processing from the notification mechanism, promoting a more modular and maintainable architecture.
https://aws.amazon.com/lambda/

5. **Amazon SNS for Notification:** SNS provides a highly available and scalable publish/subscribe messaging service, ensuring notifications are reliably delivered to the remediation system.

Why other options are less ideal:

Option A (Kafka, EMR, Spark Streaming): While this option can handle streaming data, it's more complex and resource-intensive than using Kinesis Data Streams and Kinesis Data Analytics. Managing and scaling an EMR cluster is more overhead than using the managed Kinesis services. Also, while Spark Streaming is real-time, Kinesis Data Analytics is designed specifically for this kind of real-time, stream processing use case within AWS.

Option B (API Gateway, Lambda, RDS): Storing every incoming voltage event in an RDS database is not optimal for real-time analysis of streaming data. Relational databases are better suited for storing structured data for querying and reporting, not for the high-velocity ingestion and analysis required here. The latency associated with database queries would also hinder meeting the "as quickly as possible" requirement. Polling the database for sequences would be inefficient.

Option C (Kinesis Data Firehose, Lambda): Kinesis Data Firehose is primarily for loading streaming data into data lakes (like S3 or Redshift) and supports limited transformation via Lambda. It is not designed for complex event processing and sequence detection on streaming data like Kinesis Data Analytics. Firehose cannot be used to trigger events based on sequences, it's used for loading streaming data. Firehose is also not designed to store notifications, it is designed to write data to a data lake destination.

In summary:

Option D provides a solution that is specifically tailored for real-time streaming data processing, leveraging managed AWS services like Kinesis Data Streams and Kinesis Data Analytics for scalability, low latency, and ease of management. It addresses all the requirements in the question in an efficient, scalable, and highly available manner.

Question: 25

A media company has a streaming playback application. The company needs to collect and analyze data to provide near-real-time feedback on playback issues within 30 seconds. The company requires a consumer application to identify playback issues, such as decreased quality during a specified time frame. The data will be streamed in JSON format. The schema can change over time.
Which solution will meet these requirements?

A.Send the data to Amazon Kinesis Data Firehose with delivery to Amazon S3. Configure an S3 event to invoke an AWS Lambda function to process and analyze the data.

B.Send the data to Amazon Managed Streaming for Apache Kafka. Configure Amazon Kinesis Data Analytics for SQL Application as the consumer application to process and analyze the data.

C.Send the data to Amazon Kinesis Data Firehose with delivery to Amazon S3. Configure Amazon S3 to initiate an event for AWS Lambda to process and analyze the data.

D.Send the data to Amazon Kinesis Data Streams. Configure an Amazon Kinesis Data Analytics for Apache Flink application as the consumer application to process and analyze the data.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most suitable solution, along with explanations of why the other options are less ideal:

Why Option D is Correct: Send the data to Amazon Kinesis Data Streams. Configure an Amazon Kinesis Data Analytics for Apache Flink application as the consumer application to process and analyze the data.

Option D utilizes Amazon Kinesis Data Streams for ingesting real-time data, and Kinesis Data Analytics for Apache Flink for processing it. This combination directly addresses the requirements of near-real-time analysis (within 30 seconds) and handling schema evolution.

1. **Real-time Ingestion:** Kinesis Data Streams is designed for high-throughput, real-time data streaming.

It provides the necessary speed and scalability for ingesting streaming playback data. https://aws.amazon.com/kinesis/data-streams/

2. **Near-Real-Time Processing:** Kinesis Data Analytics for Apache Flink is a powerful engine that allows for complex event processing in near real time. Flink excels at handling streaming data and performing stateful computations. https://aws.amazon.com/kinesis/data-analytics/

3. **Schema Evolution:** Flink allows for handling schema changes gracefully because of its capacity to use dynamic data types and JSON parsing functions. These features can handle evolving JSON data structures.

4. **Consumer Application:** The Flink application acts as the consumer, analyzing the data stream and identifying playback issues according to specified criteria (e.g., decreased quality during a time frame). It can then generate alerts or feed the results into a dashboard.

Why Other Options are Incorrect

Option A: Kinesis Data Firehose to S3, S3 event to Lambda: Kinesis Data Firehose is primarily intended for loading data into data lakes or data warehouses, not for near-real-time analytics. It is intended for batch processing, not low latency stream processing. S3 and Lambda are also not suited for streaming applications with 30 seconds requirements.

Option B: Managed Streaming for Apache Kafka (MSK), Kinesis Data Analytics for SQL: While MSK can handle streaming data, this option selects Kinesis Data Analytics for SQL. SQL-based analytics is less flexible in handling schema evolution compared to Flink. The constant altering JSON schema could necessitate complex SQL queries to extract and process information, which may be less effective than Flink's schema handling.

Option C: Kinesis Data Firehose to S3, S3 event to Lambda: This option has the same limitations as Option A.

Firehose/S3 is appropriate for batch data loading, not real-time analysis within 30 seconds. Lambda invocation latency would add significant delay.

Question: 26

An ecommerce company stores customer purchase data in Amazon RDS. The company wants a solution to store and analyze historical data. The most recent 6 months of data will be queried frequently for analytics workloads. This data is several terabytes large. Once a month, historical data for the last 5 years must be accessible and will be joined with the more recent data. The company wants to optimize performance and cost.
Which storage solution will meet these requirements?

A.Create a read replica of the RDS database to store the most recent 6 months of data. Copy the historical data into Amazon S3. Create an AWS Glue Data Catalog of the data in Amazon S3 and Amazon RDS. Run historical queries using Amazon Athena.

B.Use an ETL tool to incrementally load the most recent 6 months of data into an Amazon Redshift cluster. Run more frequent queries against this cluster. Create a read replica of the RDS database to run queries on the historical data.

C.Incrementally copy data from Amazon RDS to Amazon S3. Create an AWS Glue Data Catalog of the data in Amazon S3. Use Amazon Athena to query the data.

D.Incrementally copy data from Amazon RDS to Amazon S3. Load and store the most recent 6 months of data in Amazon Redshift. Configure an Amazon Redshift Spectrum table to connect to all historical data.

Answer: D

Explanation:

Option D is the most suitable storage solution due to its balance of performance, cost-effectiveness, and scalability for both recent and historical data analysis.

Here's a detailed breakdown:

Incremental Copy to S3: Regularly copying data from RDS to S3 provides a cost-effective and durable long-term storage solution for historical data. Amazon S3 offers high durability and availability at a lower cost than keeping all data in RDS or Redshift.

Recent Data in Amazon Redshift: Loading the most recent 6 months of data into Amazon Redshift is optimized for high-performance analytics. Redshift is a columnar data warehouse specifically designed for fast query execution on large datasets. This satisfies the requirement of frequent analytics workloads.

Redshift Spectrum for Historical Data: Configuring Amazon Redshift Spectrum allows Redshift to directly query data stored in S3. This eliminates the need to load all historical data into Redshift, significantly reducing storage costs. Redshift Spectrum bridges the gap between Redshift's high-performance query engine and S3's cost-effective storage.

Optimized for Performance and Cost: This approach offers the best of both worlds: high performance for frequently accessed recent data and cost-effective storage and accessibility for less frequently accessed historical data. The "optimize performance and cost" requirement is therefore best met by option D.

Why other options are less suitable:

Option A: Using Athena directly against RDS and S3 for frequent queries on terabytes of recent data is inefficient and costly. RDS read replicas are not designed for heavy analytics workloads and can impact the performance of the primary database.

Option B: Using a read replica for historical data queries can strain the RDS instance. ETL into Redshift for 6 months of data is good but does not address cost-effectively accessing and joining historical data.

Option C: Querying data directly from S3 using Athena, while cost-effective, might not provide the performance needed for frequent analytics on terabytes of recent data. This misses the requirement of optimized performance.

Relevant links for further research:

Amazon Redshift Spectrum:https://aws.amazon.com/redshift/spectrum/
Amazon Redshift:https://aws.amazon.com/redshift/
Amazon S3:https://aws.amazon.com/s3/
Amazon Athena:https://aws.amazon.com/athena/

Question: 27

A company leverages Amazon Athena for ad-hoc queries against data stored in Amazon S3. The company wants to implement additional controls to separate query execution and query history among users, teams, or applications running in the same AWS account to comply with internal security policies.
Which solution meets these requirements?

A.Create an S3 bucket for each given use case, create an S3 bucket policy that grants permissions to appropriate individual IAM users. and apply the S3 bucket policy to the S3 bucket.

B.Create an Athena workgroup for each given use case, apply tags to the workgroup, and create an IAM policy using the tags to apply appropriate permissions to the workgroup.

C.Create an IAM role for each given use case, assign appropriate permissions to the role for the given use case, and add the role to associate the role with Athena.

D.Create an AWS Glue Data Catalog resource policy for each given use case that grants permissions to appropriate individual IAM users, and apply the resource policy to the specific tables used by Athena.

Answer: B

Explanation:

The correct answer is **B: Create an Athena workgroup for each given use case, apply tags to the workgroup, and create an IAM policy using the tags to apply appropriate permissions to the workgroup.**

Here's why:

Athena Workgroups for Isolation: Athena workgroups provide a mechanism to isolate query execution and query history. Each workgroup acts as a separate environment within Athena, controlling query execution, data access, and cost management. This directly addresses the requirement for separating query execution and history among users, teams, or applications. https://docs.aws.amazon.com/athena/latest/ug/workgroups-concept.html

Tags for Granular Control: Applying tags to Athena workgroups enables fine-grained access control using IAM policies. IAM policies can be written to grant permissions based on these tags. This allows you to define different levels of access to different workgroups based on team membership, application needs, or other relevant criteria. https://docs.aws.amazon.com/IAM/latest/UserGuide/access-control-tags.html

IAM Policies for Permissions: IAM policies using workgroup tags allow you to grant specific permissions to IAM users or roles. For example, you can create an IAM policy that allows users in the "Marketing" team to only execute queries against the "MarketingWorkgroup" using the athena:workgroup tag.

Why other options are less suitable:

A (S3 bucket policies): While S3 bucket policies control access to data stored in S3, they don't provide the necessary isolation for query execution and history within Athena. Users with access to the S3 bucket could still potentially run queries against all data within the bucket through Athena. Also, bucket policies become cumbersome to manage as the number of use cases grows.

C (IAM roles): IAM roles are essential for managing permissions, but creating a role for each use case and associating it directly with Athena is not the best approach for managing query execution and history isolation. Roles define who can assume these permissions, workgroups define what query permissions they can assume.

D (Glue Data Catalog resource policies): Glue Data Catalog resource policies control access to the metadata information about the data stored in S3. While important for data governance, these policies don't directly address the requirement for isolating query execution and query history within Athena. They would need to be combined with other controls.

Question: 28

A company wants to use an automatic machine learning (ML) Random Cut Forest (RCF) algorithm to visualize complex real-world scenarios, such as detecting seasonality and trends, excluding outers, and imputing missing values.
The team working on this project is non-technical and is looking for an out-of-the-box solution that will require the LEAST amount of management overhead.
Which solution will meet these requirements?

A.Use an AWS Glue ML transform to create a forecast and then use Amazon QuickSight to visualize the data.

B.Use Amazon QuickSight to visualize the data and then use ML-powered forecasting to forecast the key business metrics.

C.Use a pre-build ML AMI from the AWS Marketplace to create forecasts and then use Amazon QuickSight to visualize the data.

D.Use calculated fields to create a new forecast and then use Amazon QuickSight to visualize the data.

Answer: B

Explanation:

The correct answer is **B. Use Amazon QuickSight to visualize the data and then use ML-powered forecasting to forecast the key business metrics.**

Here's why:

Least Management Overhead: Amazon QuickSight offers built-in ML-powered forecasting capabilities. This is a managed service, meaning AWS handles the underlying infrastructure, scaling, and maintenance of the ML algorithms. This minimizes the management burden on the non-technical team. Options A and C involve more configuration and maintenance.

Out-of-the-Box Solution: QuickSight provides a user-friendly interface for visualizing data and applying forecasting models without requiring extensive coding or ML expertise. The team can directly leverage QuickSight's features.

RCF for Anomaly Detection: While the question mentions Random Cut Forest, QuickSight's ML forecasting internally incorporates anomaly detection as part of its forecasting process to provide more accurate predictions by identifying and adjusting for outliers.

Direct Integration: QuickSight integrates seamlessly with various AWS data sources (e.g., S3, Redshift, databases), simplifying data ingestion and preparation.

Visualization First: Visualizing the data first in QuickSight allows the team to understand the data patterns before applying any forecasting models. This helps them choose the most appropriate forecasting parameters.

Let's look at why the other options are less suitable:

A. AWS Glue ML transform: Glue ML transforms are more suitable when needing more customization and data preparation before forecasting. In the use case outlined in the question this adds un-needed overhead and is not an out-of-the-box solution.

C. Pre-built ML AMI: An AMI still needs to be launched, configured, and managed, increasing operational complexity. It doesn't align with the need for the least management overhead for a non-technical team. **D. Calculated Fields:** While calculated fields in QuickSight can perform some basic calculations, they lack the sophistication of ML-powered forecasting for handling seasonality, trends, outliers, and missing values effectively.

Authoritative Links:

Amazon QuickSight ML Forecasting:https://aws.amazon.com/quicksight/features/machine-learning-powered-forecasting/

Question: 29

A retail company's data analytics team recently created multiple product sales analysis dashboards for the average selling price per product using Amazon
QuickSight. The dashboards were created from .csv files uploaded to Amazon S3. The team is now planning to share the dashboards with the respective external product owners by creating individual users in Amazon QuickSight. For compliance and governance reasons, restricting access is a key requirement. The product owners should view only their respective product analysis in the dashboard reports.
Which approach should the data analytics team take to allow product owners to view only their products in the dashboard?

A.Separate the data by product and use S3 bucket policies for authorization.

B.Separate the data by product and use IAM policies for authorization.

C.Create a manifest file with row-level security.

D.Create dataset rules with row-level security.

Answer: D

Explanation:

The correct answer is D: Create dataset rules with row-level security. Here's a detailed justification:

Row-Level Security (RLS) in Amazon QuickSight is the appropriate solution for restricting data access within dashboards based on user attributes. The requirement states that external product owners should only see data related to their specific products. RLS directly addresses this need.

By creating dataset rules, you define filters that are applied based on the user viewing the dashboard. These rules effectively limit the data displayed to each product owner to only their respective product analysis. You would configure these rules using the product owner's username or a group they belong to, linking them to the appropriate product data.

Options A and B are not suitable because they focus on controlling access at the S3 bucket or IAM level, which is too broad for this specific requirement. S3 bucket policies and IAM policies manage access to entire files or resources; they can't filter data within a QuickSight dashboard based on the user. While you could create separate S3 buckets for each product owner, this is administratively complex and not a scalable or efficient solution for data analysis.

Option C, using a manifest file with row-level security, is a viable approach in certain contexts, particularly for controlling access to underlying S3 data. However, it is more commonly used for managing data sources.
While it could potentially be combined with QuickSight RLS, it isn't the simplest or most direct approach. QuickSight's built-in dataset rules for RLS are specifically designed for this kind of use case, making option D more appropriate. QuickSight dataset rules are specifically created to filter what data is displayed on the dashboard based on the user.

Therefore, using QuickSight dataset rules with row-level security provides the granular control necessary to restrict access to specific product data for individual product owners, meeting compliance and governance requirements in the most efficient and straightforward way.

Further research:

Amazon QuickSight Row-Level Security:https://docs.aws.amazon.com/quicksight/latest/user/restrict-access-to-a-data-set-using-row-level-security.html

Question: 30

A company has developed an Apache Hive script to batch process data stared in Amazon S3. The script needs to run once every day and store the output in
Amazon S3. The company tested the script, and it completes within 30 minutes on a small local three-node cluster. Which solution is the MOST cost-effective for scheduling and executing the script?

A.Create an AWS Lambda function to spin up an Amazon EMR cluster with a Hive execution step. Set KeepJobFlowAliveWhenNoSteps to false and disable the termination protection flag. Use Amazon CloudWatch Events to schedule the Lambda function to run daily.

B.Use the AWS Management Console to spin up an Amazon EMR cluster with Python Hue. Hive, and Apache Oozie. Set the termination protection flag to true and use Spot Instances for the core nodes of the cluster. Configure an Oozie workflow in the cluster to invoke the Hive script daily.

C.Create an AWS Glue job with the Hive script to perform the batch operation. Configure the job to run once a day using a time-based schedule.

D.Use AWS Lambda layers and load the Hive runtime to AWS Lambda and copy the Hive script. Schedule the Lambda function to run daily by creating a workflow using AWS Step Functions.

Answer: A

Explanation:

Here's a detailed justification for why option A is the most cost-effective solution for scheduling and executing the Hive script:

Option A leverages a combination of serverless and managed services to optimize costs. AWS Lambda, being a serverless compute service, only charges you for the actual compute time consumed. This is significantly cheaper than running a persistent cluster, especially since the script completes within 30 minutes. Spinning up an Amazon EMR cluster on demand using Lambda for a short duration is far more cost-efficient than keeping an EMR cluster constantly running, as suggested by options B.

The "KeepJobFlowAliveWhenNoSteps" set to false ensures that the EMR cluster terminates immediately after the Hive script completes. Disabling termination protection further ensures the cluster doesn't remain active unintentionally, preventing unnecessary charges. Amazon CloudWatch Events provides a cost-effective scheduling mechanism, triggering the Lambda function daily without incurring ongoing charges for a dedicated scheduler.

Option B involves running a persistent EMR cluster which can result in unnecessary costs, even with Spot Instances. Spot Instances provide cost savings, but also carry the risk of interruption. Termination protection will lead to high cost. Additionally, managing Oozie on a persistent cluster requires administrative overhead.

Option C might seem appealing due to AWS Glue's serverless nature, but Hive scripts are not directly compatible with AWS Glue's primary purpose of ETL operations using Spark. Converting the Hive script to a Spark-based job might require significant refactoring effort, increasing the initial cost and complexity.

Option D is impractical. Lambda has limited runtime and storage capacity. Loading the entire Hive runtime into Lambda would likely exceed these limitations. Hive requires a distributed environment which Lambda doesn't naturally provide, making this approach infeasible. Furthermore, step functions would add unnecessary overhead and complexity for such a simple scheduled task.

Therefore, the dynamic, on-demand provisioning of an EMR cluster with a short lifespan triggered by a cost-effective scheduling mechanism like CloudWatch Events makes option A the most cost-effective solution.

Supporting links:

AWS Lambda Pricing:https://aws.amazon.com/lambda/pricing/
Amazon EMR Pricing:https://aws.amazon.com/emr/pricing/
Amazon CloudWatch Events (now EventBridge):https://aws.amazon.com/eventbridge/ (Examine pricing and understand scheduling capabilities)

Question: 31

A company wants to improve the data load time of a sales data dashboard. Data has been collected as .csv files and stored within an Amazon S3 bucket that is partitioned by date. The data is then loaded to an Amazon Redshift data warehouse for frequent analysis. The data volume is up to 500 GB per day.
Which solution will improve the data loading performance?

A.Compress .csv files and use an INSERT statement to ingest data into Amazon Redshift.
B.Split large .csv files, then use a COPY command to load data into Amazon Redshift.
C.Use Amazon Kinesis Data Firehose to ingest data into Amazon Redshift.
D.Load the .csv files in an unsorted key order and vacuum the table in Amazon Redshift.

Answer: B

Explanation:

The most effective way to improve data loading performance into Amazon Redshift, given the described scenario, is option B: splitting large .csv files and using the COPY command. Here's why:

The COPY command is explicitly designed for bulk loading data into Amazon Redshift from sources like Amazon S3. It leverages Redshift's massively parallel processing (MPP) architecture, distributing the data load across multiple compute nodes simultaneously, resulting in significantly faster ingestion compared to row-by-row insertion methods. [https://docs.aws.amazon.com/redshift/latest/dg/t_loading-tables-from-s3.html]

Splitting large .csv files into smaller ones enhances parallelism during the COPY operation. Redshift can then process these smaller files concurrently, further accelerating the overall load time. This distribution of work across multiple nodes is a key benefit of Redshift's architecture.

Option A, while suggesting compression (which is beneficial), relies on INSERT statements. INSERT statements load data one row at a time, which is significantly slower for large datasets like 500 GB per day. This approach doesn't leverage Redshift's parallel processing capabilities effectively.

Option C, using Amazon Kinesis Data Firehose, is more suitable for streaming data ingestion. While Firehose can deliver data to Redshift, it adds an unnecessary layer of complexity for data already residing in S3. It's also not as optimized for bulk loading from files as the COPY command.

Option D, focusing on unsorted keys and vacuuming, addresses query performance after the data is loaded. While crucial for maintaining data integrity and query speed, it doesn't directly improve the initial data load time. Vacuuming and sorting are post-load optimizations. Loading data sorted does help, but splitting files is generally more impactful for initial load.

Therefore, the combination of splitting the large .csv files for improved parallelism and using the COPY command for efficient bulk loading provides the most direct and effective improvement to the data loading performance into Amazon Redshift. Compressing the data as well could add further speed improvements.

Question: 32

A company has a data warehouse in Amazon Redshift that is approximately 500 TB in size. New data is imported every few hours and read-only queries are run throughout the day and evening. There is a particularly heavy load with no writes for several hours each morning on business days. During those hours, some queries are queued and take a long time to execute. The company needs to optimize query execution and avoid any downtime.
What is the MOST cost-effective solution?

A.Enable concurrency scaling in the workload management (WLM) queue.

B.Add more nodes using the AWS Management Console during peak hours. Set the distribution style to ALL.

C.Use elastic resize to quickly add nodes during peak times. Remove the nodes when they are not needed.

D.Use a snapshot, restore, and resize operation. Switch to the new target cluster.

Answer: A

Explanation:

The most cost-effective solution is to enable concurrency scaling in the workload management (WLM) queue. Here's why:

Concurrency Scaling: Amazon Redshift concurrency scaling automatically adds compute capacity to handle spikes in read concurrency, ensuring queries execute quickly even during peak load periods. This is ideal for the described scenario where there's a consistent, predictable peak load.

Cost Efficiency: Concurrency scaling is billed per second of usage, only when extra capacity is required. This "pay-as-you-go" model makes it significantly more cost-effective than permanently adding nodes.

No Downtime: Concurrency scaling seamlessly adds capacity without any downtime or requiring any manual intervention. This fulfills the requirement of avoiding downtime.

WLM Integration: Enabling concurrency scaling within WLM allows you to control which queries benefit from the additional capacity, further optimizing resource utilization.

Alternative B (Adding Nodes Manually): Adding nodes manually (Option B) requires constant monitoring and intervention, which is operationally burdensome. Setting the distribution style to ALL is generally not recommended for tables of that size due to storage and performance implications.

Alternative C (Elastic Resize): While elastic resize does add nodes quickly, it still requires manual intervention to resize up and down, and while it minimizes downtime, it still leads to a brief outage. Also, it's billed for entire resize durations, even if nodes aren't fully utilized.

Alternative D (Snapshot, Restore, Resize): Snapshot, restore, and resize is designed for major cluster changes, like different node types, and is unsuitable for short-term peak load management because it involves a complete cluster replacement and unacceptable downtime.

Therefore, concurrency scaling provides the necessary elasticity, automation, cost-effectiveness, and zero downtime that this company needs to optimize query execution during peak morning loads.

Authoritative Links:

Amazon Redshift Concurrency Scaling:https://docs.aws.amazon.com/redshift/latest/dg/concurrency-scaling.html
Amazon Redshift Workload Management (WLM):https://docs.aws.amazon.com/redshift/latest/dg/workload-management.html
Amazon Redshift Resizing Clusters:https://docs.aws.amazon.com/redshift/latest/dg/t_resizing-clusters.html

Question: 33

A company analyzes its data in an Amazon Redshift data warehouse, which currently has a cluster of three dense storage nodes. Due to a recent business acquisition, the company needs to load an additional 4 TB of user data into Amazon Redshift. The engineering team will combine all the user data and apply complex calculations that require I/O intensive resources. The company needs to adjust the cluster's capacity to support the change in analytical and storage requirements.
Which solution meets these requirements?

A.Resize the cluster using elastic resize with dense compute nodes.
B.Resize the cluster using classic resize with dense compute nodes.
C.Resize the cluster using elastic resize with dense storage nodes.
D.Resize the cluster using classic resize with dense storage nodes.

Answer: A

Explanation:

The correct answer is **A. Resize the cluster using elastic resize with dense compute nodes.**

Here's a detailed justification:

The scenario describes an increase in both storage (4TB) and computational needs (complex calculations, I/O intensive). Therefore, choosing the right node type and resize method is crucial.

Dense Compute Nodes: The problem states that the calculations are I/O intensive, implying that the company needs faster CPUs, more RAM, and SSD storage for faster data processing. Dense compute nodes, optimized for high performance, are appropriate for these complex calculations. Dense storage nodes are designed for large storage capacity but are less performant computationally.

Elastic Resize: Elastic resize is a faster and more efficient method for resizing an Amazon Redshift cluster compared to classic resize. It allows for scaling without a prolonged downtime as it scales without requiring a full data reload. Classic resize involves creating a new cluster and moving the data, resulting in substantial downtime. Given that the company needs to quickly adapt to the new data load, Elastic Resize is preferred.

Therefore, using elastic resize to switch to dense compute nodes will address both the computational needs for the complex calculations (due to the I/O intensive nature of them) and the increased storage requirements. Options C and D which mentions Dense storage nodes are less optimal since this type of node mainly focuses on increasing storage and not computational power. Moreover, option B is incorrect as Elastic Resize is more effective than Classic Resize.

In Summary: Answer A efficiently addresses the I/O-intensive computational needs and the storage increase with minimal downtime by combining dense compute nodes and elastic resize.

Supporting Links:

Amazon Redshift Resizing
Amazon Redshift Node Types: This documentation describes the differences between node types, highlighting the I/O performance benefits of dense compute nodes in this specific situation.

Question: 34

A company stores its sales and marketing data that includes personally identifiable information (PII) in Amazon S3. The company allows its analysts to launch their own Amazon EMR cluster and run analytics reports with the data. To meet compliance requirements, the company must ensure the data is not publicly accessible throughout this process. A data engineer has secured Amazon S3 but must ensure the individual EMR clusters created by the analysts are not exposed to the public internet.
Which solution should the data engineer to meet this compliance requirement with LEAST amount of effort?

A.Create an EMR security configuration and ensure the security configuration is associated with the EMR clusters when they are created.

B.Check the security group of the EMR clusters regularly to ensure it does not allow inbound traffic from IPv4 0.0.0.0/0 or IPv6 ::/0.

C.Enable the block public access setting for Amazon EMR at the account level before any EMR cluster is created.

D.Use AWS WAF to block public internet access to the EMR clusters across the board.

Answer: C

Explanation:

The correct answer is C, enabling Block Public Access for Amazon EMR at the account level. Here's why:

Why C is the best solution:

Centralized Control: Block Public Access provides a centralized and straightforward mechanism to prevent unintentional public exposure of EMR clusters. Enabling it at the account level ensures that every newly created EMR cluster will automatically inherit the restriction.
Least Effort: This option requires a one-time configuration change at the account level, minimizing ongoing administrative overhead compared to other solutions.
Prevention is Better than Cure: It proactively prevents public access issues instead of relying on reactive

monitoring or security group adjustments.

Compliance: Directly addresses the compliance requirement of preventing publicly accessible clusters and PII exposure.

Why other options are less suitable:

A (EMR Security Configuration): While security configurations offer comprehensive security features, they are more complex and might require analysts to correctly associate them with each cluster, increasing the chance of misconfiguration. Also, Security Configurations are primarily for encryption and authentication, not directly for blocking public access to the cluster itself.

B (Regularly check Security Groups): This approach is reactive, manual, and prone to human error. It requires continuous monitoring of each cluster's security group settings, making it less scalable and less reliable than a preventative solution.

D (AWS WAF): WAF is primarily designed to protect web applications against malicious traffic. While it can be configured to block access to EMR clusters, it adds unnecessary complexity since EMR clusters don't inherently expose web interfaces and is more geared towards network level application security. It would be an overkill and add unnecessary management overhead.

Supporting Concepts & Links:

Amazon EMR Block Public Access: This feature directly addresses the requirement of preventing public accessibility to EMR clusters, making it the most efficient and reliable solution.
(https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-block-public-access.html) **Security Best Practices for Amazon EMR:** Provides guidance on securing EMR clusters.
(https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-security-best-practices.html)

In summary, enabling Block Public Access for Amazon EMR at the account level is the best approach because it is a proactive, centralized, easy-to-implement, and compliant solution for preventing public exposure of EMR clusters, aligning with the principle of least effort.

Question: 35

A financial company uses Amazon S3 as its data lake and has set up a data warehouse using a multi-node Amazon Redshift cluster. The data files in the data lake are organized in folders based on the data source of each data file.
All the data files are loaded to one table in the Amazon Redshift cluster using a separate
COPY command for each data file location. With this approach, loading all the data files into Amazon Redshift takes a long time to complete. Users want a faster solution with little or no increase in cost while maintaining the segregation of the data files in the S3 data lake.
Which solution meets these requirements?

A.Use Amazon EMR to copy all the data files into one folder and issue a COPY command to load the data into Amazon Redshift.

B.Load all the data files in parallel to Amazon Aurora, and run an AWS Glue job to load the data into Amazon Redshift.

C.Use an AWS Glue job to copy all the data files into one folder and issue a COPY command to load the data into Amazon Redshift.

D.Create a manifest file that contains the data file locations and issue a COPY command to load the data into Amazon Redshift.

Answer: D

Explanation:

The best solution is to use a manifest file and the COPY command in Amazon Redshift (Option D). Here's why:

Efficiency of COPY command: The COPY command is the most efficient way to load data into Amazon Redshift from Amazon S3. It's optimized for parallel loading, especially when reading multiple files simultaneously.

Manifest files for multiple files: Redshift COPY command supports manifest files, which are JSON files that list the S3 objects to load. Using a manifest file avoids needing to run separate COPY commands for each data file, significantly reducing the overhead of managing multiple load processes.

https://docs.aws.amazon.com/redshift/latest/dg/r_COPY_command_examples.html#r_COPY_command_examples-copy-from-s3

No data movement (until load): This approach doesn't require moving or consolidating the data files in S3 before loading, preserving the existing folder-based organization based on data source. This maintains the current segregation requirements while minimizing data preparation steps.

Cost-effectiveness: Manifest files and the COPY command are built-in Redshift features. They do not introduce additional service costs, fulfilling the requirement of little to no increase in cost.

Alternatives are less optimal:

Option A (Amazon EMR) and C (AWS Glue) involve the extra step of copying all the data files into one folder. This adds complexity, compute costs, and potentially unnecessary data movement. While these services can handle large datasets, using them solely for data consolidation before loading into Redshift isn't as efficient as leveraging the COPY command directly.

Option B (Amazon Aurora and Glue) introduces an entirely new database system (Aurora) and an ETL process that are not required and are unnecessary for the task.

Preserves folder structure and parallelism: Even with the manifest file, Redshift can still load data from different folders in parallel, if the data is partitioned suitably.

Simplified Management: This approach reduces the overall complexity of the data loading process, making it easier to manage and maintain.

In summary, using a manifest file with the COPY command provides a cost-effective, efficient, and simple solution for loading data into Amazon Redshift while maintaining the organization of data files in the data lake and avoiding unnecessary data duplication.

Question: 36

A company's marketing team has asked for help in identifying a high performing long-term storage service for their data based on the following requirements:

⇨ The data size is approximately 32 TB uncompressed.
⇨ There is a low volume of single-row inserts each day.
⇨ There is a high volume of aggregation queries each day.
⇨ Multiple complex joins are performed.
⇨ The queries typically involve a small subset of the columns in a table.

Which storage service will provide the MOST performant solution?

A.Amazon Aurora MySQL

B.Amazon Redshift

C.Amazon Neptune

D.Amazon Elasticsearch

Answer: B

Explanation:

Amazon Redshift is the most performant solution for this scenario due to its columnar storage and massively parallel processing (MPP) architecture, which are optimized for analytical workloads involving large datasets and complex queries. The requirement of "high volume of aggregation queries each day" aligns perfectly with Redshift's strengths. Redshift's columnar storage efficiently stores and retrieves only the necessary columns involved in the queries, satisfying the requirement that "queries typically involve a small subset of the columns in a table." The ability to perform "multiple complex joins" is also well-supported by Redshift's query engine. The 32 TB uncompressed data size falls within Redshift's operational range. While there are low volume inserts, Redshift handles this effectively when batch loading data.

Aurora MySQL is primarily designed for transactional workloads (OLTP) rather than analytical ones (OLAP). Although it can handle some analytical queries, it is not optimized for the scale and complexity described in the question. Amazon Neptune is a graph database service, suitable for relationship-heavy data but less effective for aggregation queries on structured data. Amazon Elasticsearch is optimized for search and log analytics, not for complex joins and large-scale aggregation over a 32 TB dataset. Therefore, Redshift's columnar storage and MPP architecture are the key features that make it superior for this use case.

Amazon Redshift DocumentationAmazon Redshift Use Cases

Question: 37

A technology company is creating a dashboard that will visualize and analyze time-sensitive data. The data will come in through Amazon Kinesis Data Firehose with the butter interval set to 60 seconds. The dashboard must support near-real-time data.
Which visualization solution will meet these requirements?

A.Select Amazon OpenSearch Service (Amazon Elasticsearch Service) as the endpoint for Kinesis Data Firehose. Set up an OpenSearch Dashboards (Kibana) using the data in Amazon OpenSearch Service (Amazon ES) with the desired analyses and visualizations.

B.Select Amazon S3 as the endpoint for Kinesis Data Firehose. Read data into an Amazon SageMaker Jupyter notebook and carry out the desired analyses and visualizations.

C.Select Amazon Redshift as the endpoint for Kinesis Data Firehose. Connect Amazon QuickSight with SPICE to Amazon Redshift to create the desired analyses and visualizations.

D.Select Amazon S3 as the endpoint for Kinesis Data Firehose. Use AWS Glue to catalog the data and Amazon Athena to query it. Connect Amazon QuickSight with SPICE to Athena to create the desired analyses and visualizations.

Answer: A

Explanation:

The correct answer is A because it provides a near-real-time visualization solution for time-sensitive data delivered by Kinesis Data Firehose. Kinesis Data Firehose directly integrates with Amazon OpenSearch Service (successor to Amazon Elasticsearch Service). Configuring OpenSearch Service as the Firehose destination allows for automatic indexing and ingestion of incoming data. OpenSearch Dashboards (Kibana) is designed for near-real-time data exploration and visualization, enabling timely analysis of the streamed data. The relatively low buffering interval of 60 seconds in Firehose further enhances the near-real-time nature of the dashboard.

Option B is not optimal because using Amazon S3 as the endpoint and then reading the data into a SageMaker Jupyter notebook introduces significant latency. Jupyter notebooks are not designed for real-time visualization, and manually running analyses on incoming data would be cumbersome and slow.

Option C, using Amazon Redshift, is a data warehousing solution and is better suited for batch processing and historical analysis rather than near-real-time visualization. While QuickSight can connect to Redshift, the SPICE engine introduces a delay as data needs to be imported into it.

Option D, involving Amazon S3, AWS Glue, and Amazon Athena, also introduces significant delays due to the need for data cataloging, querying, and SPICE importing. This approach is geared towards ad-hoc querying and reporting, not near-real-time dashboards. Athena is not designed for the very low latency queries needed for near-real-time visualization.

Therefore, the direct integration of Kinesis Data Firehose with Amazon OpenSearch Service, combined with the near-real-time visualization capabilities of OpenSearch Dashboards, makes option A the most suitable solution for the stated requirements. The other options are not optimized for the near-real-time requirement stated in the question.

Authoritative Links:

Amazon Kinesis Data Firehose:https://aws.amazon.com/kinesis/data-firehose/
Amazon OpenSearch Service:https://aws.amazon.com/opensearch-service/
OpenSearch Dashboards:https://opensearch.org/docs/latest/dashboards/index/

Question: 38

A financial company uses Apache Hive on Amazon EMR for ad-hoc queries. Users are complaining of sluggish performance.
A data analyst notes the following:
➯ Approximately 90% of queries are submitted 1 hour after the market opens.
Hadoop Distributed File System (HDFS) utilization never exceeds 10%.

Which solution would help address the performance issues?

A.Create instance fleet configurations for core and task nodes. Create an automatic scaling policy to scale out the instance groups based on the Amazon CloudWatch CapacityRemainingGB metric. Create an automatic scaling policy to scale in the instance fleet based on the CloudWatch CapacityRemainingGB metric.

B.Create instance fleet configurations for core and task nodes. Create an automatic scaling policy to scale out the instance groups based on the Amazon CloudWatch YARNMemoryAvailablePercentage metric. Create an automatic scaling policy to scale in the instance fleet based on the CloudWatch YARNMemoryAvailablePercentage metric.

C.Create instance group configurations for core and task nodes. Create an automatic scaling policy to scale out the instance groups based on the Amazon CloudWatch CapacityRemainingGB metric. Create an automatic scaling policy to scale in the instance groups based on the CloudWatch CapacityRemainingGB metric.

D.Create instance group configurations for core and task nodes. Create an automatic scaling policy to scale out the instance groups based on the Amazon CloudWatch YARNMemoryAvailablePercentage metric. Create an automatic scaling policy to scale in the instance groups based on the CloudWatch YARNMemoryAvailablePercentage metric.

Answer: D

Explanation:

Correct answer is D as instance group configurations for core and task nodes can be used to scale as per the YARNMemoryAvailablePercentage metric.options A & B are incorrect because an Instance Fleet doesn't have an automatic scaling policy. Only an Instance Group has this feature.Option C is incorrect as the CapacityRemainingGB metric is just the amount of remaining HDFS disk capacity and this does not exceed 10% for each run. The cluster will not scale-in or scale-out if you choose this metric.

Question: 39

A media company has been performing analytics on log data generated by its applications. There has been a recent increase in the number of concurrent analytics jobs running, and the overall performance of existing jobs is decreasing as the number of new jobs is increasing. The partitioned data is stored in
Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA) and the analytic processing is performed on Amazon EMR clusters using the EMR File System
(EMRFS) with consistent view enabled. A data analyst has determined that it is taking longer for the EMR task

nodes to list objects in Amazon S3.
Which action would MOST likely increase the performance of accessing log data in Amazon S3?

A.Use a hash function to create a random string and add that to the beginning of the object prefixes when storing the log data in Amazon S3.

B.Use a lifecycle policy to change the S3 storage class to S3 Standard for the log data.

C.Increase the read capacity units (RCUs) for the shared Amazon DynamoDB table.

D.Redeploy the EMR clusters that are running slowly to a different Availability Zone.

Answer: C

Explanation:

The correct answer is **C. Increase the read capacity units (RCUs) for the shared Amazon DynamoDB table.**

The problem describes performance degradation in EMR jobs accessing log data in S3, specifically identifying object listing as a bottleneck. EMRFS with consistent view enabled uses a DynamoDB table to maintain metadata about objects in S3. This consistent view ensures that EMR jobs see a consistent state of data even when objects are being added, updated, or deleted in S3. Each time an EMR task node needs to list objects (e.g., to find input data for a job), it queries the DynamoDB table.

When the number of concurrent analytics jobs increases, so does the number of requests to the DynamoDB table. If the read capacity of the table is insufficient to handle this increased load, the DynamoDB table will throttle requests. This throttling causes significant latency in object listing, directly impacting the overall performance of EMR jobs. Increasing the Read Capacity Units (RCUs) allows DynamoDB to handle more read requests per second, reducing throttling and improving the performance of EMRFS consistent view operations.

Option A might help with S3 listing limits but consistent view relies on DynamoDB lookups. Option B changes storage tier and does not impact metadata lookups. Option D does not impact metadata operations and instead adds complexity.

Therefore, increasing RCU for DynamoDB is the most direct approach to mitigate the performance bottleneck, as it specifically addresses the increased load on the EMRFS consistent view metadata store.

Further Research:

EMRFS Consistent View:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emrfs-consistent-view.html
Amazon DynamoDB Scalability:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html

Question: 40

A company has developed several AWS Glue jobs to validate and transform its data from Amazon S3 and load it into Amazon RDS for MySQL in batches once every day. The ETL jobs read the S3 data using a DynamicFrame. Currently, the ETL developers are experiencing challenges in processing only the incremental data on every run, as the AWS Glue job processes all the S3 input data on each run.
Which approach would allow the developers to solve the issue with minimal coding effort?

A.Have the ETL jobs read the data from Amazon S3 using a DataFrame.

B.Enable job bookmarks on the AWS Glue jobs.

C.Create custom logic on the ETL jobs to track the processed S3 objects.

D.Have the ETL jobs delete the processed objects or data from Amazon S3 after each run.

Answer: B

Explanation:

The best solution is to enable job bookmarks on the AWS Glue jobs. AWS Glue job bookmarks provide a built-in mechanism for tracking the state of data that has already been processed during a previous run. This feature allows Glue jobs to process only the new or updated data in subsequent runs, significantly reducing processing time and costs for incremental ETL operations.

Option B is superior to option C because it eliminates the need for developers to write and maintain custom logic for tracking processed objects. Creating and managing such logic is more complex and error-prone compared to using the built-in bookmark functionality.

Option A, using DataFrames instead of DynamicFrames, does not directly address the incremental processing issue. While DataFrames are useful, they don't inherently track which data has been processed.

Option D, deleting processed data, is generally not a good practice. It can lead to data loss if the ETL process fails or if the data is needed for other purposes in the future. Additionally, deleting objects in S3 can be a time-consuming and costly operation.

Job bookmarks automatically manage the state information, allowing the Glue job to determine which files or records were processed in the last run and only process the newly added or modified ones. By enabling job bookmarks, developers can leverage AWS Glue's native capabilities for incremental data processing, minimizing coding effort and ensuring efficient ETL operations.See the documentation for further details: https://docs.aws.amazon.com/glue/latest/dg/monitor-continuations.html and https://aws.amazon.com/blogs/big-data/building-an-etl-pipeline-with-aws-glue-for-incremental-data-loads-using-job-bookmarks/

Question: 41

A mortgage company has a microservice for accepting payments. This microservice uses the Amazon DynamoDB encryption client with AWS KMS managed keys to encrypt the sensitive data before writing the data to DynamoDB.
The finance team should be able to load this data into Amazon Redshift and aggregate the values within the sensitive fields. The Amazon Redshift cluster is shared with other data analysts from different business units. Which steps should a data analyst take to accomplish this task efficiently and securely?

A.Create an AWS Lambda function to process the DynamoDB stream. Decrypt the sensitive data using the same KMS key. Save the output to a restricted S3 bucket for the finance team. Create a finance table in Amazon Redshift that is accessible to the finance team only. Use the COPY command to load the data from Amazon S3 to the finance table.

B.Create an AWS Lambda function to process the DynamoDB stream. Save the output to a restricted S3 bucket for the finance team. Create a finance table in Amazon Redshift that is accessible to the finance team only. Use the COPY command with the IAM role that has access to the KMS key to load the data from S3 to the finance table.

C.Create an Amazon EMR cluster with an EMR_EC2_DefaultRole role that has access to the KMS key. Create Apache Hive tables that reference the data stored in DynamoDB and the finance table in Amazon Redshift. In Hive, select the data from DynamoDB and then insert the output to the finance table in Amazon Redshift.

D.Create an Amazon EMR cluster. Create Apache Hive tables that reference the data stored in DynamoDB. Insert the output to the restricted Amazon S3 bucket for the finance team. Use the COPY command with the IAM role that has access to the KMS key to load the data from Amazon S3 to the finance table in Amazon Redshift.

Answer: B

Explanation:

The correct answer is B because it prioritizes security and efficiency while adhering to best practices for data access and management. Here's a detailed justification:

DynamoDB Streams and Lambda: Using a Lambda function triggered by DynamoDB Streams provides a real-time or near real-time mechanism to capture data changes. This avoids the need for batch processing or polling the DynamoDB table directly.

Restricted S3 Bucket: Storing the output in a restricted S3 bucket ensures that only the finance team has access to the decrypted data, aligning with the requirement for data access control.

Redshift Finance Table with Restricted Access: Creating a separate finance table in Redshift and limiting access to the finance team fulfills the requirement for segregating sensitive data and preventing unauthorized access from other business units.

COPY Command with IAM Role and KMS Key Access: The COPY command is the most efficient way to load data into Redshift from S3. Importantly, granting the Redshift cluster's IAM role access to the KMS key allows Redshift to decrypt the data directly from S3 during the COPY process. This avoids the need to decrypt the data in Lambda, reducing complexity and enhancing security by minimizing the exposure of the decryption key.

Option A is less desirable because decrypting the data in the Lambda function exposes the KMS key within the Lambda environment, which can be a security concern. The approach in Option B offloads the decryption process to Redshift, where it can be handled more securely using IAM roles and KMS key permissions.

Options C and D involve EMR and Hive, which are more complex and potentially overkill for this relatively simple data transformation and loading task. EMR clusters take longer to provision and manage, adding unnecessary overhead. Hive, while useful for data querying and transformation, introduces unnecessary complexity when a simple COPY command with KMS access suffices. Additionally, option D incorrectly implies that the EMR cluster could have direct access to DynamoDB, which isn't the best architectural pattern in this case.

Supporting Links:

Amazon DynamoDB Streams:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html
AWS Lambda:https://aws.amazon.com/lambda/
Amazon S3 Access Control:https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-overview.html
Amazon Redshift COPY command:https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html **AWS KMS:**https://aws.amazon.com/kms/
DynamoDB Encryption Client:https://github.com/aws/aws-dynamodb-encryption-java

Question: 42

A company is building a data lake and needs to ingest data from a relational database that has time-series data. The company wants to use managed services to accomplish this. The process needs to be scheduled daily and bring incremental data only from the source into Amazon S3.
What is the MOST cost-effective approach to meet these requirements?

A.Use AWS Glue to connect to the data source using JDBC Drivers. Ingest incremental records only using job bookmarks.

B.Use AWS Glue to connect to the data source using JDBC Drivers. Store the last updated key in an Amazon DynamoDB table and ingest the data using the updated key as a filter.

C.Use AWS Glue to connect to the data source using JDBC Drivers and ingest the entire dataset. Use appropriate Apache Spark libraries to compare the dataset, and find the delta.

D.Use AWS Glue to connect to the data source using JDBC Drivers and ingest the full data. Use AWS DataSync to ensure the delta only is written into Amazon S3.

Answer: A

Explanation:

The most cost-effective approach is to use AWS Glue with job bookmarks. Option A efficiently addresses the requirement for scheduled, incremental data ingestion from a relational database into Amazon S3. AWS Glue is a fully managed ETL service, minimizing operational overhead. JDBC drivers enable connectivity to the relational database. Job bookmarks are specifically designed to track the progress of Glue jobs, allowing them to only extract new or updated data since the last run. This incremental approach minimizes data transfer and processing costs, as opposed to repeatedly ingesting the entire dataset.

Option B, while functional, introduces additional complexity and cost by involving DynamoDB for tracking the last updated key. DynamoDB, though fast, incurs costs for storage and read/write capacity.

Option C involves ingesting the entire dataset daily and using Spark to find the delta. This is highly inefficient and expensive due to the redundant data transfer and processing involved.

Option D also suggests ingesting the full data each time, which is wasteful. AWS DataSync is designed for larger-scale data migrations and replication, and is not the best fit for this specific incremental data ingestion requirement, especially considering its cost implications compared to Glue bookmarks.

Therefore, using Glue's built-in job bookmark feature provides the most cost-effective and streamlined solution for scheduled, incremental data ingestion. It avoids unnecessary data transfer and the added costs and complexity of managing a separate DynamoDB table or using more heavyweight tools like DataSync for a relatively simple incremental load scenario.

Refer to AWS documentation for further details on:

AWS Glue Job Bookmarks:https://docs.aws.amazon.com/glue/latest/dg/monitor-continuations.html **AWS Glue:**https://aws.amazon.com/glue/

Question: 43

An Amazon Redshift database contains sensitive user data. Logging is necessary to meet compliance requirements. The logs must contain database authentication attempts, connections, and disconnections. The logs must also contain each query run against the database and record which database user ran each query. Which steps will create the required logs?

A.Enable Amazon Redshift Enhanced VPC Routing. Enable VPC Flow Logs to monitor traffic.

B.Allow access to the Amazon Redshift database using AWS IAM only. Log access using AWS CloudTrail.

C.Enable audit logging for Amazon Redshift using the AWS Management Console or the AWS CLI.

D.Enable and download audit reports from AWS Artifact.

Answer: C

Explanation:

The correct answer is C: Enable audit logging for Amazon Redshift using the AWS Management Console or the AWS CLI. This is because Amazon Redshift has a built-in auditing feature specifically designed to capture the details required for compliance, such as authentication attempts, connections, disconnections, and query execution details including the user who ran the query.

Option A is incorrect because Enhanced VPC Routing and VPC Flow Logs primarily capture network traffic information at the IP address level, which is not sufficient for the detailed database activity logging required for compliance. VPC Flow Logs do not capture the specific queries being executed or the user identity associated with those queries.

Option B is incorrect because while using IAM for Redshift access is a good security practice, CloudTrail logs IAM actions, not the specific SQL queries run within Redshift or database connection events. CloudTrail is useful for tracking who made changes to the Redshift cluster itself (e.g., scaling, configuration changes) but not user activity within the database.

Option D is incorrect because AWS Artifact provides on-demand access to AWS' compliance reports. While valuable for understanding AWS' overall security posture, Artifact does not provide audit logs generated by your Redshift instance related to user activity. It is a source of compliance documentation, not an auditing tool for your specific Redshift usage.

Therefore, directly enabling audit logging within Amazon Redshift is the most appropriate solution. The logs capture user login attempts, connection details, and every SQL query executed, along with the user who executed each query. These logs can be stored in Amazon S3 for long-term retention and analysis, meeting the specified compliance requirements.

Reference: https://docs.aws.amazon.com/redshift/latest/mgmt/db-auditing.html

Question: 44

A company that monitors weather conditions from remote construction sites is setting up a solution to collect temperature data from the following two weather stations.
➾ Station A, which has 10 sensors
➾ Station B, which has five sensors
These weather stations were placed by onsite subject-matter experts.
Each sensor has a unique ID. The data collected from each sensor will be collected using Amazon Kinesis Data Streams.
Based on the total incoming and outgoing data throughput, a single Amazon Kinesis data stream with two shards is created.
Two partition keys are created based on the station names. During testing, there is a bottleneck on data coming from Station A, but not from Station B. Upon review, it is confirmed that the total stream throughput is still less than the allocated Kinesis Data Streams throughput.
How can this bottleneck be resolved without increasing the overall cost and complexity of the solution, while retaining the data collection quality requirements?

A.Increase the number of shards in Kinesis Data Streams to increase the level of parallelism.

B.Create a separate Kinesis data stream for Station A with two shards, and stream Station A sensor data to the new stream.

C.Modify the partition key to use the sensor ID instead of the station name.

D.Reduce the number of sensors in Station A from 10 to 5 sensors.

Answer: C

Explanation:

The correct answer is C: Modify the partition key to use the sensor ID instead of the station name.

The problem lies in the uneven distribution of data across the Kinesis Data Streams shards. Kinesis Data Streams uses the partition key to determine which shard a record is written to. With station name as the partition key, all data from Station A goes to one shard, and all data from Station B goes to the other. Even though the total stream throughput is within limits, the single shard handling Station A's data is overloaded, causing a bottleneck.

Option A, increasing the number of shards, would indeed increase parallelism. However, without changing the partition key, all data from Station A would still hash to the same shard, rendering the additional shards ineffective in resolving Station A's bottleneck. Therefore, it does not address the root cause, which is poor data distribution.

Option B, creating a separate Kinesis Data Stream for Station A, would solve the immediate bottleneck issue. However, it increases the cost and complexity of the solution by requiring separate stream management and potential cross-stream processing. The question specifically asks for a solution without increasing cost and complexity.

Option D, reducing the number of sensors in Station A, is unacceptable as it violates the data collection quality requirements. The solution must maintain the current sensor count and the data they provide.

By changing the partition key to the sensor ID, the data is now distributed across shards based on individual sensors, not stations. Since each sensor has a unique ID, this will lead to a more uniform distribution of data across the two shards. This resolves the bottleneck for Station A because its data is no longer concentrated on a single shard. Furthermore, using sensor IDs does not increase the cost or complexity, and maintains the quality requirements.

Therefore, modifying the partition key to use the sensor ID is the optimal solution because it addresses the bottleneck by improving data distribution without increasing costs, complexity, or compromising data collection quality.

Further research:

Amazon Kinesis Data Streams Partitioning:https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html
Choosing a Kinesis Data Streams Partition Key:https://aws.amazon.com/blogs/big-data/best-practices-for-scaling-amazon-kinesis-data-streams-to-optimize-performance-and-cost/

Question: 45

Once a month, a company receives a 100 MB .csv file compressed with gzip. The file contains 50,000 property listing records and is stored in Amazon S3 Glacier.
The company needs its data analyst to query a subset of the data for a specific vendor. What is the most cost-effective solution?

A.Load the data into Amazon S3 and query it with Amazon S3 Select.
B.Query the data from Amazon S3 Glacier directly with Amazon Glacier Select.
C.Load the data to Amazon S3 and query it with Amazon Athena.
D.Load the data to Amazon S3 and query it with Amazon Redshift Spectrum.

Answer: A

Explanation:

The most cost-effective solution is A: Load the data into Amazon S3 and query it with Amazon S3 Select. Here's why:

Cost Considerations: The goal is cost-effectiveness. S3 Glacier is designed for archival storage, and retrieving data from it incurs significant retrieval costs, particularly for frequent access. Options B, C, and D all involve retrieving data from Glacier.

S3 Select Efficiency: Amazon S3 Select allows querying specific data within an S3 object without retrieving the entire file. This minimizes the amount of data transferred and processed, significantly reducing costs.

Alternatives Inefficiency:

Amazon Glacier Select: While it seems similar, Glacier Select is generally more expensive than using S3

Standard and then S3 Select because of Glacier's retrieval costs.

Amazon Athena: While Athena is a serverless query service, it incurs costs based on the amount of data scanned. Retrieving the data from Glacier and then using Athena would be cost-prohibitive for monthly querying.

Amazon Redshift Spectrum: Redshift Spectrum is best suited for querying large datasets stored in S3 data lakes. For a single, relatively small 100MB file, the overhead and cost of Redshift Spectrum are not justified.

Suitability for the Task: The scenario describes querying a subset of the data for a specific vendor. S3 Select is perfectly designed for this kind of targeted data extraction. Moving the .csv file to standard S3 allows for efficient querying using S3 Select.

Frequency of Access: The file is accessed monthly. This frequency justifies the cost of transferring the data from Glacier to S3 Standard.

Therefore, loading the data into S3 Standard and then querying it using S3 Select strikes the right balance between performance and cost for the given requirements.

Supporting Documentation:

Amazon S3 Select:https://aws.amazon.com/s3/select/
Amazon S3 Storage Classes:https://aws.amazon.com/s3/storage-classes/

Question: 46

A retail company is building its data warehouse solution using Amazon Redshift. As a part of that effort, the company is loading hundreds of files into the fact table created in its Amazon Redshift cluster. The company wants the solution to achieve the highest throughput and optimally use cluster resources when loading data into the company's fact table.
How should the company meet these requirements?

A.Use multiple COPY commands to load the data into the Amazon Redshift cluster.

B.Use S3DistCp to load multiple files into the Hadoop Distributed File System (HDFS) and use an HDFS connector to ingest the data into the Amazon Redshift cluster.

C.Use LOAD commands equal to the number of Amazon Redshift cluster nodes and load the data in parallel into each node.

D.Use a single COPY command to load the data into the Amazon Redshift cluster.

Answer: D

Explanation:

The best approach to load hundreds of files into an Amazon Redshift fact table with maximum throughput and optimal resource utilization is to use a single COPY command. The COPY command leverages Redshift's massively parallel processing (MPP) architecture to efficiently distribute the data load across all compute nodes in the cluster. Loading data in parallel is beneficial, but it's best handled internally by Redshift. A single COPY command allows Redshift to orchestrate the parallel loading effectively by dividing the input data among the available compute nodes and using all the computing resources in parallel.

Option A (multiple COPY commands) would create overhead associated with initiating and managing multiple commands concurrently. This can lead to resource contention and slower overall performance. Option B, which involves using S3DistCp and HDFS, is unnecessarily complex for loading data directly into Redshift from S3 and creates more steps in the ETL pipeline that are not needed. Redshift is designed to directly ingest data from Amazon S3. Option C (LOAD commands) is not a valid or recommended way to load data into Redshift.

A single COPY command lets Redshift internally optimize the parallelism and distribution of the data loading process, therefore maximizing throughput and optimizing the use of cluster resources, because the command will distribute the work to all the compute nodes within the Redshift cluster.

Therefore, the COPY command is the simplest and most efficient means of loading data from S3 to Redshift.

Refer to the AWS documentation on the COPY command for further details:
https://docs.aws.amazon.com/redshift/latest/dg/t_loading-tables-from-s3.html

Question: 47

A data analyst is designing a solution to interactively query datasets with SQL using a JDBC connection. Users will join data stored in Amazon S3 in Apache ORC format with data stored in Amazon OpenSearch Service (Amazon Elasticsearch Service) and Amazon Aurora MySQL.
Which solution will provide the MOST up-to-date results?

A.Use AWS Glue jobs to ETL data from Amazon ES and Aurora MySQL to Amazon S3. Query the data with Amazon Athena.

B.Use Amazon DMS to stream data from Amazon ES and Aurora MySQL to Amazon Redshift. Query the data with Amazon Redshift.

C.Query all the datasets in place with Apache Spark SQL running on an AWS Glue developer endpoint.

D.Query all the datasets in place with Apache Presto running on Amazon EMR.

Answer: D

Explanation:

The correct answer is D because Apache Presto on Amazon EMR is designed for querying data across multiple heterogeneous data sources, including S3, OpenSearch Service (formerly Elasticsearch Service), and Aurora MySQL, using standard SQL. This allows for querying the data in place without the need for ETL, ensuring the most up-to-date results. Presto supports JDBC connections, making it suitable for interactive SQL queries.

Option A is less ideal because it involves ETLing data into S3 using Glue. ETL processes introduce latency, meaning the data queried might not be the most current. While Athena is suitable for querying data in S3, the ETL process needed for the other data sources makes it less favorable for near real-time querying.

Option B uses DMS to stream data to Redshift. While Redshift is a powerful data warehouse, the streaming process introduces latency, and it requires data to be ingested into Redshift. This doesn't allow for querying the original data sources directly in place.

Option C uses Spark SQL on a Glue developer endpoint. While Spark SQL can query multiple data sources, setting up and managing dependencies, especially for JDBC connections to diverse data sources, can be complex. The Glue developer endpoint might not be optimized for the kind of concurrent, interactive querying needed in the scenario, and can become costly.

Therefore, Presto on EMR offers the best solution for querying heterogeneous data sources in place with low latency using SQL and providing JDBC connectivity for interactive queries, making it the most appropriate choice to ensure up-to-date results.**Supporting Documentation:**

Amazon EMR:https://aws.amazon.com/emr/
Apache Presto:https://prestodb.io/
Amazon Athena:https://aws.amazon.com/athena/
AWS Glue:https://aws.amazon.com/glue/
Amazon Redshift:https://aws.amazon.com/redshift/
Amazon DMS:https://aws.amazon.com/dms/

Question: 48

A company developed a new elections reporting website that uses Amazon Kinesis Data Firehose to deliver full logs from AWS WAF to an Amazon S3 bucket.
The company is now seeking a low-cost option to perform this infrequent data analysis with visualizations of logs in a way that requires minimal development effort.
Which solution meets these requirements?

A.Use an AWS Glue crawler to create and update a table in the Glue data catalog from the logs. Use Athena to perform ad-hoc analyses and use Amazon QuickSight to develop data visualizations.

B.Create a second Kinesis Data Firehose delivery stream to deliver the log files to Amazon OpenSearch Service (Amazon Elasticsearch Service). Use Amazon ES to perform text-based searches of the logs for ad-hoc analyses and use OpenSearch Dashboards (Kibana) for data visualizations.

C.Create an AWS Lambda function to convert the logs into .csv format. Then add the function to the Kinesis Data Firehose transformation configuration. Use Amazon Redshift to perform ad-hoc analyses of the logs using SQL queries and use Amazon QuickSight to develop data visualizations.

D.Create an Amazon EMR cluster and use Amazon S3 as the data source. Create an Apache Spark job to perform ad-hoc analyses and use Amazon QuickSight to develop data visualizations.

Answer: A

Explanation:

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

Why A is the best solution:

Low-Cost and Minimal Development Effort: AWS Glue, Athena, and QuickSight are all serverless services, meaning the company doesn't need to manage any infrastructure. This minimizes operational overhead and development effort. Pay-as-you-go pricing makes it cost-effective for infrequent analysis.

AWS Glue for Data Cataloging: AWS Glue crawler can automatically infer the schema of the logs stored in S3 and create a table in the AWS Glue Data Catalog. This catalog acts as a central metadata repository, enabling Athena to query the data. (https://aws.amazon.com/glue/)

Athena for Ad-hoc Analysis: Athena is a serverless query service that allows direct querying of data in S3 using standard SQL. This removes the need to load data into a database. The ad-hoc nature suits infrequent analysis. (https://aws.amazon.com/athena/)

QuickSight for Visualizations: Amazon QuickSight is a business intelligence (BI) service that integrates with Athena. It allows creating interactive dashboards and visualizations based on the results of Athena queries, without requiring significant coding or data manipulation. (https://aws.amazon.com/quicksight/)

Log Format Agnostic: While not explicitly mentioned, the logs are likely in a structured format like JSON given they are WAF logs. Glue and Athena can readily handle JSON data.

Why other options are less suitable:

B (OpenSearch Service): While OpenSearch is good for searching logs, it's more suited for real-time or near real-time analysis and monitoring. It is also more complex and expensive to set up and manage than Glue/Athena.

C (Lambda & Redshift): Converting logs to CSV with Lambda adds complexity. Redshift, while powerful, is a data warehouse intended for structured data and larger datasets. It's overkill and more expensive for infrequent ad-hoc analysis of WAF logs.

D (EMR & Spark): EMR and Spark are more appropriate for large-scale data processing and analysis, often requiring custom code. For ad-hoc analysis of WAF logs, this solution is significantly more complex and costly than Glue/Athena. It demands more development effort for Spark job creation and cluster management.

In summary, option A provides the most cost-effective, low-development-effort solution for infrequent analysis and visualization of WAF logs.

Question: 49

A large company has a central data lake to run analytics across different departments. Each department uses a separate AWS account and stores its data in an
Amazon S3 bucket in that account. Each AWS account uses the AWS Glue Data Catalog as its data catalog. There are different data lake access requirements based on roles. Associate analysts should only have read access to their departmental data. Senior data analysts can have access in multiple departments including theirs, but for a subset of columns only.
Which solution achieves these required access patterns to minimize costs and administrative tasks?

A.Consolidate all AWS accounts into one account. Create different S3 buckets for each department and move all the data from every account to the central data lake account. Migrate the individual data catalogs into a central data catalog and apply fine-grained permissions to give to each user the required access to tables and databases in AWS Glue and Amazon S3.

B.Keep the account structure and the individual AWS Glue catalogs on each account. Add a central data lake account and use AWS Glue to catalog data from various accounts. Configure cross-account access for AWS Glue crawlers to scan the data in each departmental S3 bucket to identify the schema and populate the catalog. Add the senior data analysts into the central account and apply highly detailed access controls in the Data Catalog and Amazon S3.

C.Set up an individual AWS account for the central data lake. Use AWS Lake Formation to catalog the cross-account locations. On each individual S3 bucket, modify the bucket policy to grant S3 permissions to the Lake Formation service-linked role. Use Lake Formation permissions to add fine-grained access controls to allow senior analysts to view specific tables and columns.

D.Set up an individual AWS account for the central data lake and configure a central S3 bucket. Use an AWS Lake Formation blueprint to move the data from the various buckets into the central S3 bucket. On each individual bucket, modify the bucket policy to grant S3 permissions to the Lake Formation service-linked role. Use Lake Formation permissions to add fine-grained access controls for both associate and senior analysts to view specific tables and columns.

Answer: C

Explanation:

The best solution is C because it leverages AWS Lake Formation's cross-account data sharing and fine-grained access control capabilities, minimizing administrative overhead and costs. Here's a breakdown:

Centralized Data Lake Account: Having a dedicated account for the data lake ensures a clear separation of concerns and centralized governance.

AWS Lake Formation: Lake Formation simplifies the process of setting up, securing, and managing data lakes. It integrates with AWS Glue for cataloging data and provides fine-grained access control policies.
https://aws.amazon.com/lake-formation/

Cross-Account Access: Lake Formation's ability to catalog cross-account locations eliminates the need to move data, reducing storage costs and complexity. It allows the data to reside in the department's S3 buckets while making metadata available in the central data lake. https://docs.aws.amazon.com/lake-formation/latest/dg/granting-cross-account-permissions.html

S3 Bucket Policies and Service-Linked Roles: Modifying the S3 bucket policies to grant permissions to the Lake Formation service-linked role enables Lake Formation to access the data in the departmental S3 buckets.

Fine-Grained Access Control: Lake Formation enables precise control over data access at the table and column level. This is essential for providing associate analysts with read access to their departmental data and senior analysts with access to specific columns across multiple departments.
https://docs.aws.amazon.com/lake-formation/latest/dg/security-data-access.html

Option A is less desirable because consolidating all accounts introduces potential organizational complexities and may not be feasible due to department autonomy requirements. Option B would make the administration of fine-grained access controls and maintaining consistency across accounts cumbersome. Option D involves moving data using Lake Formation blueprints, which adds complexity and increases storage costs. Lake Formation blueprint isn't the primary use case for cross-account scenario.

Question: 50

A company wants to improve user satisfaction for its smart home system by adding more features to its recommendation engine. Each sensor asynchronously pushes its nested JSON data into Amazon Kinesis Data Streams using the Kinesis Producer Library (KPL) in Java. Statistics from a set of failed sensors showed that, when a sensor is malfunctioning, its recorded data is not always sent to the cloud.
The company needs a solution that offers near-real-time analytics on the data from the most updated sensors. Which solution enables the company to meet these requirements?

A.Set the RecordMaxBufferedTime property of the KPL to "1’ˆג" to disable the buffering on the sensor side. Use Kinesis Data Analytics to enrich the data based on a company-developed anomaly detection SQL script. Push the enriched data to a fleet of Kinesis data streams and enable the data transformation feature to flatten the JSON file. Instantiate a dense storage Amazon Redshift cluster and use it as the destination for the Kinesis Data Firehose delivery stream.

B.Update the sensors code to use the PutRecord/PutRecords call from the Kinesis Data Streams API with the AWS SDK for Java. Use Kinesis Data Analytics to enrich the data based on a company-developed anomaly detection SQL script. Direct the output of KDA application to a Kinesis Data Firehose delivery stream, enable the data transformation feature to flatten the JSON file, and set the Kinesis Data Firehose destination to an Amazon OpenSearch Service (Amazon Elasticsearch Service) cluster.

C.Set the RecordMaxBufferedTime property of the KPL to "0" to disable the buffering on the sensor side. Connect for each stream a dedicated Kinesis Data Firehose delivery stream and enable the data transformation feature to flatten the JSON file before sending it to an Amazon S3 bucket. Load the S3 data into an Amazon Redshift cluster.

D.Update the sensors code to use the PutRecord/PutRecords call from the Kinesis Data Streams API with the AWS SDK for Java. Use AWS Glue to fetch and process data from the stream using the Kinesis Client Library (KCL). Instantiate an Amazon Elasticsearch Service cluster and use AWS Lambda to directly push data into it.

Answer: B

Explanation:

Here's a detailed justification for why option B is the most suitable solution:

Option B addresses the key requirements: near-real-time analytics, handling potential data loss from malfunctioning sensors, and flattening nested JSON data. Using PutRecord/PutRecords directly with the Kinesis Data Streams API in the AWS SDK for Java ensures that data is sent immediately without buffering on the sensor side. This reduces the risk of losing data if a sensor fails, addressing the concern of data being "not always sent to the cloud." The Kinesis Producer Library (KPL), while useful for high throughput, introduces buffering, which can lead to delays in data delivery, conflicting with the near-real-time requirement. Directly using the API offers more control and immediate sending.

Kinesis Data Analytics (KDA) is the correct service for near-real-time analytics on streaming data. The prompt specifies using a "company-developed anomaly detection SQL script," which fits perfectly with KDA's SQL-based processing capabilities. KDA can process the raw stream data and identify anomalies based on the specified script. The output from KDA, containing enriched data (with anomaly flags or scores), is then directed to Kinesis Data Firehose.

Kinesis Data Firehose is used to deliver the transformed and enriched data to a destination. Enabling the data transformation feature within Firehose allows for flattening the nested JSON data. This transformation can be done using AWS Lambda. Finally, Amazon OpenSearch Service (successor to Elasticsearch Service) is a good choice for storing and analyzing the transformed data because it's designed for search and analytics on large datasets. It allows for flexible querying and visualization of the sensor data and identified anomalies.

Option A's attempt to disable buffering with RecordMaxBufferedTime set to a high value ("1′ˆג") is incorrect; a value of 0 is required to effectively disable buffering. Furthermore, using Redshift directly from Firehose for streaming data is generally not ideal, as Redshift is better suited for batch processing. Option C uses S3 and then Redshift which creates latency and doesn't provide the required near-real-time analysis. Option D inappropriately uses AWS Glue with KCL to process streaming data. Glue is better suited for batch ETL operations and KCL requires setting up and managing workers. This solution also unnecessarily involves Lambda for pushing data into OpenSearch Service, which Kinesis Data Firehose directly supports.

Therefore, option B is the most complete and efficient solution.

Relevant Links:

Kinesis Data Streams API
Kinesis Data Analytics
Kinesis Data Firehose
Amazon OpenSearch Service