Free Amazon AIP-C01 Practice Questions

Shared by Garza on 12-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

An ecommerce company is using Amazon Bedrock to build a generative AI (GenAI) application. The application uses AWS Step Functions to orchestrate a multi-agent workflow to produce detailed product descriptions. The workflow consists of three sequential states: a description generator, a technical specifications validator, and a brand voice consistency checker. Each state produces intermediate reasoning traces and outputs that are passed to the next state. The application uses an Amazon S3 bucket for process storage and to store outputs. During testing, the company discovers that outputs between Step Functions states frequently exceed the 256 KB quota and cause workflow failures. A GenAI Developer needs to revise the application architecture to efficiently handle the Step Functions 256 KB quota and maintain workflow observability. The revised architecture must preserve the existing multi-agent reasoning and acting (ReAct) pattern. Which solution will meet these requirements with the LEAST operational overhead?

Options:

A- Store intermediate outputs in Amazon DynamoDB. Pass only references between states. Create a Map state that retrieves the complete data from DynamoDB when required for each agent's processing step.
B- Configure an Amazon Bedrock integration to use the S3 bucket URI in the input parameters for large outputs. Use the ResultPath and ResultSelector fields to route S3 references between the agent steps while maintaining the sequential validation workflow.
C- Use AWS Lambda functions to compress outputs to less than 256 KB before each agent state. Configure each agent task to decompress outputs before processing and to compress results before passing them to the next state.
D- Configure a separate Step Functions state machine to handle each agent's processing. Use Amazon EventBridge to coordinate the execution flow between state machines. Use S3 references for the outputs as event data.

Answer:

B

Explanation:

Option B is the best solution because it directly addresses the Step Functions 256 KB state payload quota by externalizing large intermediate artifacts to Amazon S3 and passing only lightweight references (URIs/keys) between states. This is a standard AWS pattern for workflows that produce large intermediate results, and it avoids introducing additional databases, compression logic, or cross-state-machine coordination that increases operational overhead.

In a multi-agent ReAct workflow, intermediate reasoning traces can be verbose and grow

quickly as each agent produces chain-of-thought style artifacts, structured outputs, and supporting evidence. Step Functions is designed to orchestrate state transitions and pass JSON payloads, but large payloads should be stored outside the state machine and referenced by pointer values. Using Amazon S3 for intermediate outputs is operationally efficient because the application already uses S3 for storage, and S3 provides durable, low-cost storage with simple access patterns.

ResultPath and ResultSelector allow each state to store or reshape results so that only the required reference fields (such as s3Uri, object key, metadata, trace IDs) are forwarded to subsequent states. This preserves observability because the workflow can still log trace references, correlate steps with S3 objects, and store structured metadata for debugging. It also preserves the sequential validation design, keeping the existing ReAct pattern intact while preventing failures due to oversized payloads.

Option A adds additional services and read/write patterns that increase operational complexity. Option C introduces custom compression/decompression logic that is fragile, adds latency, and complicates troubleshooting. Option D increases orchestration overhead by splitting workflows and coordinating with events, which makes debugging harder and increases failure modes.

Therefore, Option B meets the payload limit requirement while keeping the architecture simple and observable.

Question 2

Question Type: MultipleChoice

A company provides a service that helps users from around the world discover new restaurants. The service has 50 million monthly active users. The company wants to implement a semantic search solution across a database that contains 20 million restaurants and 200 million reviews. The company currently stores the data in PostgreSQL.

The solution must support complex natural language queries and return results for at least 95% of queries within 500 ms. The solution must maintain data freshness for restaurant details that update hourly. The solution must also scale cost-effectively during peak usage periods.

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

Options:

A- Migrate the restaurant data to Amazon OpenSearch Service. Implement keyword-based search rules that use custom analyzers and relevance tuning to find restaurants based on attributes such as cuisine type, features, and location. Create Amazon API Gateway HTTP API endpoints to transform user queries into structured search parameters.

B- Migrate the restaurant data to Amazon OpenSearch Service. Use a foundation model (FM) in Amazon Bedrock to generate vector embeddings from restaurant descriptions, reviews, and

menu items. When users submit natural language queries, convert the queries to embeddings by using the same FM. Perform k-nearest neighbors (k-NN) searches to find semantically similar results.
C- Keep the restaurant data in PostgreSQL and implement a pgvector extension. Use a foundation model (FM) in Amazon Bedrock to generate vector embeddings from restaurant data. Store the vector embeddings directly in PostgreSQL. Create an AWS Lambda function to convert natural language queries to vector representations by using the same FM. Configure the Lambda function to perform similarity searches within the database.
D- Migrate restaurant data to an Amazon Bedrock knowledge base by using a custom ingestion pipeline. Configure the knowledge base to automatically generate embeddings from restaurant information. Use the Amazon Bedrock Retrieve API with built-in vector search capabilities to query the knowledge base directly by using natural language input.

Answer:

B

Explanation:

Option B best satisfies the requirements while minimizing development effort by combining managed semantic search capabilities with fully managed foundation models. AWS Generative AI guidance describes semantic search as a vector-based retrieval pattern where both documents and user queries are embedded into a shared vector space. Similarity search (such as k-nearest neighbors) then retrieves results based on meaning rather than exact keywords.

Amazon OpenSearch Service natively supports vector indexing and k-NN search at scale. This makes it well suited for large datasets such as 20 million restaurants and 200 million reviews while still achieving sub-second latency for the majority of queries. Because OpenSearch is a distributed, managed service, it automatically scales during peak traffic periods and provides cost-effective performance compared with building and tuning custom vector search pipelines on relational databases.

Using Amazon Bedrock to generate embeddings significantly reduces development complexity. AWS manages the foundation models, eliminates the need for custom model hosting, and ensures consistency by using the same FM for both document embeddings and query embeddings. This aligns directly with AWS-recommended semantic search architectures and removes the need for model lifecycle management.

Hourly updates to restaurant data can be handled efficiently through incremental re-indexing in OpenSearch without disrupting query performance. This approach cleanly separates transactional data storage from search workloads, which is a best practice in AWS architectures.

Option A does not meet the semantic search requirement because keyword-based search cannot reliably interpret complex natural language intent. Option C introduces scalability and performance risks by running large-scale vector similarity searches inside PostgreSQL, which increases operational complexity. Option D adds unnecessary ingestion and abstraction layers intended for retrieval-augmented generation, not high-throughput semantic search.

Therefore, Option B provides the optimal balance of performance, scalability, data freshness,

and minimal development effort using AWS Generative AI services.

Question 3

Question Type: MultipleChoice

A medical company uses Amazon Bedrock to power a clinical documentation summarization system. The system produces inconsistent summaries when handling complex clinical documents. The system performed well on simple clinical documents.

The company needs a solution that diagnoses inconsistencies, compares prompt performance against established metrics, and maintains historical records of prompt versions.

Which solution will meet these requirements?

Options:

A- Create multiple prompt variants by using Prompt management in Amazon Bedrock. Manually test the prompts with simple clinical documents. Deploy the highest performing version by using the Amazon Bedrock console.
B- Implement version control for prompts in a code repository with a test suite that contains complex clinical documents and quantifiable evaluation metrics. Use an automated testing framework to compare prompt versions and document performance patterns.
C- Deploy each new prompt version to separate Amazon Bedrock API endpoints. Split production traffic between the endpoints. Configure Amazon CloudWatch to capture response metrics and user feedback for automatic version selection.
D- Create a custom prompt evaluation flow in Amazon Bedrock Flows that applies the same clinical document inputs to different prompt variants. Use Amazon Comprehend Medical to analyze and score the factual accuracy of each version.

Answer:

B

Explanation:

Option B best meets the requirements because it provides systematic diagnosis, measurable comparison, and historical traceability of prompt performance. By placing prompts under version control and testing them against complex clinical documents, the company can consistently reproduce issues, track regressions, and compare prompt behavior using quantifiable metrics such as factual accuracy, completeness, and consistency. Automated testing ensures scalability and repeatability, while version history preserves prompt evolution over time.

Option A lacks objective metrics and does not address complex documents. Option C focuses on live traffic experimentation but does not inherently diagnose prompt inconsistencies or preserve detailed historical evaluations. Option D adds medical entity analysis but introduces unnecessary service coupling and does not provide robust prompt version history or automated comparative benchmarking. Therefore, Option B is the most complete and disciplined solution.

Question 4

Question Type: MultipleChoice

A company has a customer service application that uses Amazon Bedrock to generate personalized responses to customer inquiries. The company needs to establish a quality assurance process to evaluate prompt effectiveness and model configurations across updates. The process must automatically compare outputs from multiple prompt templates, detect response quality issues, provide quantitative metrics, and allow human reviewers to give feedback on responses. The process must prevent configurations that do not meet a predefined quality threshold from being deployed.

Which solution will meet these requirements?

Options:

A- Create an AWS Lambda function that sends sample customer inquiries to multiple Amazon Bedrock model configurations and stores responses in Amazon S3. Use Amazon QuickSight to visualize response patterns. Manually review outputs daily. Use AWS CodePipeline to deploy configurations that meet the quality threshold.

B- Use Amazon Bedrock evaluation jobs to compare model outputs by using custom prompt datasets. Configure AWS CodePipeline to run the evaluation jobs when prompt templates change. Configure CodePipeline to deploy only configurations that exceed the predefined quality threshold.

C- Set up Amazon CloudWatch alarms to monitor response latency and error rates from Amazon Bedrock. Use Amazon EventBridge rules to notify teams when thresholds are exceeded. Configure a manual approval workflow in AWS Systems Manager.

D- Use AWS Lambda functions to create an automated testing framework that samples production traffic and routes duplicate requests to the updated model version. Use Amazon Comprehend sentiment analysis to compare results. Block deployment if sentiment scores decrease.

Answer:

B

Explanation:

Option B is the correct solution because Amazon Bedrock evaluation jobs are purpose-built to assess prompt effectiveness, model behavior, and response quality in a repeatable and automated manner. Evaluation jobs support both quantitative metrics and LLM-based judgment, making them suitable for detecting subtle response quality regressions that simple sentiment or latency metrics cannot capture.

By using custom prompt datasets, the company can consistently test multiple prompt templates and model configurations against the same inputs. This enables accurate comparison across updates and eliminates variability introduced by live traffic sampling. Amazon Bedrock evaluation jobs also support structured scoring outputs, which can be used to enforce objective quality thresholds.

Integrating evaluation jobs directly into AWS CodePipeline ensures that quality checks are automatically triggered whenever prompt templates or configurations change. This creates a gated deployment workflow in which only configurations that meet or exceed the predefined quality threshold are promoted. This directly satisfies the requirement to prevent low-quality configurations from being deployed.

Human reviewers can be incorporated by reviewing evaluation results and scores produced by the jobs, enabling informed feedback without manual data collection. Option A and D rely on custom frameworks and indirect quality signals, increasing complexity and reducing reliability. Option C focuses on operational health rather than response quality.

Therefore, Option B provides the most robust, scalable, and AWS-aligned quality assurance process for Amazon Bedrock--based applications.

Question 5

Question Type: MultipleChoice

A company is developing a customer support application that uses Amazon Bedrock foundation models (FMs) to provide real-time AI assistance to the company's employees. The application must display AI-generated responses character by character as the responses are generated. The application needs to support thousands of concurrent users with minimal latency. The responses typically take 15 to 45 seconds to finish.

Which solution will meet these requirements?

Options:

A- Configure an Amazon API Gateway WebSocket API with an AWS Lambda integration. Configure the WebSocket API to invoke the Amazon Bedrock InvokeModelWithResponseStream API and stream partial responses through WebSocket connections.
B- Configure an Amazon API Gateway REST API with an AWS Lambda integration. Configure the REST API to invoke the Amazon Bedrock standard InvokeModel API and implement frontend

client-side polling every 100 ms for complete response chunks.
C- Implement direct frontend client connections to Amazon Bedrock by using IAM user credentials and the InvokeModelWithResponseStream API without any intermediate gateway or proxy layer.
D- Configure an Amazon API Gateway HTTP API with an AWS Lambda integration. Configure the HTTP API to cache complete responses in an Amazon DynamoDB table and serve the responses through multiple paginated GET requests to frontend clients.

Answer:

A

Explanation:

This requirement explicitly calls for character-by-character streaming, long-running responses, low latency, and massive concurrency, which aligns directly with Amazon Bedrock streaming inference patterns.

Amazon Bedrock provides the InvokeModelWithResponseStream API specifically for streaming partial model outputs as tokens are generated. This enables near-instant feedback to users instead of waiting for the full response to complete, which is essential when responses last up to 45 seconds.

Amazon API Gateway WebSocket APIs are purpose-built for bidirectional, low-latency, server-initiated communication, allowing the backend to push characters or tokens to clients in real time. This eliminates inefficient polling and supports thousands of concurrent open connections.

AWS Lambda integrates natively with WebSocket APIs and scales automatically with connection volume, enabling a fully managed, serverless architecture. This approach maintains security, centralized authentication, throttling, and observability while avoiding direct client access to Bedrock APIs.

Option B introduces polling latency and unnecessary API overhead and does not provide true streaming. Option C violates AWS security best practices by exposing Bedrock directly to clients and does not scale securely. Option D only serves completed responses and cannot meet the real-time streaming requirement.

Therefore, Option A is the only solution that fully satisfies streaming behavior, concurrency, latency, and managed-service constraints.

Question 6

Question Type: MultipleChoice

A healthcare company is using Amazon Bedrock to develop a real-time patient care AI assistant

to respond to queries for separate departments that handle clinical inquiries, insurance verification, appointment scheduling, and insurance claims. The company wants to use a multi-agent architecture.

The company must ensure that the AI assistant is scalable and can onboard new features for patients. The AI assistant must be able to handle thousands of parallel patient interactions. The company must ensure that patients receive appropriate domain-specific responses to queries.

Which solution will meet these requirements?

Options:

A- Isolate data for each agent by using separate knowledge bases. Use IAM filtering to control access to each knowledge base. Deploy a supervisor agent to perform natural language intent classification on patient inquiries. Configure the supervisor agent to route queries to specialized collaborator agents to respond to department-specific queries. Configure each specialized collaborator agent to use Retrieval Augmented Generation (RAG) with the agent's department-specific knowledge base.
B- Create a separate supervisor agent for each department. Configure individual collaborator agents to perform natural language intent classification for each specialty domain within each department. Integrate each collaborator agent with department-specific knowledge bases only. Implement manual handoff processes between the supervisor agents.
C- Isolate data for each department in separate knowledge bases. Use IAM filtering to control access to each knowledge base. Deploy a single general-purpose agent. Configure multiple action groups within the general-purpose agent to perform specific department functions. Implement rule-based routing logic in the general-purpose agent instructions.
D- Implement multiple independent supervisor agents that run in parallel to respond to patient inquiries for each department. Configure multiple collaborator agents for each supervisor agent. Integrate all agents with the same knowledge base. Use external routing logic to merge responses from multiple supervisor agents.

Answer:

A

Explanation:

Option A best meets the requirements because it applies an AWS-aligned multi-agent pattern that cleanly separates responsibilities: a supervisor agent performs intent classification and orchestration, while specialized collaborator agents handle domain-specific tasks using the right knowledge sources. This structure is well suited for healthcare workflows where clinical questions, scheduling, and insurance processes require different policies, terminology, and data access boundaries.

The requirement for appropriate domain-specific responses is addressed by routing each user query to a department-focused collaborator agent that is grounded with its own department-specific knowledge base. Using Retrieval Augmented Generation with the correct knowledge base improves factual alignment and reduces cross-department leakage (for example, avoiding

claims content in a clinical answer). It also supports better prompt grounding and more consistent tone and constraints per department.

The requirement to isolate data maps to using separate knowledge bases per agent and enforcing access through IAM controls, ensuring that each agent can retrieve only from the authorized datasets. This is important for minimizing unintended exposure of sensitive or irrelevant departmental data and supports governance and compliance needs.

For scalability and thousands of parallel interactions, this architecture minimizes contention and bottlenecks. Each collaborator agent can scale independently because requests are distributed across multiple agents and multiple retrieval backends. Operationally, onboarding new features is also simpler: the company can add a new collaborator agent (for example, ''billing disputes'' or ''pharmacy refills'') with its own knowledge base and policies without redesigning the entire assistant.

Option B introduces unnecessary complexity with multiple supervisors and manual handoffs. Option C overloads a single agent with broad instructions and rule-based routing, which increases prompt complexity and reduces maintainability as features grow. Option D creates high operational complexity and risks inconsistent outputs when merging responses from parallel supervisors, and it weakens data isolation by using a shared knowledge base across agents.

Question 7

Question Type: MultipleChoice

A book publishing company wants to build a book recommendation system that uses an AI assistant. The AI assistant will use ML to generate a list of recommended books from the company's book catalog. The system must suggest books based on conversations with customers.

The company stores the text of the books, customers' and editors' reviews of the books, and extracted book metadata in Amazon S3. The system must support low-latency responses and scale efficiently to handle more than 10,000 concurrent users.

Which solution will meet these requirements?

Options:

A- Use Amazon Bedrock Knowledge Bases to generate embeddings. Store the embeddings as a vector store in Amazon OpenSearch Service. Create an AWS Lambda function that queries the knowledge base. Configure Amazon API Gateway to invoke the Lambda function when handling user requests.

B- Use Amazon Bedrock Knowledge Bases to generate embeddings. Store the embeddings as a vector store in Amazon DynamoDB. Create an AWS Lambda function that queries the knowledge

base. Configure Amazon API Gateway to invoke the Lambda function when handling user requests.
C- Use Amazon SageMaker AI to deploy a pre-trained model to build a personalized recommendation engine for books. Deploy the model as a SageMaker AI endpoint. Invoke the model endpoint by using Amazon API Gateway.
D- Create an Amazon Kendra GenAI Enterprise Edition index that uses the S3 connector to index the book catalog data stored in Amazon S3. Configure built-in FAQ in the Kendra index. Develop an AWS Lambda function that queries the Kendra index based on user conversations. Deploy Amazon API Gateway to expose this functionality and invoke the Lambda function.

Answer:

A

Explanation:

Option A best meets the requirements because it directly implements a Retrieval Augmented Generation pattern for conversational recommendations using managed Amazon Bedrock capabilities and a scalable vector store. The company's source data already resides in Amazon S3, which aligns naturally with Amazon Bedrock Knowledge Bases ingestion workflows. A knowledge base can ingest book text, reviews, and metadata, generate embeddings using a supported embedding model, and persist those vectors in a purpose-built vector backend such as Amazon OpenSearch Service. This enables semantic retrieval that is well suited to conversation-driven intent, where user prompts are often descriptive and do not map cleanly to keyword filters.

The requirement to suggest books based on conversations implies the system must interpret natural language context and retrieve relevant passages, reviews, and metadata to ground the recommendation. Knowledge Bases provide managed orchestration for embedding creation and retrieval, which reduces development effort compared to building custom embedding pipelines. OpenSearch Service provides scalable vector search and k-nearest neighbors style similarity retrieval, which supports low-latency responses when properly indexed and sized.

For scaling to more than 10,000 concurrent users, the API layer design in option A is a common AWS pattern: Amazon API Gateway provides a managed front door with throttling and request handling, while AWS Lambda scales horizontally with demand and can invoke the knowledge base retrieval operations. This separates compute scaling from the vector store scaling and helps keep latency predictable under load.

Option B is not the best choice because DynamoDB is not the standard native vector store target for Amazon Bedrock Knowledge Bases in this context and would introduce additional implementation complexity around vector indexing and similarity search behavior. Option C requires substantial ML lifecycle work, model hosting, tuning, and continuous iteration to achieve quality recommendations at scale. Option D provides strong enterprise search, but it focuses on retrieval and FAQs rather than a managed RAG recommendation workflow grounded in embeddings and conversational context for generative responses.

Question 8

Question Type: MultipleChoice

A retail company is using Amazon Bedrock to develop a customer service AI assistant. Analysis shows that 70% of customer inquiries are simple product questions that a smaller model can effectively handle. However, 30% of inquiries are complex return policy questions that require advanced reasoning.

The company wants to implement a cost-effective model selection framework to automatically route customer inquiries to appropriate models based on inquiry complexity. The framework must maintain high customer satisfaction and minimize response latency.

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

Options:

A- Create a multi-stage architecture that uses a small foundation model (FM) to classify the complexity of each inquiry. Route simple inquiries to a smaller, more cost-effective model. Route complex inquiries to a larger, more capable model. Use AWS Lambda functions to handle routing logic.
B- Use Amazon Bedrock intelligent prompt routing to automatically analyze inquiries. Route simple product inquiries to smaller models and route complex return policy inquiries to more capable larger models.
C- Implement a single-model solution that uses an Amazon Bedrock mid-sized foundation model (FM) with on-demand pricing. Include special instructions in model prompts to handle both simple and complex inquiries by using the same model.
D- Create separate Amazon Bedrock endpoints for simple and complex inquiries. Implement a rule-based routing system based on keyword detection. Use on-demand pricing for the smaller model and provisioned throughput for the larger model.

Answer:

B

Explanation:

Option B is the correct solution because it leverages native Amazon Bedrock intelligent prompt routing, which is specifically designed to reduce cost and complexity in multi-model GenAI architectures. Intelligent prompt routing automatically analyzes incoming prompts and selects the most appropriate foundation model based on prompt characteristics and complexity---without requiring custom classification logic or orchestration code.

This approach directly meets the requirement for least implementation effort. The company does not need to deploy additional Lambda functions, maintain routing rules, or manage separate classification stages. Routing decisions are handled by Bedrock, which simplifies

architecture and reduces operational risk.

By routing the majority (70%) of simple product inquiries to smaller, lower-cost models, the company minimizes inference cost and latency. More complex return policy inquiries are automatically routed to larger models that provide better reasoning capabilities, preserving response quality and customer satisfaction.

Because routing is handled inline by Bedrock, response latency remains low compared to multi-stage architectures that require an additional classification model call before inference. This is critical for customer service scenarios where responsiveness directly impacts satisfaction.

Option A introduces additional inference steps and custom logic. Option C increases cost by overusing a mid-sized model for all queries. Option D relies on brittle keyword rules and increases operational overhead through endpoint management.

Therefore, Option B delivers the optimal balance of cost efficiency, performance, and simplicity for dynamic model selection in Amazon Bedrock.

Question 9

Question Type: MultipleChoice

A company has a customer service application that uses Amazon Bedrock to generate personalized responses to customer inquiries. The company needs to establish a quality assurance process to evaluate prompt effectiveness and model configurations across updates. The process must automatically compare outputs from multiple prompt templates, detect response quality issues, provide quantitative metrics, and allow human reviewers to give feedback on responses. The process must prevent configurations that do not meet a predefined quality threshold from being deployed.

Which solution will meet these requirements?

Options:

A- Create an AWS Lambda function that sends sample customer inquiries to multiple Amazon Bedrock model configurations and stores responses in Amazon S3. Use Amazon QuickSight to visualize response patterns. Manually review outputs daily. Use AWS CodePipeline to deploy configurations that meet the quality threshold.

B- Use Amazon Bedrock evaluation jobs to compare model outputs by using custom prompt datasets. Configure AWS CodePipeline to run the evaluation jobs when prompt templates change. Configure CodePipeline to deploy only configurations that exceed the predefined quality threshold.

C- Set up Amazon CloudWatch alarms to monitor response latency and error rates from Amazon Bedrock. Use Amazon EventBridge rules to notify teams when thresholds are exceeded. Configure a manual approval workflow in AWS Systems Manager.

D- Use AWS Lambda functions to create an automated testing framework that samples production traffic and routes duplicate requests to the updated model version. Use Amazon Comprehend sentiment analysis to compare results. Block deployment if sentiment scores decrease.

Answer:

B

Explanation:

Option B is the correct solution because Amazon Bedrock evaluation jobs are purpose-built to assess prompt effectiveness, model behavior, and response quality in a repeatable and automated manner. Evaluation jobs support both quantitative metrics and LLM-based judgment, making them suitable for detecting subtle response quality regressions that simple sentiment or latency metrics cannot capture.

By using custom prompt datasets, the company can consistently test multiple prompt templates and model configurations against the same inputs. This enables accurate comparison across updates and eliminates variability introduced by live traffic sampling. Amazon Bedrock evaluation jobs also support structured scoring outputs, which can be used to enforce objective quality thresholds.

Integrating evaluation jobs directly into AWS CodePipeline ensures that quality checks are automatically triggered whenever prompt templates or configurations change. This creates a gated deployment workflow in which only configurations that meet or exceed the predefined quality threshold are promoted. This directly satisfies the requirement to prevent low-quality configurations from being deployed.

Human reviewers can be incorporated by reviewing evaluation results and scores produced by the jobs, enabling informed feedback without manual data collection. Option A and D rely on custom frameworks and indirect quality signals, increasing complexity and reducing reliability. Option C focuses on operational health rather than response quality.

Therefore, Option B provides the most robust, scalable, and AWS-aligned quality assurance process for Amazon Bedrock--based applications.

Question 10

Question Type: MultipleChoice

A university recently digitized a collection of archival documents, academic journals, and manuscripts. The university stores the digital files in an AWS Lake Formation data lake.

The university hires a GenAI developer to build a solution to allow users to search the digital

files by using text queries. The solution must return journal abstracts that are semantically similar to a user's query. Users must be able to search the digitized collection based on text and metadata that is associated with the journal abstracts. The metadata of the digitized files does not contain keywords. The solution must match similar abstracts to one another based on the similarity of their text. The data lake contains fewer than 1 million files.

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

Options:

A- Use Amazon Titan Embeddings in Amazon Bedrock to create vector representations of the digitized files. Store embeddings in the OpenSearch Neural plugin for Amazon OpenSearch Service.
B- Use Amazon Comprehend to extract topics from the digitized files. Store the topics and file metadata in an Amazon Aurora PostgreSQL database. Query the abstract metadata against the data in the Aurora database.
C- Use Amazon SageMaker AI to deploy a sentence-transformer model. Use the model to create vector representations of the digitized files. Store embeddings in an Amazon Aurora PostgreSQL database that has the pgvector extension.
D- Use Amazon Titan Embeddings in Amazon Bedrock to create vector representations of the digitized files. Store embeddings in an Amazon Aurora PostgreSQL Serverless database that has the pgvector extension.

Answer:

D

Explanation:

Option D is the best choice because it delivers true semantic search with the smallest operational footprint by combining a fully managed embedding service with an automatically scaling vector-capable database. The university's requirement is explicitly semantic: the metadata has no keywords, and the system must match abstracts based on similarity of meaning. This is a direct fit for an embeddings-based approach, where each abstract is converted into a vector representation and searched using vector similarity. Amazon Titan Embeddings in Amazon Bedrock provides a managed way to generate these vectors without hosting or maintaining an ML model, eliminating the operational work of model provisioning, patching, scaling, and lifecycle management.

For storage and retrieval, Amazon Aurora PostgreSQL Serverless with the pgvector extension supports vector storage and similarity search while minimizing infrastructure operations. Aurora Serverless reduces capacity planning and scaling tasks because it can automatically adjust to changes in workload, which is valuable for a university search application with variable usage patterns. With fewer than 1 million files, a PostgreSQL-based vector store is commonly operationally simpler than running a dedicated search cluster, while still meeting the requirement to query using both text-derived similarity and associated metadata filters stored alongside the vectors.

Option A can also enable vector search, but operating an OpenSearch domain typically introduces additional concerns such as domain sizing, shard strategy, cluster scaling, and performance tuning for k-NN workloads. Option C increases operational overhead the most because it requires deploying and operating a sentence-transformer model endpoint in SageMaker AI, including scaling, monitoring, and model management. Option B does not meet the semantic similarity requirement reliably because topic extraction is not equivalent to embedding-based semantic matching, especially when the metadata lacks keywords and the system must compare abstracts by meaning.

Therefore, D best satisfies semantic search needs with the least operational overhead.

Question 11

Question Type: MultipleChoice

A financial services company is developing a generative AI (GenAI) application that serves both premium customers and standard customers. The application uses AWS Lambda functions behind an Amazon API Gateway REST API to process requests. The company needs to dynamically switch between AI models based on which customer tier each user belongs to. The company also wants to perform A/B testing for new features without redeploying code. The company needs to validate model parameters like temperature and maximum token limits before applying changes.

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

Options:

A- Create AWS Systems Manager Parameter Store parameters for each configuration. Use Lambda functions to poll for parameter updates. Use Amazon EventBridge events to trigger redeployments when configurations change.
B- Store model configurations in Amazon DynamoDB tables. Optimize access patterns to retrieve configurations according to customer tier. Configure Lambda functions to query DynamoDB at the beginning of each request to determine which model to use.
C- Use AWS AppConfig to manage model configurations. Use feature flags to perform A/B testing. Define JSON schema validation rules for model parameters. Configure Lambda functions to retrieve configurations by using the AWS AppConfig Agent.
D- Create an Amazon ElastiCache (Redis OSS) cluster to store model configurations. Set short TTL values. Run custom validation logic in Lambda functions. Use Amazon CloudWatch metrics to monitor configuration usage.

Answer:

C

Explanation:

Option C is the correct solution because AWS AppConfig is purpose-built to manage dynamic application configurations with low latency, strong validation, and minimal operational overhead, which directly matches the company's requirements.

AWS AppConfig enables the company to centrally manage model selection logic, inference parameters, and customer-tier routing rules without redeploying Lambda functions. By using feature flags, the company can easily perform A/B testing of new models or prompt strategies by gradually rolling out changes to a subset of users or customer tiers. This allows experimentation and controlled releases without code changes.

AppConfig also supports JSON schema validation, which is critical for validating parameters such as temperature, maximum token limits, and other model-specific settings before they are applied. This prevents invalid or unsafe configurations from being deployed and reduces the risk of runtime errors or degraded model behavior in production.

Using the AWS AppConfig Agent allows Lambda functions to retrieve configurations efficiently with built-in caching and polling mechanisms, minimizing latency and avoiding excessive calls to configuration services. This approach scales well for high-throughput, low-latency applications such as GenAI APIs behind Amazon API Gateway.

Option A introduces unnecessary redeployment logic and polling complexity. Option B requires building and maintaining custom configuration access patterns in DynamoDB and does not natively support feature flags or schema validation. Option D adds operational overhead by requiring ElastiCache cluster management and custom validation logic.

Therefore, Option C provides the most scalable, flexible, and low-maintenance solution for dynamic model switching, A/B testing, and safe configuration management in a GenAI application.