Amazon

(AWS Certified Big Data - Specialty)

exam

Total: **85 Questions**

Link:

Question: 1

A data engineer in a manufacturing company is designing a data processing platform that receives a large volume of unstructured data. The data engineer must populate a well-structured star schema in Amazon Redshift.
What is the most efficient architecture strategy for this purpose?

A. Transform the unstructured data using Amazon EMR and generate CSV data. COPY the CSV data into the analysis schema within Redshift.

B. Load the unstructured data into Redshift, and use string parsing functions to extract structured data for inserting into the analysis schema.

C. When the data is saved to Amazon S3, use S3 Event Notifications and AWS Lambda to transform the file contents. Insert the data into the analysis schema on Redshift.

D. Normalize the data using an AWS Marketplace ETL tool, persist the results to Amazon S3, and use AWS Lambda to INSERT the data into Redshift.

Answer: A

Explanation:

The most efficient architecture is **A. Transform the unstructured data using Amazon EMR and generate CSV data. COPY the CSV data into the analysis schema within Redshift.**

Here's the justification:

Scalability and Processing Power: Amazon EMR is designed for processing large volumes of data using distributed computing frameworks like Spark or Hadoop. This makes it ideal for transforming unstructured data where parsing and manipulation are required at scale.

Cost-Effectiveness: EMR clusters can be spun up on-demand and scaled based on the processing needs, optimizing costs. Options B, C, and D might lead to higher operational overhead or unnecessary resource consumption for data transformation.

Redshift's COPY Command Optimization: Redshift's COPY command is highly optimized for bulk loading data from Amazon S3. Transforming the data into CSV format first aligns with this optimization, providing the fastest ingestion path.

Avoiding Bottlenecks: Option B (loading directly into Redshift and using string parsing) can significantly strain Redshift's compute resources, leading to performance bottlenecks, particularly with large volumes of unstructured data. Redshift is designed for analysis, not heavy transformation.

Minimizing Complexity: Option C (Lambda) might face concurrency limitations or cold starts with large data volumes. Option D (Marketplace ETL tool and Lambda) introduces additional complexity and potential vendor lock-in without necessarily providing a performance advantage over a well-configured EMR job. Furthermore, using Lambda to insert data row by row is extremely inefficient.

S3 as a Staging Area: Using Amazon S3 as a staging area after transformation in EMR allows for data validation and auditing before loading into Redshift.

In summary, offloading the intensive unstructured data transformation to a specialized service like Amazon EMR and leveraging Redshift's optimized COPY command provides the most scalable, cost-effective, and performant architecture for populating a star schema in Redshift from a large volume of unstructured data.

Amazon EMR:https://aws.amazon.com/emr/
Amazon Redshift COPY command:https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html

Question: 2

A new algorithm has been written in Python to identify SPAM e-mails. The algorithm analyzes the free text contained within a sample set of 1 million e-mails stored on Amazon S3. The algorithm must be scaled across a production dataset of 5 PB, which also resides in Amazon S3 storage.
Which AWS service strategy is best for this use case?

A. Copy the data into Amazon ElastiCache to perform text analysis on the in-memory data and export the results of the model into Amazon Machine Learning.

B. Use Amazon EMR to parallelize the text analysis tasks across the cluster using a streaming program step.

C. Use Amazon Elasticsearch Service to store the text and then use the Python Elasticsearch Client to run analysis against the text index.

D. Initiate a Python job from AWS Data Pipeline to run directly against the Amazon S3 text files.

Answer: B

Explanation:

Here's a detailed justification for why option B, using Amazon EMR, is the best AWS service strategy for analyzing the 5 PB of email data for SPAM identification:

The problem requires analyzing a massive (5 PB) dataset of emails stored in Amazon S3. Option A (ElastiCache) is unsuitable because ElastiCache is an in-memory caching service designed for fast data retrieval, not for large-scale data processing. Copying 5 PB of data into ElastiCache is impractical and financially infeasible. Additionally, Amazon Machine Learning is deprecated, making this option outdated. Option C (Elasticsearch) is better suited for searching and indexing, not for the complex text analysis required for SPAM detection across such a large volume of data. Storing 5 PB of data in Elasticsearch and performing analysis through the Python client would be expensive and inefficient. Option D (AWS Data Pipeline) is designed for data orchestration and movement, not for intensive computation. While it can trigger Python jobs, Data Pipeline itself doesn't provide the compute resources needed for analyzing 5 PB of data efficiently.

Amazon EMR, on the other hand, is specifically designed for processing large datasets using distributed computing frameworks like Hadoop and Spark. By using EMR, we can parallelize the Python-based SPAM detection algorithm across a cluster of EC2 instances. Streaming programs in EMR allow us to process data sequentially without loading the entire dataset into memory at once, which is critical when dealing with petabytes of data. A streaming program can read data from S3, analyze it in chunks, and then write the results back to S3 or another data store. This approach provides scalability, fault tolerance, and cost-effectiveness for large-scale data analysis tasks. EMR also simplifies the management of the underlying infrastructure, allowing data scientists and engineers to focus on developing and deploying the SPAM detection algorithm.https://aws.amazon.com/emr/https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html

Question: 3

A data engineer chooses Amazon DynamoDB as a data store for a regulated application. This application must be submitted to regulators for review. The data engineer needs to provide a control framework that lists the security controls from the process to follow to add new users down to the physical controls of the data center, including items like security guards and cameras.
How should this control mapping be achieved using AWS?

A. Request AWS third-party audit reports and/or the AWS quality addendum and map the AWS responsibilities to the controls that must be provided.

B. Request data center Temporary Auditor access to an AWS data center to verify the control mapping.

C. Request relevant SLAs and security guidelines for Amazon DynamoDB and define these guidelines within the applications architecture to map to the control framework.

D. Request Amazon DynamoDB system architecture designs to determine how to map the AWS responsibilities to the control that must be provided.

Answer: A

Explanation:

The correct answer is A because it leverages AWS's existing compliance documentation to fulfill the data engineer's requirement. A control framework for a regulated application requires outlining security measures at various levels, including user access management and physical security. AWS operates under a shared responsibility model. The customer is responsible for security in the cloud, while AWS is responsible for security of the cloud.

Option A directly addresses this shared responsibility. AWS undergoes regular third-party audits (like SOC 2, PCI DSS, ISO 27001) and makes these reports available to customers. These reports detail the controls AWS implements to secure its infrastructure, including data centers, network, and services like DynamoDB. By reviewing these reports, the data engineer can map AWS's responsibilities to the controls that must be demonstrated to regulators. The AWS Quality Addendum provides further clarification on AWS's compliance commitments.

Option B is incorrect because AWS does not grant direct physical access to its data centers to customers or auditors for security reasons. This would pose a security risk and is counter to AWS's operational model.

Option C is partially correct but incomplete. SLAs primarily focus on service availability and performance. Security guidelines provide some information, but they don't offer the comprehensive control mapping provided by audit reports.

Option D is incorrect because architectural designs are not the primary source for control information. While architectural diagrams illustrate the system's structure, they don't detail the specific security controls and processes implemented by AWS. Audit reports offer the required level of detail and assurance.

In summary, option A provides the most efficient and secure method for mapping AWS's security controls to the required control framework for the regulated application by leveraging existing compliance documentation. This approach aligns with the shared responsibility model and avoids violating AWS security protocols.

For further research, you can explore the following resources:

AWS Compliance:https://aws.amazon.com/compliance/
AWS Security Best Practices:https://aws.amazon.com/security/
AWS Shared Responsibility Model:https://aws.amazon.com/compliance/shared-responsibility-model/ **SOC 2 Reports:**https://aws.amazon.com/compliance/soc/

Question: 4

An administrator needs to design a distribution strategy for a star schema in a Redshift cluster. The administrator needs to determine the optimal distribution style for the tables in the Redshift schema.
In which three circumstances would choosing Key-based distribution be most appropriate? (Select three.)

A. When the administrator needs to optimize a large, slowly changing dimension table.

B. When the administrator needs to reduce cross-node traffic.

C. When the administrator needs to optimize the fact table for parity with the number of slices.

D. When the administrator needs to balance data distribution and collocation data.

E. When the administrator needs to take advantage of data locality on a local node for joins and aggregates.

Answer: BDE

Explanation:

Here's a detailed justification for why options B, D, and E are the most appropriate circumstances for using key-based distribution in an Amazon Redshift star schema:

Key distribution aims to place rows with the same key value on the same compute node. This design facilitates efficient joins.

B. When the administrator needs to reduce cross-node traffic: Key distribution minimizes data transfer across nodes because rows participating in join operations are likely to be co-located. This reduces the need for Redshift to move data between nodes to complete joins, resulting in faster query execution and lower network congestion.

D. When the administrator needs to balance data distribution and collocation data: Key distribution provides a middle ground between even data distribution (like EVEN distribution) and full replication (like ALL distribution). It aims to balance the workload across nodes based on the distribution of the chosen key while ensuring related data (based on the key) is located together. This is crucial for maintaining query performance and avoiding skewness.

E. When the administrator needs to take advantage of data locality on a local node for joins and aggregates: The core advantage of key distribution is data locality. When data related by the distribution key resides on the same node, joins and aggregations involving that key can be performed entirely locally. This significantly speeds up query execution as it eliminates the need to move data between nodes during these operations. For example, if the fact table is distributed on user_id, all facts for a specific user will reside on the same node allowing efficient user-specific aggregations.

Why the other options are less suitable:

A. When the administrator needs to optimize a large, slowly changing dimension table: ALL distribution is typically more suitable for large, slowly changing dimension tables. ALL distribution replicates the entire table to every node in the cluster, allowing for fast lookups without any data transfer.

C. When the administrator needs to optimize the fact table for parity with the number of slices: EVEN distribution is more appropriate when you want to distribute rows evenly across the compute nodes regardless of the data itself.

Authoritative Links:

Amazon Redshift Distribution Styles:
https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_a_distribution_style.html
Amazon Redshift Engineering Best Practices:https://docs.aws.amazon.com/redshift/latest/dg/best-practices.html

Question: 5

Company A operates in Country X. Company A maintains a large dataset of historical purchase orders that contains personal data of their customers in the form of full names and telephone numbers. The dataset consists of 5 text files, 1TB each. Currently the dataset resides on-premises due to legal requirements of storing personal data in-country. The research and development department needs to run a clustering algorithm on the dataset and wants to use Elastic Map Reduce service in the closest AWS region. Due to geographic distance, the minimum latency between the on-premises system and the closet AWS region is 200 ms.
Which option allows Company A to do clustering in the AWS Cloud and meet the legal requirement of maintaining personal data in-country?

A. Anonymize the personal data portions of the dataset and transfer the data files into Amazon S3 in the AWS region. Have the EMR cluster read the dataset using EMRFS.

B. Establish a Direct Connect link between the on-premises system and the AWS region to reduce latency. Have the EMR cluster read the data directly from the on-premises storage system over Direct Connect.

C. Encrypt the data files according to encryption standards of Country X and store them on AWS region in Amazon S3. Have the EMR cluster read the dataset using EMRFS.

D. Use AWS Import/Export Snowball device to securely transfer the data to the AWS region and copy the files onto an EBS volume. Have the EMR cluster read the dataset using EMRFS.

Answer: A

Explanation:

The correct answer is A because it addresses both the legal requirement of keeping personal data in-country and the need to perform clustering in AWS. Anonymizing the data removes the personally identifiable information (PII), thus satisfying the legal constraint. Once anonymized, the data can be safely transferred to Amazon S3 in the nearest AWS region without violating any data residency laws. EMRFS (EMR File System) allows the EMR cluster to efficiently read and process the data stored in S3.

Option B is not ideal because while Direct Connect reduces latency, it doesn't address the core issue of personal data needing to remain in-country. Transferring data to AWS, even with low latency, still violates the legal requirement. Furthermore, reading 5TB of data across a network connection for intensive computation is highly inefficient and could lead to prolonged processing times, even with Direct Connect.

Option C, while encrypting data is a good security practice, it doesn't fulfill the requirement of keeping personal data in-country. Encryption only protects the data in transit and at rest, but the data still resides outside the country, violating the legal requirement.

Option D is problematic because copying the data onto an EBS volume doesn't leverage the scalability and cost-effectiveness of S3 with EMR. While Snowball facilitates secure data transfer, the key issue of data residency isn't addressed unless the data is anonymized first. EBS volumes are also tied to specific EC2 instances and are less suitable for large-scale data storage compared to S3. Moreover, using EBS is less aligned with best practices for EMR, which are optimized for S3 usage via EMRFS.

In summary, anonymization allows for compliant cloud processing by removing the sensitive attributes, enabling the use of scalable AWS services like S3 and EMR without legal repercussions. Other options either fail to address the legal requirements or propose less efficient and scalable solutions.

Relevant Links:

Amazon S3: https://aws.amazon.com/s3/
Amazon EMR: https://aws.amazon.com/emr/
Data anonymization techniques: https://en.wikipedia.org/wiki/Data_anonymization

Question: 6

An administrator needs to design a strategy for the schema in a Redshift cluster. The administrator needs to determine the optimal distribution style for the tables in the Redshift schema.
In which two circumstances would choosing EVEN distribution be most appropriate? (Choose two.)

A. When the tables are highly denormalized and do NOT participate in frequent joins.
B. When data must be grouped based on a specific key on a defined slice.
C. When data transfer between nodes must be eliminated.
D. When a new table has been loaded and it is unclear how it will be joined to dimension.

Answer: AD

Explanation:

Here's a detailed justification for why options A and D are the most appropriate circumstances for choosing EVEN distribution in a Redshift cluster, along with supporting concepts and authoritative links:

The EVEN distribution style in Redshift distributes rows across the slices in a round-robin fashion, without considering the data values themselves. This makes it suitable for situations where you don't have a clear join key or don't need to colocate data based on a specific column.

Justification for A: When the tables are highly denormalized and do NOT participate in frequent joins.

In a denormalized schema, data is often duplicated across tables to avoid complex joins and improve query performance. If tables are already designed to minimize or eliminate joins, the benefit of KEY distribution (where rows with the same key are co-located) is lost.
EVEN distribution ensures that data is spread evenly across all nodes. Since no particular join key is used frequently, this avoids hotspots and maximizes the utilization of all computing resources. Using KEY distribution without a prevalent join key would be less effective and potentially lead to uneven data distribution.

Justification for D: When a new table has been loaded and it is unclear how it will be joined to dimension tables.

When a new table is created, the relationships between this table and other dimension tables may not be immediately known or understood. Choosing KEY distribution at this point could be premature and based on incorrect assumptions.
EVEN distribution is a safe initial choice for a new table. It guarantees a uniform distribution of the data across the cluster's nodes, which prevents skew and ensures all slices are being utilized.
Later, if the join patterns become clearer through queries and analysis, you can redistribute the table to KEY or ALL distribution based on the observed join behavior to further optimize performance.

Why the other options are less appropriate:

B. When data must be grouped based on a specific key on a defined slice: This is precisely when you should use KEY distribution. KEY distribution allows you to specify a column as the distribution key, ensuring that rows with the same value for that key are stored on the same compute node.
C. When data transfer between nodes must be eliminated: While minimizing data transfer is a goal, EVEN distribution doesn't inherently eliminate it. ALL distribution does, where an entire copy of the table is available on each node, reducing the need for inter-node communication during joins. KEY distribution can minimize data transfer if the tables are joined on the distribution key.

In summary: EVEN distribution provides a balanced approach to data distribution when join patterns are either nonexistent (denormalized schemas) or uncertain (newly loaded tables). It ensures even data distribution across the cluster nodes and prevents hotspots, making it a good choice for general-purpose scenarios or when a more specialized distribution style (KEY or ALL) is not obviously appropriate.

Authoritative Links:

Amazon Redshift Data Distribution Styles:
https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_a_data_distribution_style.html
Amazon Redshift Best Practices for Loading Data:
https://docs.aws.amazon.com/redshift/latest/dg/t_loading-tables.html

Question: 7

A large grocery distributor receives daily depletion reports from the field in the form of gzip archives od CSV files

uploaded to Amazon S3. The files range from 500MB to 5GB. These files are processed daily by an EMR job. Recently it has been observed that the file sizes vary, and the EMR jobs take too long. The distributor needs to tune and optimize the data processing workflow with this limited information to improve the performance of the EMR job.
Which recommendation should an administrator provide?

A.Reduce the HDFS block size to increase the number of task processors.

B.Use bzip2 or Snappy rather than gzip for the archives.

C.Decompress the gzip archives and store the data as CSV files.

D.Use Avro rather than gzip for the archives.

Answer: B

Explanation:

Here's a detailed justification for why using bzip2 or Snappy (option B) is the most suitable recommendation for improving the performance of the EMR job in this scenario:

The core problem is the variability in file sizes and the resulting long EMR job execution times. Gzip, while a decent compression algorithm, is not optimized for parallel processing, which is crucial for distributed computing environments like EMR. Gzip is primarily designed for sequential compression and decompression. When dealing with large files, this sequential nature can become a bottleneck.

Option B, suggesting bzip2 or Snappy, addresses this bottleneck directly. Bzip2 offers a higher compression ratio than gzip, meaning smaller files that can be transferred and processed faster. However, the compression and decompression process for bzip2 can be computationally intensive, potentially offsetting some of the benefits for very large files. Snappy, on the other hand, prioritizes speed over compression ratio. It
compresses less than gzip or bzip2 but offers significantly faster compression and decompression speeds. This speed is particularly advantageous in EMR jobs because it minimizes the time spent on data I/O, allowing the processing to be I/O bound. This is especially important for data warehousing or other big data scenarios.

Hadoop and EMR are designed to work efficiently with splittable compression formats. Splittable compression allows Hadoop to divide a compressed file into multiple blocks and process them in parallel across the cluster. While gzip is technically splittable, it requires additional indexing and is not efficiently splittable by default. Bzip2 is splittable by design, and Snappy is also designed to be splittable by default, making them suitable for parallel processing in Hadoop/EMR.

Option A, reducing HDFS block size, might increase the number of tasks, but it also increases the overhead of managing those tasks. It doesn't address the underlying problem of slow decompression. Option C, decompressing and storing as CSV, eliminates compression altogether, which will significantly increase storage space and network transfer times, making the problem worse. Option D, using Avro, changes the data format entirely. While Avro is a good choice for serialization and schema evolution, it doesn't directly address the compression problem. It would require a major refactoring of the data ingestion pipeline.

Therefore, switching to a splittable compression format like bzip2 or, more likely, Snappy offers the best trade-off between compression ratio and processing speed, allowing for efficient parallel processing in the EMR environment and improving job performance. Snappy's speed advantage usually outweighs the slightly lower compression, in the context of larger data amounts on EMR.

Authoritative Links:

Apache Hadoop Compression:https://hadoop.apache.org/docs/r1.2.1/compression.html
Snappy:https://github.com/google/snappy
Bzip2:https://en.wikipedia.org/wiki/Bzip2
EMR Performance Tuning Guide: Consult the official AWS EMR documentation for detailed guidance on performance optimization techniques, including data compression strategies. Search on AWS documentation "EMR Performance Tuning".

Question: 8

A web-hosting company is building a web analytics tool to capture clickstream data from all of the websites hosted within its platform and to provide near-real-time business intelligence. This entire system is built on AWS services. The web-hosting company is interested in using Amazon Kinesis to collect this data and perform sliding window analytics.
What is the most reliable and fault-tolerant technique to get each website to send data to Amazon Kinesis with every click?

A.After receiving a request, each web server sends it to Amazon Kinesis using the Amazon Kinesis PutRecord API. Use the sessionID as a partition key and set up a loop to retry until a success response is received.

B.After receiving a request, each web server sends it to Amazon Kinesis using the Amazon Kinesis Producer Library .addRecords method.

C.Each web server buffers the requests until the count reaches 500 and sends them to Amazon Kinesis using the Amazon Kinesis PutRecord API.

D.After receiving a request, each web server sends it to Amazon Kinesis using the Amazon Kinesis PutRecord API. Use the exponential back-off algorithm for retries until a successful response is received.

Answer: B

Explanation:

Here's a detailed justification for why option B is the most reliable and fault-tolerant technique for sending clickstream data to Amazon Kinesis, along with supporting concepts and links:

Option B, using the Amazon Kinesis Producer Library (KPL) .addRecords method, is the superior solution for several reasons. The KPL is designed specifically to simplify and optimize sending data to Kinesis Data Streams. It handles tasks like batching records, retrying failed requests, collecting metrics, and handling local retries, all automatically. This significantly offloads complexity from the individual web servers. Batching, especially, helps to improve throughput and reduce the number of API calls, leading to better performance.

Fault tolerance is enhanced by the KPL's automatic retry mechanism. If a request fails, the KPL will attempt to resend the data according to its configured retry policy. This ensures that data is not lost due to transient network issues or Kinesis throttling. The addRecords method also supports asynchronous operations, allowing web servers to continue processing requests without blocking while the KPL handles sending data in the background.

Option A, using the PutRecord API with a retry loop, is less efficient and more complex to implement correctly.
While retry mechanisms are essential, managing them manually within each web server adds overhead and increases the risk of errors. Retrying indefinitely without proper backoff can also exacerbate throttling issues. Using sessionID as the partition key is viable but independent of the reliability aspect.

Option C, buffering records until a count of 500 before sending using PutRecord, introduces latency. Near-real-time analytics require data to be available as quickly as possible. Delaying sending data until a buffer is full contradicts this requirement. Furthermore, losing buffered data before it is sent represents a significant reliability risk.

Option D, using PutRecord with exponential backoff for retries, is better than option A but still less optimal than using the KPL. While exponential backoff is a good strategy for handling retries, it still requires manual implementation and does not offer the other benefits provided by the KPL, such as batching and aggregation.

In summary, the KPL abstracts away the complexities of sending data to Kinesis Data Streams, providing built-in mechanisms for batching, retrying, and handling errors, making it the most reliable and fault-tolerant option. The other options either introduce significant performance overhead, increase latency, or require more complex manual implementation. The KPL is the recommended approach for producing data to Kinesis streams.

Authoritative Links:

Amazon Kinesis Producer Library (KPL):https://aws.amazon.com/kinesis/data-streams/developer-resources/ (search for KPL on the page)
Amazon Kinesis Data Streams Best Practices:https://docs.aws.amazon.com/streams/latest/dev/kinesis-data-streams-best-practices.html
Amazon Kinesis Data Streams Developer Guide:https://docs.aws.amazon.com/streams/latest/dev/

Question: 9

A customer has an Amazon S3 bucket. Objects are uploaded simultaneously by a cluster of servers from multiple streams of data. The customer maintains a catalog of objects uploaded in Amazon S3 using an
Amazon DynamoDB table. This catalog has the following fileds: StreamName, TimeStamp, and ServerName, from which ObjectName can be obtained.
The customer needs to define the catalog to support querying for a given stream or server within a defined time range.
Which DynamoDB table scheme is most efficient to support these queries?

A.Define a Primary Key with ServerName as Partition Key and TimeStamp as Sort Key. Do NOT define a Local Secondary Index or Global Secondary Index.

B.Define a Primary Key with StreamName as Partition Key and TimeStamp followed by ServerName as Sort Key. Define a Global Secondary Index with ServerName as partition key and TimeStamp followed by StreamName.

C.Define a Primary Key with ServerName as Partition Key. Define a Local Secondary Index with StreamName as Partition Key. Define a Global Secondary Index with TimeStamp as Partition Key.

D.Define a Primary Key with ServerName as Partition Key. Define a Local Secondary Index with TimeStamp as Partition Key. Define a Global Secondary Index with StreamName as Partition Key and TimeStamp as Sort Key.

Answer: B

Explanation:

Here's a detailed justification for why option B is the most efficient DynamoDB table scheme for the scenario described:

The core requirement is to efficiently query the catalog for objects within a given stream or server and a specified time range. This involves two query patterns: (1) Query by StreamName and TimeStamp, and (2) Query by ServerName and TimeStamp. DynamoDB's design emphasizes that queries using the primary key (partition key and optional sort key) are the most efficient. Therefore, designing the primary key to directly support one of the query patterns is crucial.

Option B defines the primary key with StreamName as the partition key and TimeStamp followed by ServerName as the sort key. This directly supports queries for a specific stream within a time range because DynamoDB can quickly locate the relevant partition (StreamName) and then efficiently filter the data within that partition using the composite sort key (TimeStamp followed by ServerName) based on the specified time range.

To support querying by ServerName and TimeStamp, option B defines a Global Secondary Index (GSI) with ServerName as the partition key and TimeStamp followed by StreamName as the sort key. GSIs allow you to efficiently query the data using attributes other than the primary key. The GSI will let you query using a server and timestamp and it can query across all the data. The addition of StreamName to the sort key is not necessarily required.

Option A is not optimal because it does not allow efficient querying by StreamName.

Option C is less efficient because it uses Local Secondary Indexes (LSIs), which must reside on the same partition as the base table. Since the base table's partition key is ServerName, you can only query for a limited set of servers on the same partition which can be very limited. It can not query using a server and a time range.

Option D is less efficient as it uses LSI for TimeStamp based on ServerName. While queries by stream name will be fast the time range filter will be difficult since DynamoDB can only query for a limited set of servers based on the partition key of ServerName.

Therefore, option B strikes the best balance, directly supporting one query pattern through the primary key and efficiently supporting the other through a GSI.

Authoritative Links:

DynamoDB Primary Keys:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html#HowItW
DynamoDB Secondary Indexes:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html
Choosing the Right DynamoDB Partition Key:https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/

Question: 10

A company has several teams of analysts. Each team of analysts has their own cluster. The teams need to run SQL queries using Hive, Spark-SQL, and Presto with Amazon EMR. The company needs to enable a centralized metadata layer to expose the Amazon S3 objects as tables to the analysts.
Which approach meets the requirement for a centralized metadata layer?

A.EMRFS consistent view with a common Amazon DynamoDB table

B.Bootstrap action to change the Hive Metastore to an Amazon RDS database

C.s3distcp with the outputManifest option to generate RDS DDL

D.Naming scheme support with automatic partition discovery from Amazon S3

Answer: A

Explanation:

The best approach is EMRFS consistent view with a common Amazon DynamoDB table because it provides a centralized and consistent metadata layer across multiple EMR clusters using various query engines (Hive, Spark-SQL, Presto). EMRFS consistent view, coupled with DynamoDB, ensures that all clusters accessing the same data in S3 see a consistent view of the metadata, even if changes are made concurrently. This avoids issues like reading stale data or inconsistent schemas. A shared DynamoDB table acts as the single source of truth for the metadata, centrally managed and accessible to all teams.

Option B, while using RDS for Hive Metastore is also a viable solution for centralizing metadata, it only caters to Hive. It does not inherently provide metadata consistency for Spark-SQL or Presto, which may require their own metastore configurations or integration strategies. Therefore, it doesn't provide a unified solution for all the mentioned query engines.

Option C, s3distcp with outputManifest, is used for data copying and doesn't provide a continuously consistent metadata layer. It is more suitable for data migration and synchronization rather than metadata management.

Option D, Naming scheme with automatic partition discovery, relies on a well-defined directory structure for partitioning data in S3. While useful, it doesn't inherently provide metadata consistency across multiple clusters and requires specific configurations for each query engine to discover partitions. It's less robust compared to the EMRFS consistent view for ensuring consistency across different query engines like Hive, Spark, and Presto.

In essence, EMRFS consistent view with DynamoDB offers the most comprehensive and centralized solution for maintaining consistent metadata across multiple EMR clusters using diverse query engines.

Further reading:

EMRFS consistent view
Using the Hive Metastore with Amazon EMR

Question: 11

An administrator needs to manage a large catalog of items from various external sellers. The administrator needs to determine if the items should be identified as minimally dangerous, dangerous, or highly dangerous based on their textual descriptions. The administrator already has some items with the danger attribute, but receives hundreds of new item descriptions every day without such classification.
The administrator has a system that captures dangerous goods reports from customer support team of from user feedback.
What is a cost-effective architecture to solve this issue?

A.Build a set of regular expression rules that are based on the existing examples, and run them on the DynamoDB Streams as every new item description is added to the system.

B.Build a Kinesis Streams process that captures and marks the relevant items in the dangerous goods reports using a Lambda function once more than two reports have been filed.

C.Build a machine learning model to properly classify dangerous goods and run it on the DynamoDB Streams as every new item description is added to the system.

D.Build a machine learning model with binary classification for dangerous goods and run it on the DynamoDB Streams as every new item description is added to the system.

Answer: C

Explanation:

Here's a detailed justification for why option C is the most cost-effective and suitable solution for classifying dangerous goods based on item descriptions, compared to the other options:

Option A, relying solely on regular expressions, is insufficient for nuanced classification. Regular expressions are brittle and struggle with variations in language, synonyms, and implicit meanings. While they might catch obvious keywords, they'll miss subtle indicators of danger, leading to inaccurate classifications and potential risks. They also require continuous manual maintenance as the catalog and language evolve.

Option B, using Kinesis Streams and Lambda based on customer reports, is reactive. It only flags items after two reports have been filed, meaning potentially dangerous items are available for purchase until reported. This is unacceptable from a safety perspective. Furthermore, relying on user reports introduces significant delays and inconsistencies in classification.

Option C, building and applying a machine learning (ML) model, provides a proactive and scalable solution. ML models, specifically those trained on natural language processing (NLP), can learn complex patterns and relationships between words and the danger attribute. Given the existing dataset of classified items and customer feedback, a supervised learning approach is viable. The model can be deployed to analyze incoming item descriptions via DynamoDB Streams, providing real-time classification. This leverages the existing data for automated, consistent classification.

Option D is close, but unnecessarily limits the model to binary classification (dangerous/not dangerous). The requirement specifies identifying items as minimally dangerous, dangerous, or highly dangerous, necessitating a multi-class classification model. While a binary classification could be the first stage, it wouldn't directly address the core requirement. Furthermore, the cost difference between a binary and multi-class classification model is usually negligible.

Considering cost-effectiveness, while building and training an ML model requires initial investment, it ultimately reduces manual review and improves accuracy, minimizing the risk of misclassification. Furthermore, using a managed ML service like Amazon SageMaker reduces the operational overhead of managing the ML infrastructure. Running the model on DynamoDB Streams via Lambda for inference is a cost-effective way to process incoming data. The cost associated with maintaining and updating regular expressions over time can easily outweigh the cost of running a managed machine learning model.

In summary, Option C offers a proactive, scalable, and cost-effective approach to classifying dangerous goods based on textual descriptions, leveraging the power of machine learning to achieve higher accuracy and reduce manual effort, and fulfilling all requirements, including different levels of dangerousness.

Authoritative Links:

Amazon SageMaker:https://aws.amazon.com/sagemaker/
Amazon DynamoDB Streams:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html **AWS Lambda:**https://aws.amazon.com/lambda/
Natural Language Processing (NLP):https://aws.amazon.com/machine-learning/nlp/

Question: 12

A company receives data sets coming from external providers on Amazon S3. Data sets from different providers are dependent on one another. Data sets will arrive at different times and in no particular order.
A data architect needs to design a solution that enables the company to do the following:
➾ Rapidly perform cross data set analysis as soon as the data becomes available
➾ Manage dependencies between data sets that arrive at different times
Which architecture strategy offers a scalable and cost-effective solution that meets these requirements?

A.Maintain data dependency information in Amazon RDS for MySQL. Use an AWS Data Pipeline job to load an Amazon EMR Hive table based on task dependencies and event notification triggers in Amazon S3.

B.Maintain data dependency information in an Amazon DynamoDB table. Use Amazon SNS and event notifications to publish data to fleet of Amazon EC2 workers. Once the task dependencies have been resolved, process the data with Amazon EMR.

C.Maintain data dependency information in an Amazon ElastiCache Redis cluster. Use Amazon S3 event notifications to trigger an AWS Lambda function that maps the S3 object to Redis. Once the task dependencies have been resolved, process the data with Amazon EMR.

D.Maintain data dependency information in an Amazon DynamoDB table. Use Amazon S3 event notifications to trigger an AWS Lambda function that maps the S3 object to the task associated with it in DynamoDB. Once all task dependencies have been resolved, process the data with Amazon EMR.

Answer: D

Explanation:

Here's a detailed justification for why option D is the most suitable architecture strategy, along with supporting explanations and links:

Option D leverages a combination of services to achieve rapid cross-dataset analysis while effectively managing dependencies in a scalable and cost-effective manner.

1. **DynamoDB for Dependency Tracking:** DynamoDB's ability to handle high read/write throughput and its key-value store structure make it ideal for maintaining data dependency information. Each data set can be represented as an item in DynamoDB, with attributes indicating its dependencies and status (e.g., 'received', 'processed'). The ACID properties for transactions in DynamoDB can be used to

assure data consistency.

2. **S3 Event Notifications and Lambda:** Amazon S3 event notifications triggered by data arrival can invoke an AWS Lambda function. This function maps the newly arrived S3 object to its corresponding task/data set entry in DynamoDB. It then updates the status of the dependency in DynamoDB and checks if all dependencies for a given task have been met. Lambda provides a serverless, event-driven compute environment, enabling cost-effective and rapid response to S3 events.

3. **EMR for Processing:** Once all dependencies for a particular task/data set are resolved (as tracked in DynamoDB), Lambda triggers an Amazon EMR job. EMR is well-suited for large-scale data processing and analytics. By only launching EMR when dependencies are met, processing resources are utilized efficiently.

Here's why the other options are less suitable:

Option A (RDS for MySQL and Data Pipeline): Using RDS for dependency tracking is less scalable than DynamoDB. AWS Data Pipeline, while capable, is generally used for more complex workflows and might be overkill for simple dependency management. Hive is tightly coupled to Hadoop and requires knowledge of it.

Option B (SNS and EC2): While SNS and EC2 can handle event processing, managing dependencies and state across a fleet of EC2 instances is more complex and less scalable than using DynamoDB to store and track the status. It also involves more overhead in managing the EC2 instances.

Option C (ElastiCache Redis and Lambda): ElastiCache Redis is primarily designed for caching, not for persistent storage and dependency management. While it offers fast read/write access, it lacks the transactional consistency and durability that DynamoDB provides, making it unsuitable for tracking critical dependency information.

In summary, Option D offers a scalable, cost-effective, and efficient solution by utilizing DynamoDB for dependency management, Lambda for event-driven processing, and EMR for large-scale data analysis.**Authoritative Links:**

Amazon DynamoDB: https://aws.amazon.com/dynamodb/
AWS Lambda: https://aws.amazon.com/lambda/
Amazon EMR: https://aws.amazon.com/emr/
Amazon S3 Event Notifications: https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-event-notifications.html

Question: 13

A media advertising company handles a large number of real-time messages sourced from over 200 websites in real time. Processing latency must be kept low. Based on calculations, a 60-shard Amazon Kinesis stream is more than sufficient to handle the maximum data throughput, even with traffic spikes. The company also uses an Amazon Kinesis Client Library (KCL) application running on Amazon Elastic Compute Cloud (EC2) managed by an Auto Scaling group. Amazon CloudWatch indicates an average of 25% CPU and a modest level of network traffic across all running servers.
The company reports a 150% to 200% increase in latency of processing messages from Amazon Kinesis during peak times. There are NO reports of delay from the sites publishing to Amazon Kinesis.
What is the appropriate solution to address the latency?

A.Increase the number of shards in the Amazon Kinesis stream to 80 for greater concurrency.
B.Increase the size of the Amazon EC2 instances to increase network throughput.
C.Increase the minimum number of instances in the Auto Scaling group.
D.Increase Amazon DynamoDB throughput on the checkpoint table.

Answer: D

Explanation:

Here's a detailed justification for why increasing DynamoDB throughput on the checkpoint table is the most likely solution, along with links for further reading:

The problem is increased latency during peak times in Kinesis processing, while the data ingestion into Kinesis is reportedly fine, and EC2 resources appear underutilized. This points to a bottleneck in the KCL application itself, specifically in its ability to maintain its state and track progress. The KCL uses a DynamoDB table to checkpoint its progress through the Kinesis stream, ensuring that data processing resumes correctly after failures or scale-out events.

If the DynamoDB table's throughput is insufficient, especially during peak load, the KCL application will experience significant delays when updating checkpoints. This delay directly translates into increased latency in processing Kinesis records. The KCL workers will spend more time waiting to write/read checkpoints, thus slowing down the overall processing pipeline. Increasing the number of shards (option A) might seem helpful, but without addressing the checkpointing bottleneck, it will only exacerbate the problem by increasing the load on the DynamoDB table. Upgrading the EC2 instance size (option B) will not impact since CloudWatch indicates that CPU and network traffic across the instances are low, meaning that they are not the constraint.

While increasing the number of EC2 instances (option C) could potentially help, it doesn't address the fundamental bottleneck. Each new instance relies on the DynamoDB checkpoint table, so if that's slow, adding instances will just distribute the bottleneck, not eliminate it.

The fact that latency increases drastically during peak times strongly suggests that the DynamoDB table is struggling to keep up with the increased checkpointing demands. The underutilized EC2 resources further support this idea, because the KCL worker tasks are waiting on I/O for the DynamoDB table. Therefore, increasing DynamoDB throughput will reduce the time spent on checkpointing, directly reducing the overall processing latency of messages from Amazon Kinesis.

Here are some resources for further reading:

Amazon Kinesis Client Library (KCL):https://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-scaling.html
Monitoring KCL Applications:https://docs.aws.amazon.com/streams/latest/dev/monitoring.html **Amazon DynamoDB Auto Scaling:**
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html

Question: 14

A Redshift data warehouse has different user teams that need to query the same table with very different query types.
These user teams are experiencing poor performance.
Which action improves performance for the user teams in this situation?

A.Create custom table views.
B.Add interleaved sort keys per team.
C.Maintain team-specific copies of the table.
D.Add support for workload management queue hopping.

Answer: D

Explanation:

The correct answer is **D. Add support for workload management queue hopping.**

Here's a detailed justification:

Redshift performance can degrade when different user teams with vastly different query patterns compete for resources on the same cluster. Workload management (WLM) in Redshift allows you to prioritize queries based on their resource needs and importance. However, if a long-running query from one team is occupying a WLM queue, shorter, more important queries from another team may be delayed, leading to performance issues.

Queue hopping directly addresses this. It allows a query to be moved from one WLM queue to another based on defined rules and conditions, like query execution time exceeding a threshold. This prevents long-running, resource-intensive queries from indefinitely blocking shorter, higher-priority queries from other teams. By automatically escalating or de-escalating queries based on their behavior and defined parameters, queue hopping ensures that resources are dynamically allocated to where they are most needed, improving overall cluster performance.

Option A, creating custom table views, can simplify queries for specific teams but does not address the underlying resource contention. Views merely provide a different perspective on the data but don't resolve performance bottlenecks caused by competing workloads.

Option B, adding interleaved sort keys per team, is impractical and inefficient. Interleaved sort keys are designed to improve performance for queries that frequently filter on multiple columns. However, adding multiple interleaved sort keys for different teams leads to excessive storage overhead, increased data skew, and potential performance degradation during data loading and updates. Only one interleaved sort key is supported per table.

Option C, maintaining team-specific copies of the table, is also a poor solution. It duplicates data, significantly increasing storage costs, and introduces data consistency challenges. It also requires complex data replication mechanisms to keep the copies synchronized, which can add latency and overhead. Furthermore, even with separate copies, the underlying resource contention problem on the Redshift cluster remains if these teams access other shared resources simultaneously.

WLM queue hopping is a more efficient and cost-effective solution that dynamically manages query prioritization and resource allocation to improve performance for all user teams by optimizing resource utilization without requiring data duplication or complex data management strategies.

Authoritative Links:

AWS Redshift Workload Management (WLM):https://docs.aws.amazon.com/redshift/latest/dg/tutorial-defining-wlm-queues.html
AWS Redshift WLM Queue Hopping:https://docs.aws.amazon.com/redshift/latest/dg/wlm-queue-hopping.html

Question: 15

A company operates an international business served from a single AWS region. The company wants to expand into a new country. The regulator for that country requires the Data Architect to maintain a log of financial transactions in the country within 24 hours of the product transaction. The production application is latency insensitive. The new country contains another AWS region.
What is the most cost-effective way to meet this requirement?

A.Use CloudFormation to replicate the production application to the new region.

B.Use Amazon CloudFront to serve application content locally in the country; Amazon CloudFront logs will satisfy the requirement.

C.Continue to serve customers from the existing region while using Amazon Kinesis to stream transaction data to the regulator.

D.Use Amazon S3 cross-region replication to copy and persist production transaction logs to a bucket in the new countrys region.

Answer: D

Explanation:

The most cost-effective solution is to use Amazon S3 cross-region replication to copy transaction logs to a bucket within the new country's AWS Region. This approach directly addresses the regulator's requirement to maintain a log of financial transactions within the specified timeframe.

Option D is superior to other options for several reasons:

Cost-Effectiveness: S3 cross-region replication is generally cheaper than replicating the entire application stack using CloudFormation (Option A). It focuses solely on replicating the necessary log data.

Data Residency Compliance: S3 replication ensures that a copy of the logs resides within the new country's region, satisfying data residency requirements.

Simplicity and Maintainability: It's simpler to implement and maintain compared to a full application replication or complex Kinesis data streams (Option C).

Relevance: CloudFront (Option B) caches content for faster delivery, but CloudFront logs do not directly satisfy the requirement for maintaining financial transaction logs within the country's region.

Specific Focus: This solution specifically addresses the log residency requirement without incurring the overhead of replicating the entire production environment.

Option A is too broad and expensive, as it replicates the entire application. Option B is irrelevant because CloudFront logs aren't suitable for detailed financial transaction auditing. Option C might work, but Kinesis introduces additional complexity and cost compared to the simplicity of S3 replication for log files. S3 replication is designed for asynchronous data transfer across regions, making it ideal for near real-time replication of log data, provided the latency of replication aligns with the 24-hour window.

Here are some authoritative links for further research:

Amazon S3 Cross-Region Replication:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication.html
Amazon Kinesis:https://aws.amazon.com/kinesis/
AWS CloudFormation:https://aws.amazon.com/cloudformation/
Amazon CloudFront:https://aws.amazon.com/cloudfront/

Question: 16

An administrator needs to design the event log storage architecture for events from mobile devices. The event data will be processed by an Amazon EMR cluster daily for aggregated reporting and analytics before being archived.
How should the administrator recommend storing the log data?

A.Create an Amazon S3 bucket and write log data into folders by device. Execute the EMR job on the device folders.

B.Create an Amazon DynamoDB table partitioned on the device and sorted on date, write log data to table. Execute the EMR job on the Amazon DynamoDB table.

C.Create an Amazon S3 bucket and write data into folders by day. Execute the EMR job on the daily folder.

D.Create an Amazon DynamoDB table partitioned on EventID, write log data to table. Execute the EMR job on the table.

Answer: C

Explanation:

Here's a detailed justification for why option C is the best choice for storing mobile device event logs for daily EMR processing, along with why the other options are less suitable:

Justification for Option C: Create an Amazon S3 bucket and write data into folders by day. Execute the EMR job on the daily folder.

Option C leverages the strengths of Amazon S3 and its efficient integration with Amazon EMR. S3 is a highly scalable, durable, and cost-effective object storage service ideal for storing large volumes of log data. Organizing logs into folders by day aligns perfectly with the requirement of daily processing. EMR jobs can be easily configured to target specific S3 folders (representing a single day's worth of logs), allowing for parallel processing and efficient data ingestion. This approach avoids complex data partitioning or querying inherent in database solutions, which simplifies the EMR job design and reduces processing overhead. Furthermore, S3's object-based storage is optimized for batch processing like the daily aggregation and analytics described in the problem. This means that EMR can efficiently read the entire day's log data without being bottlenecked by database read operations. This is particularly crucial for handling potentially large volumes of mobile event data. Finally, after the daily processing is complete, the logs can be transitioned to lower-cost storage tiers within S3 (like S3 Glacier or S3 Glacier Deep Archive) for long-term archival, optimizing cost efficiency.

Why other options are less suitable:

Option A: Create an Amazon S3 bucket and write log data into folders by device. Execute the EMR job on the device folders. While using S3 is a good start, partitioning by device is less optimal for daily aggregation. The EMR job would need to iterate through many device folders to gather all data for a single day, which adds complexity and overhead.

Option B: Create an Amazon DynamoDB table partitioned on the device and sorted on date, write log data to table. Execute the EMR job on the Amazon DynamoDB table. DynamoDB is a NoSQL database suited for low-latency reads and writes, but it's not ideal for large-scale batch processing as required by the EMR job.

Scanning a DynamoDB table for daily aggregation would be inefficient and costly compared to directly processing files in S3. Also, managing DynamoDB can be more complex and expensive than using S3 for log storage. Direct integration of EMR with DynamoDB isn't as seamless as with S3, which is optimized for Hadoop-based processing.

Option D: Create an Amazon DynamoDB table partitioned on EventID, write log data to table. Execute the EMR job on the table. Partitioning by EventID makes it extremely difficult to efficiently query for all events that occurred on a particular day. It would require a full table scan, which is highly inefficient. The limitations of using DynamoDB with EMR as described above also apply here.

Authoritative Links:

Amazon S3:https://aws.amazon.com/s3/
Amazon EMR:https://aws.amazon.com/emr/
Amazon DynamoDB:https://aws.amazon.com/dynamodb/
EMR and S3 integration:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-s3distcp.html

Question: 17

A data engineer wants to use an Amazon Elastic Map Reduce for an application. The data engineer needs to make sure it complies with regulatory requirements. The auditor must be able to confirm at any point which servers are running and which network access controls are deployed.
Which action should the data engineer take to meet this requirement?

A.Provide the auditor IAM accounts with the SecurityAudit policy attached to their group.

B.Provide the auditor with SSH keys for access to the Amazon EMR cluster.

C.Provide the auditor with CloudFormation templates.

D.Provide the auditor with access to AWS DirectConnect to use their existing tools.

Answer: A

Explanation:

The correct answer is **A. Provide the auditor IAM accounts with the SecurityAudit policy attached to their group.** Here's why:

IAM (Identity and Access Management) is the core AWS service for controlling access to AWS resources. The SecurityAudit policy is a pre-defined AWS managed policy that grants read-only access to a wide range of AWS services and resources, including EC2 (used by EMR), VPC (for network configurations), CloudTrail (for audit logs), and more. This is crucial because:

Visibility: The SecurityAudit policy enables the auditor to see the configuration of the EMR cluster's underlying infrastructure, including EC2 instances, network configurations (VPC, security groups, etc.), and storage configurations. This allows them to verify which servers are running and what network access controls are in place.

Least Privilege: Providing read-only access adheres to the principle of least privilege. The auditor only needs to see the configuration, not modify it.

Centralized Control: IAM provides centralized management of user permissions, making it easy to grant and revoke access as needed.

Compliance: Using IAM and managed policies helps demonstrate compliance with various regulatory requirements, as access is controlled and auditable.

Why other options are incorrect:

B. Provide the auditor with SSH keys for access to the Amazon EMR cluster: Providing SSH keys gives direct access to the cluster's operating system, which is far more access than necessary for an audit. It violates the principle of least privilege and could potentially allow the auditor to make unintended changes or expose sensitive data. It's an operational access mechanism, not an auditing one.

C. Provide the auditor with CloudFormation templates: While CloudFormation templates define the infrastructure, they only show the intended configuration, not the actual running state. The auditor needs to verify what's currently running, not what was intended to be running. Templates alone don't show the running state or the historical configurations.

D. Provide the auditor with access to AWS DirectConnect to use their existing tools: Direct Connect provides a dedicated network connection to AWS. This is completely irrelevant to the audit requirement. It doesn't provide visibility into the EMR cluster's configuration and adds unnecessary complexity and cost. It's for network connectivity, not for auditing.

In summary, granting the auditor IAM accounts with the SecurityAudit policy is the most secure, efficient, and compliant way to meet the requirement of confirming which servers are running and which network access controls are deployed in the EMR cluster. It provides the necessary visibility without granting excessive privileges.

Authoritative Links:

AWS IAM:https://aws.amazon.com/iam/
AWS SecurityAudit Policy:https://docs.aws.amazon.com/IAM/latest/UserGuide/access-analyzer-reference-policy-checks.html
Principle of Least Privilege:https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html

Question: 18

A social media customer has data from different data sources including RDS running MySQL, Redshift, and Hive on EMR. To support better analysis, the customer needs to be able to analyze data from different data sources and to combine the results.
What is the most cost-effective solution to meet these requirements?

A.Load all data from a different database/warehouse to S3. Use Redshift COPY command to copy data to Redshift for analysis.

B.Install Presto on the EMR cluster where Hive sits. Configure MySQL and PostgreSQL connector to select from different data sources in a single query.

C.Spin up an Elasticsearch cluster. Load data from all three data sources and use Kibana to analyze.

D.Write a program running on a separate EC2 instance to run queries to three different systems. Aggregate the results after getting the responses from all three systems.

Answer: B

Explanation:

The most cost-effective solution is using Presto on EMR with connectors. Here's why:

Option B leverages the existing EMR cluster where Hive already resides, minimizing the need to provision new infrastructure. Presto is designed for querying data across multiple sources, and its connectors for MySQL and Redshift (which can connect to PostgreSQL-compatible databases) allow it to seamlessly access data from RDS, Redshift, and Hive without data movement. This eliminates the need for expensive and time-consuming ETL processes.

Option A, loading all data into S3 and then Redshift, incurs storage costs for S3 and processing costs for the Redshift COPY command. It also duplicates the data, increasing storage costs. This data duplication creates data management challenges, as maintaining consistency between the original sources and the Redshift data becomes crucial.

Option C, using Elasticsearch and Kibana, is not suitable for complex analytical queries involving joins across multiple data sources. Elasticsearch excels in full-text search and log analytics, not relational data analysis. Moreover, ingesting data into Elasticsearch would require significant ETL effort and resource consumption.

Option D, writing a custom program on EC2, involves significant development and maintenance overhead. It also introduces complexity in handling data type conversions, error handling, and query optimization across different data sources. This solution would also likely be less performant than using a specialized query engine like Presto.

Presto's distributed query engine architecture enables parallel processing, providing faster query execution compared to custom scripting. By utilizing Presto on the existing EMR cluster and leveraging its connectors, the solution achieves the required data analysis capability with minimal infrastructure cost, development effort, and data movement. This aligns with the principle of cost optimization in cloud computing.

Here are some helpful links:

Presto Documentation:https://prestodb.io/
EMR Presto:https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-presto.html

Question: 19

An Amazon EMR cluster using EMRFS has access to petabytes of data on Amazon S3, originating from multiple unique data sources. The customer needs to query common fields across some of the data sets to be able to perform interactive joins and then display results quickly.
Which technology is most appropriate to enable this capability?

A.Presto
B.MicroStrategy
C.Pig
D.R Studio

Answer: A

Explanation:

The most appropriate technology for querying common fields across multiple large datasets on Amazon S3 for interactive joins and quick results is Presto.

Here's why:

Presto is a distributed SQL query engine designed for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes. Its key strength is its ability to query data directly from various data sources like Amazon S3 without requiring data movement or transformation. This "query in place" capability makes it ideal for the described scenario.

Presto's architecture utilizes a massively parallel processing (MPP) approach, enabling it to distribute query execution across a cluster of nodes. This parallel processing significantly accelerates query execution, delivering fast results even on petabyte-scale datasets. This aligns perfectly with the requirement to display results quickly.

EMRFS (EMR File System) provides direct access to data stored in Amazon S3 from within an EMR cluster. Presto integrates seamlessly with EMRFS, allowing it to efficiently access and query the data in S3.

The other options are less suitable:

MicroStrategy is a business intelligence and data visualization tool. While it can connect to various data sources and display results, it is not primarily designed for running complex interactive SQL queries directly against petabyte-scale data on S3.

Pig is a high-level data flow language and execution framework for parallel data processing. Although Pig can process large datasets on S3, it's more suited for batch-oriented ETL (Extract, Transform, Load) processes and less ideal for interactive queries requiring quick responses. The scripting nature of Pig adds overhead compared to the direct SQL execution of Presto.

R Studio is an integrated development environment (IDE) primarily used for statistical computing and graphics. It's not designed for querying petabyte-scale data directly from S3 for interactive analysis. While R can connect to databases and data sources, it becomes impractical for large-scale data exploration and joining due to memory limitations and processing overhead.

In summary, Presto's MPP architecture, ability to query data in place on S3, seamless integration with EMRFS, and SQL-based interface make it the superior choice for interactive querying and quick result delivery from petabyte-scale datasets residing on Amazon S3, fulfilling the customer's requirements precisely.

Authoritative Links:

Presto Documentation:https://prestodb.io/
Amazon EMR Presto:https://aws.amazon.com/emr/features/presto/
EMRFS:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emrfs-s3.html

Question: 20

A game company needs to properly scale its game application, which is backed by DynamoDB. Amazon Redshift has the past two years of historical data. Game traffic varies throughout the year based on various factors

such as season, movie release, and holiday season. An administrator needs to calculate how much read and write throughput should be provisioned for DynamoDB table for each week in advance.
How should the administrator accomplish this task?

A.Feed the data into Amazon Machine Learning and build a regression model.

B.Feed the data into Spark Mlib and build a random forest modest.

C.Feed the data into Apache Mahout and build a multi-classification model.

D.Feed the data into Amazon Machine Learning and build a binary classification model.

Answer: A

Explanation:

The correct approach is **A. Feed the data into Amazon Machine Learning and build a regression model.**

Here's why:

The administrator's goal is to predict future read and write throughput for DynamoDB based on historical data. This is a classic time series forecasting problem. Regression models are specifically designed to predict continuous numerical values, making them suitable for predicting the amount of read/write throughput.

Regression vs. Classification: The core of the decision lies in choosing the right model type. Throughput is a continuous numerical value (e.g., 1000 reads/second, 500 writes/second). Classification models (like binary or multi-classification) predict categories or classes (e.g., "high traffic," "low traffic"), which doesn't give you the specific throughput values needed for provisioning. The task isn't to classify traffic, but to estimate it.

Amazon Machine Learning (AML): AML is a managed service that simplifies the process of building and deploying machine learning models on AWS. It can handle the historical data from Amazon Redshift, train a regression model, and provide predictions.

Spark MLlib and Apache Mahout: While Spark MLlib and Apache Mahout are powerful machine learning libraries, using them directly would require significantly more setup and management compared to AML. AML is designed for easier use with AWS data sources like Redshift. Also, Spark requires setting up an EC2 instance (or using EMR), and Apache Mahout is generally less preferred for scalable data processing compared to Spark. In a simple case like this, using AML is more straightforward, serverless, and fits the AWS environment seamlessly.

Regression for Forecasting: A regression model can learn the relationship between historical factors (season, movie release, holidays) and the corresponding read/write throughput. Once trained, the model can predict the expected throughput for future weeks based on the upcoming season, releases, and holidays.

Justification by Example: Imagine a scenario where high traffic occurred historically during the holiday season in December. A regression model can learn this pattern from the Redshift data. When the administrator provides information about the upcoming December, the model will predict a higher read/write throughput based on the learned holiday traffic pattern.

Therefore, feeding the historical data into Amazon Machine Learning and building a regression model allows the administrator to predict the DynamoDB throughput accurately for future weeks, enabling proper provisioning and scaling.

Authoritative Links:

Amazon Machine Learning:https://aws.amazon.com/machine-learning/
Regression Analysis:https://en.wikipedia.org/wiki/Regression_analysis

Question: 21

A data engineer is about to perform a major upgrade to the DDL contained within an Amazon Redshift cluster to support a new data warehouse application. The upgrade scripts will include user permission updates, view and table structure changes as well as additional loading and data manipulation tasks.
The data engineer must be able to restore the database to its existing state in the event of issues.
Which action should be taken prior to performing this upgrade task?

A.Run an UNLOAD command for all data in the warehouse and save it to S3.

B.Create a manual snapshot of the Amazon Redshift cluster.

C.Make a copy of the automated snapshot on the Amazon Redshift cluster.

D.Call the waitForSnapshotAvailable command from either the AWS CLI or an AWS SDK.

Answer: B

Explanation:

The correct answer is **B. Create a manual snapshot of the Amazon Redshift cluster.**

Here's why:

A major DDL upgrade involving significant changes to database structure, permissions, and data manipulation carries a considerable risk of failure or unintended consequences. In such scenarios, a reliable rollback mechanism is crucial.

Creating a manual snapshot of the Redshift cluster provides a point-in-time backup that can be used to fully restore the cluster to its pre-upgrade state. This offers the most comprehensive protection against data loss or corruption caused by the upgrade process. If the upgrade encounters problems, the cluster can be reverted to the snapshot, effectively undoing all the changes.

Option A, using the UNLOAD command, only backs up the data itself, not the DDL (table structures, views, user permissions, etc.). Restoring from unloaded data would require manually recreating the entire database schema and permissions, which is time-consuming and error-prone.

Option C, copying an automated snapshot, is less desirable because automated snapshots are typically taken on a schedule, and the most recent one may not represent the exact state immediately before the upgrade. A manual snapshot ensures that the backup reflects the database's state right before the upgrade is initiated.

Option D, calling waitForSnapshotAvailable, simply checks the status of a snapshot; it doesn't create one. It is a useful command but doesn't address the core requirement of having a backup.

Therefore, creating a manual snapshot offers the best and most complete safeguard for a critical database upgrade, allowing for a full and efficient rollback if needed.

For further reading, refer to the Amazon Redshift documentation on snapshots:

Amazon Redshift Snapshots
Create a Manual Snapshot

Question: 22

A large oil and gas company needs to provide near real-time alerts when peak thresholds are exceeded in its pipeline system. The company has developed a system to capture pipeline metrics such as flow rate, pressure, and temperature using millions of sensors. The sensors deliver to AWS IoT.
What is a cost-effective way to provide near real-time alerts on the pipeline metrics?

A.Create an AWS IoT rule to generate an Amazon SNS notification.

B.Store the data points in an Amazon DynamoDB table and poll if for peak metrics data from an Amazon EC2 application.

C.Create an Amazon Machine Learning model and invoke it with AWS Lambda.

D.Use Amazon Kinesis Streams and a KCL-based application deployed on AWS Elastic Beanstalk.

Answer: A

Explanation:

Here's a justification for choosing option A, along with supporting details and links:

Option A, creating an AWS IoT rule to generate an Amazon SNS notification, is the most cost-effective and efficient solution for near real-time alerts when peak thresholds are exceeded in the pipeline system. AWS IoT rules engine allows you to ingest messages from connected devices, process data based on defined rules, and take actions, such as sending notifications.

The IoT rule can be configured to monitor the pipeline metrics flowing from the sensors. Using a SQL-like syntax, the rule can filter for messages where flow rate, pressure, or temperature exceed predefined thresholds. When a threshold is breached, the rule triggers an action.

The recommended action is sending an Amazon SNS (Simple Notification Service) notification. SNS is a highly scalable, fully managed pub/sub messaging service. It allows you to easily fan out notifications to a large number of subscribers (e.g., email, SMS, HTTP endpoints, other AWS services). This setup provides near real-time alerting because the IoT rule evaluates the data as it arrives, and SNS delivers notifications with minimal latency.

Option B is less efficient and more costly. Polling a DynamoDB table from an EC2 instance adds unnecessary complexity and latency. Continuously querying DynamoDB consumes read capacity units, increasing costs. This approach introduces latency due to polling intervals and application processing.

Option C, using Amazon Machine Learning and Lambda, is overkill for simple threshold alerts. Machine Learning is typically used for more complex anomaly detection or predictive analysis, not simple threshold comparisons. Setting up and maintaining a Machine Learning model and associated Lambda functions is more complex and expensive than the IoT rule and SNS solution.

Option D, using Kinesis Streams and Elastic Beanstalk, is also more complex and costly than necessary. Kinesis Streams is suitable for high-throughput data ingestion and real-time processing of streaming data. While it could be used, it adds significant overhead compared to a simple IoT rule. Deploying a KCL-based application on Elastic Beanstalk involves managing infrastructure and application code, which increases operational overhead.

In summary, option A leverages the AWS IoT rule engine for real-time filtering and Amazon SNS for scalable notifications, providing a simple, efficient, and cost-effective solution for near real-time alerts based on threshold breaches.

Authoritative Links:

AWS IoT Rules Engine:https://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html
Amazon SNS:https://aws.amazon.com/sns/

Question: 23

A company is using Amazon Machine Learning as part of a medical software application. The application will predict the most likely blood type for a patient based on a variety of other clinical tests that are available when blood type knowledge is unavailable.
What is the appropriate model choice and target attribute combination for this problem?

A.Multi-class classification model with a categorical target attribute.

B.Regression model with a numeric target attribute.

C.Binary Classification with a categorical target attribute.

D.K-Nearest Neighbors model with a multi-class target attribute.

Answer: A

Explanation:

The best answer is A: Multi-class classification model with a categorical target attribute. Here's why:

The problem involves predicting a patient's blood type. Blood types (A, B, AB, O, and Rh+/-) are distinct categories, not continuous numeric values. Therefore, a classification model is more suitable than a regression model.

Regression models predict a continuous numerical value (e.g., temperature, price). Option B is incorrect because blood type isn't a numerical value.

Binary classification is appropriate when the target variable has only two possible outcomes (e.g., yes/no, true/false). Since there are more than two blood types, binary classification (Option C) isn't appropriate.

K-Nearest Neighbors (KNN) can be used for both classification and regression. However, Option D mentions a "multi-class target attribute," which is redundant. Furthermore, while KNN can be used, it's not inherently the most appropriate choice without further context. It requires careful selection of the 'K' value and distance metric, and doesn't automatically imply handling multiple classes as effectively as a multi-class classification model explicitly designed for the task. More suitable algorithms such as Random Forest or XGBoost are typically employed with multi-class problems.

Multi-class classification is designed for situations where the target variable can take on more than two categorical values. The target attribute (blood type) is categorical because it represents distinct categories. Therefore, a multi-class classification model is the most logical and direct approach.

In Amazon Machine Learning, you would choose a multi-class classification algorithm and specify the blood type column as the target attribute. This allows the model to learn the relationships between the clinical tests (features) and the different blood type categories. The model will then predict the most probable blood type based on new patient data. Using the right model type based on the target value's nature will maximize performance in this situation.

For further reading on classification algorithms:

Amazon SageMaker JumpStart (which uses ML algorithms under the hood):
https://aws.amazon.com/sagemaker/jumpstart/
Scikit-learn documentation on classification:https://scikit-learn.org/stable/modules/classes.html#classification

Question: 24

A data engineer is running a DWH on a 25-node Redshift cluster of a SaaS service. The data engineer needs to build a dashboard that will be used by customers. Five big customers represent 80% of usage, and there is a long tail of dozens of smaller customers. The data engineer has selected the dashboarding tool.
How should the data engineer make sure that the larger customer workloads do NOT interfere with the smaller customer workloads?

A.Apply query filters based on customer-id that can NOT be changed by the user and apply distribution keys on customer-id.

B.Place the largest customers into a single user group with a dedicated query queue and place the rest of the customers into a different query queue.

C.Push aggregations into an RDS for Aurora instance. Connect the dashboard application to Aurora rather than Redshift for faster queries.

D.Route the largest customers to a dedicated Redshift cluster. Raise the concurrency of the multi-tenant Redshift cluster to accommodate the remaining customers.

Answer: B

Explanation:

Here's a detailed justification for why option B is the best approach and why the other options are less suitable:

Option B, placing large customers into a dedicated query queue and smaller customers into another, effectively implements workload management within Redshift. Redshift's Workload Management (WLM) allows you to prioritize queries based on defined rules. By assigning larger customers to a dedicated queue, you can control the resources allocated to their queries, such as memory and concurrency slots. This prevents their resource-intensive queries from consuming all available resources and slowing down the queries of smaller customers. This approach guarantees a degree of isolation and ensures that all customers receive a reasonable level of performance. Query queues can be configured to prioritize shorter queries which benefit the smaller customers who likely run more ad-hoc reports. This provides a more predictable experience for all users. https://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html

Option A, while using customer_id for filtering and distribution, doesn't solve the problem of resource contention. Distribution keys affect data placement, not query execution priority. Applying unchangeable filters improves security and enforces data access restrictions, but doesn't control how Redshift schedules and executes queries from different customers concurrently. This would lead to large customers potentially overwhelming resources.

Option C, pushing aggregations to Aurora, might improve dashboard speed, but it's a premature optimization. The initial problem is workload interference, not necessarily query performance for the smaller customers. Migrating aggregations could also add complexity to the data pipeline and may not fully address the core issue of resource contention within Redshift during complex reporting or ad hoc querying by larger customers. Also, aurora might not be performant enough compared to redshift to handle that size of aggregation.

Option D, routing larger customers to a dedicated Redshift cluster, is the most expensive and complex solution. While it provides complete isolation, it involves setting up and managing another Redshift cluster, incurring additional costs and operational overhead. This level of isolation might be overkill at this stage, especially when workload management can provide sufficient isolation at a fraction of the cost and effort. Increasing the concurrency of the multi-tenant cluster does help, but does not isolate the workloads and can cause queries to spill to disk.

Therefore, option B is the most balanced approach. It utilizes Redshift's built-in WLM capabilities to provide adequate workload isolation without incurring significant costs or operational complexity. It offers a practical way to ensure that larger customer workloads don't negatively impact the performance experienced by smaller customers within the shared Redshift cluster.

Question: 25

An Amazon Kinesis stream needs to be encrypted.
Which approach should be used to accomplish this task?

A.Perform a client-side encryption of the data before it enters the Amazon Kinesis stream on the producer.

B.Use a partition key to segment the data by MD5 hash function, which makes it undecipherable while in transit.
C.Perform a client-side encryption of the data before it enters the Amazon Kinesis stream on the consumer.
D.Use a shard to segment the data, which has built-in functionality to make it indecipherable while in transit.

Answer: A

Explanation:

The correct approach for encrypting data in an Amazon Kinesis stream is to perform client-side encryption before the data enters the stream on the producer side (Option A). Here's why:

Data Security Best Practice: Encryption should ideally happen as close to the source as possible. Encrypting on the producer ensures that the data is protected before it even enters the Kinesis stream, safeguarding it during transmission and storage within Kinesis.

Kinesis Does Not Automatically Encrypt Data at Rest (Without Explicit Configuration): While Kinesis offers server-side encryption using KMS keys, encrypting data before it enters the stream adds an extra layer of security and control. It's particularly useful if you need to meet specific compliance requirements or handle sensitive data.

Partition Keys and Shards Do Not Encrypt Data: Partition keys (Option B) are used for data distribution across shards within a Kinesis stream. Shards (Option D) are the basic units of throughput. Neither of these concepts inherently provides encryption. Using an MD5 hash (as suggested in Option B) would be suitable for data distribution and might even provide basic obfuscation, but it isn't a secure encryption method and isn't intended for that purpose. It is easily reversed.

Consumer-Side Encryption is Insufficient: Encrypting only on the consumer side (Option C) leaves the data vulnerable while in transit to the Kinesis stream and while at rest in the Kinesis stream (if you haven't configured server-side encryption). The data is only protected after it is retrieved by the consumer.

End-to-End Encryption: Producer-side encryption, paired with consumer-side decryption, allows for true end-to-end encryption.

Flexibility: Client-side encryption allows you to choose the encryption algorithm and manage the encryption keys according to your own security policies.

Kinesis Server Side Encryption: If you are encrypting on the client side, you may still choose to use Kinesis server side encryption to add another layer of security.

Authoritative Links:

Amazon Kinesis Data Streams Encryption:https://docs.aws.amazon.com/streams/latest/dev/data-protection.html
AWS KMS Best Practices:https://docs.aws.amazon.com/kms/latest/developerguide/best-practices.html

Question: 26

An online photo album app has a key design feature to support multiple screens (e.g, desktop, mobile phone, and tablet) with high-quality displays. Multiple versions of the image must be saved in different resolutions and layouts.
The image-processing Java program takes an average of five seconds per upload, depending on the image size and format.
Each image upload captures the following image metadata: user, album, photo label, upload timestamp.
The app should support the following requirements:
➩ Hundreds of user image uploads per second
➩ Maximum image upload size of 10 MB
➩ Maximum image metadata size of 1 KB
➩ Image displayed in optimized resolution in all supported screens no later than one minute after image upload

Which strategy should be used to meet these requirements?

A.Write images and metadata to Amazon Kinesis. Use a Kinesis Client Library (KCL) application to run the image processing and save the image output to Amazon S3 and metadata to the app repository DB.

B.Write image and metadata RDS with BLOB data type. Use AWS Data Pipeline to run the image processing and save the image output to Amazon S3 and metadata to the app repository DB.

C.Upload image with metadata to Amazon S3, use Lambda function to run the image processing and save the images output to Amazon S3 and metadata to the app repository DB.

D.Write image and metadata to Amazon Kinesis. Use Amazon Elastic MapReduce (EMR) with Spark Streaming to run image processing and save the images output to Amazon S3 and metadata to app repository DB.

Answer: C

Explanation:

Here's a detailed justification for choosing option C, focusing on its suitability for the specified requirements:

Option C, "Upload image with metadata to Amazon S3, use Lambda function to run the image processing and save the images output to Amazon S3 and metadata to the app repository DB," is the most suitable solution for the given scenario due to its scalability, cost-effectiveness, and simplicity.

Storing the original images in Amazon S3 provides durable and scalable object storage. S3 is designed for high availability and can handle a large volume of image uploads efficiently. (https://aws.amazon.com/s3/)

Using AWS Lambda for image processing offers several advantages. Lambda functions are event-driven, meaning they automatically trigger when a new image is uploaded to S3. This eliminates the need for constantly running servers or complex orchestration. Lambda's serverless nature allows it to scale automatically to handle hundreds of uploads per second without manual intervention.
(https://aws.amazon.com/lambda/)

The processing time requirement of one minute can be easily met by Lambda given the five-second average processing time per image. Lambda allows increasing memory allocation to speed up processing as well as concurrency scaling.

Furthermore, Lambda can save the processed images (in different resolutions) back to S3 and store the metadata in a separate repository database (e.g., Amazon RDS or DynamoDB). This separation of concerns –images in S3 and metadata in a database – is a best practice for application design.
(https://aws.amazon.com/rds/)

Compared to other options, Option A using Kinesis and KCL would be overkill for simple image processing.
While Kinesis is great for real-time data streaming, it introduces unnecessary complexity for this use case.
Option B using RDS with BLOB data type can negatively affect the performance of database and AWS Data Pipeline is more suited for batch processing. Option D involving Amazon EMR with Spark Streaming is also far more complex and costly than necessary, as it is designed for very large-scale data processing. Option C avoids the complexities of Kinesis and EMR while providing adequate processing power and scalability with Lambda. Therefore, Lambda triggered by S3 events offers a simpler, more cost-effective, and scalable solution that directly addresses the stated requirements.