DVA-C03 AWS Developer Associate Free Sample (30 Questions)

Unofficial practice material. Not affiliated with or endorsed by Amazon Web Services (AWS).

This free sample contains 30 of the 700 questions from the full DVA-C03 AWS Certified Developer - Associate question bank.

Q1. A developer is building a Lambda function for a billing platform. The function generates PDF invoices and writes them to an S3 bucket. Compliance requires that the function hold no long-term credentials and be able to write only to that one bucket. What is the MOST secure way to configure this?

A) Create an IAM execution role with a policy that allows s3:PutObject only on the invoice bucket and attach the role to the function

B) Create an IAM user for the application, store its access keys in a plaintext environment variable, and rotate them monthly

C) Attach the AmazonS3FullAccess managed policy to the function's execution role so future buckets work without changes

D) Embed the account root credentials in the deployment package so the function always has access

Correct answer: A

Explanation: Correct: An execution role gives the function short-lived credentials through STS, and a policy scoped to s3:PutObject on the single bucket follows least privilege. Why others are wrong: access keys in a plaintext environment variable are long-term credentials that anyone with console or code access can steal; AmazonS3FullAccess grants write access to every bucket in the account, far beyond the requirement; root credentials in the package are never acceptable and expose the entire account.

Q2. A Lambda function that processes orders must also run inside a VPC to reach a private RDS database. After the developer attaches the function to private subnets, calls from the function to the public DynamoDB API start timing out. What should the developer do to restore DynamoDB access with the least operational overhead?

A) Move the function out of the VPC and connect to RDS over the public internet

B) Create a NAT gateway in a public subnet and route the private subnets' internet-bound traffic through it

C) Add a gateway VPC endpoint for DynamoDB to the route tables of the private subnets

D) Give each private subnet a public IP range so the function can reach AWS APIs directly

Correct answer: C

Explanation: Correct: A gateway VPC endpoint for DynamoDB is free, highly available, and routes DynamoDB traffic privately without leaving the AWS network. Why others are wrong: moving the function out of the VPC breaks the requirement to reach the private database and exposes traffic publicly; a NAT gateway works but adds hourly and data-processing cost plus management overhead when an endpoint solves it directly; private subnets cannot simply be given public addressing, and instances/functions in them still need NAT or endpoints.

Q3. A developer needs to share a set of utility libraries and a custom JSON-parsing binary across twelve Lambda functions written by different teams. The shared code changes about once a month and must be deployable independently of each function. Which approach meets these requirements?

A) Copy the libraries into every function's deployment package and redeploy all twelve on each change

B) Package the shared code as a Lambda layer and attach the layer version to each function

C) Store the libraries in an S3 bucket and download them to /tmp on every cold start

D) Merge the twelve functions into one large function that contains all the shared code

Correct answer: B

Explanation: Correct: Lambda layers are designed for sharing code and binaries across functions, are versioned independently, and functions pick up a new layer version when updated. Why others are wrong: copying code into every package duplicates the code and forces twelve redeployments per change; downloading from S3 at cold start adds latency and failure modes and loses the code if /tmp is reclaimed; merging functions destroys team boundaries and independent scaling.

Q4. A retail company's Lambda function validates shopping carts. During flash sales, bursts of invocations overwhelm a downstream inventory API that can only handle 200 concurrent calls. The developer must cap the function's concurrency while still accepting events. Which configuration achieves this?

A) Set the function's reserved concurrency to 200

B) Set the function's provisioned concurrency to 200

C) Lower the function's memory allocation so each invocation runs more slowly

D) Reduce the account-level concurrency quota through a Service Quotas request

Correct answer: A

Explanation: Correct: Reserved concurrency sets a maximum number of simultaneous executions for that function, guaranteeing both a cap and a reserved pool. Why others are wrong: provisioned concurrency pre-warms environments to remove cold starts but does not cap total concurrency; lowering memory slows execution but does not bound simultaneous calls and hurts latency; the account quota affects all functions in the Region and cannot target this function's downstream limit.

Q5. A Java Lambda function that serves a customer-facing API has unacceptable cold-start latency because of heavy framework initialization. The team cannot rewrite the application. Which feature MOST directly reduces the initialization latency?

A) Increase the function's memory allocation to its maximum

B) Enable Lambda SnapStart for the function

C) Move the function to a container image deployment

D) Place the function inside a VPC with enhanced networking

Correct answer: B

Explanation: Correct: SnapStart takes a snapshot of the initialized execution environment and restores functions from it, cutting Java startup latency dramatically without code rewrites. Why others are wrong: more memory adds CPU during execution but initialization still runs on every cold start; container images do not remove JVM/framework initialization and can start slower; VPC attachment has no effect on initialization time.

Q6. A Lambda function triggered by SQS must process each message, but occasionally a single malformed message causes the entire batch of ten to be retried and reprocessed. The developer wants only the failed messages returned to the queue. How should the function be configured?

A) Reduce the SQS batch size to 1 in the event source mapping

B) Enable partial batch responses on the SQS event source mapping and return the message IDs of only the failed items

C) Set the queue's visibility timeout to zero so failures are visible immediately

D) Configure a dead-letter queue on the function and let Lambda move the whole batch there

Correct answer: B

Explanation: Correct: With the partial batch response feature (ReportBatchItemFailures), the function returns only the IDs of failed messages, and Lambda deletes the successful ones while the failures become visible again. Why others are wrong: a batch size of 1 works but destroys batching efficiency; a zero visibility timeout causes duplicates and runaway retries for all messages, not just failures; a function-level DLQ receives the entire failed batch/event after retries, not just the bad message.

Q7. A developer is choosing how to expose a new Lambda function as an HTTPS endpoint for an internal tool. The tool needs simple request/response behavior, its own custom domain, and the lowest cost, and does not need request validation, usage plans, or API keys. Which option is the best fit?

A) Create a REST API in API Gateway with a Lambda proxy integration

B) Create a Lambda function URL and map the custom domain to it

C) Create a WebSocket API in API Gateway for persistent connections

D) Put an Application Load Balancer in front of the function with a Lambda target group

Correct answer: B

Explanation: Correct: A Lambda function URL is a dedicated HTTPS endpoint for a single function at no extra charge, and a custom domain can be mapped through Route 53/CloudFront or an ALB-free setup, matching the simple requirements. Why others are wrong: a REST API adds cost and features the tool does not need; WebSocket APIs are for two-way persistent connections, not simple request/response; an ALB adds hourly cost and another layer of management for a single function.

Q8. A data-processing Lambda function downloads a 700 MB reference dataset at startup and needs scratch space for intermediate files larger than the default ephemeral storage allows. What should the developer configure?

A) Increase the function's /tmp ephemeral storage allocation, up to 10 GB

B) Write intermediate files to the function's read-only deployment directory

C) Mount the function's own zip package as a writable file system

D) Store all intermediate files as Lambda environment variables

Correct answer: A

Explanation: Correct: Lambda lets you configure ephemeral storage between 512 MB and 10,240 MB, which is mounted at /tmp and is the supported scratch space. Why others are wrong: the deployment package directory is read-only at runtime; the zip package cannot be mounted as writable storage; environment variables are limited to 4 KB in total and are for configuration, not file storage.

Q9. A developer notices a Python Lambda function is CPU-bound when compressing files: execution time barely drops when the timeout is raised, but billing duration is high. Which change will MOST improve its performance?

A) Increase the function's configured memory, which proportionally increases allocated CPU

B) Decrease the memory so the function is throttled into running longer batches

C) Split the function into ten chained functions that each compress part of the file

D) Enable Lambda SnapStart to warm the compression libraries

Correct answer: A

Explanation: Correct: Lambda allocates CPU in proportion to configured memory, so raising memory speeds CPU-bound work and can even lower total cost by shortening duration. Why others are wrong: decreasing memory reduces CPU and lengthens execution; chaining functions adds orchestration complexity and the

per-function CPU limit remains; SnapStart targets Java initialization latency, not steady-state CPU throughput of a Python function.

Q10. A nightly reconciliation job written as a single Lambda function now needs to run for about 40 minutes and includes pauses where it waits for an external approval. Which approach should the developer take?

A) Raise the Lambda timeout above 15 minutes by requesting a quota increase

B) Split the job into steps orchestrated by AWS Step Functions, with Lambda functions performing each task

C) Run the Lambda function recursively so each invocation restarts the timer

D) Keep the job in Lambda but store progress in environment variables between invocations

Correct answer: B

Explanation: Correct: Lambda invocations are capped at 15 minutes, so long-running work with waits belongs in Step Functions, which orchestrates short Lambda tasks and can wait for approvals with callback task tokens. Why others are wrong: the 15-minute timeout is a hard limit that cannot be raised by quota request; recursive invocations are an anti-pattern Lambda actively detects and can loop or lose state; environment variables do not persist reliably between execution environments.

Q11. A developer must deploy a Lambda function whose dependency stack is 6 GB, far above the zip deployment limit. Which deployment method supports this with the LEAST change to the team's workflow?

A) Split the zip into several functions that share variables through Parameter Store

B) Deploy the function as a container image stored in Amazon ECR

C) Compress the dependencies with a Lambda layer stack of five layers

D) Base64-encode the package and pass it as an environment variable

Correct answer: B

Explanation: Correct: Lambda container images support images up to 10 GB stored in ECR, which fits the 6 GB dependency stack using a familiar Docker workflow. Why others are wrong: splitting the function changes the architecture and still leaves oversized dependencies; layers are limited to 250 MB of unzipped content in total across up to five layers; environment variables are capped at 4 KB total and cannot carry a package.

Q12. EXHIBIT - A payments team reviews the configuration of a production Lambda function and its recent behavior: Function configuration: Memory 128 MB; Timeout 3 seconds; Runtime: Python 3.12; Handler performs JSON parsing of large payloads, three sequential calls to an internal pricing API, and writes results to DynamoDB. Observed behavior: p50 duration 2,850 ms; frequent DurationExceeded timeouts at peak; CPU utilization reported as saturated; error rate rises with payload size. The team can change ONE setting first and wants the change most likely to remove the timeouts. Which change should they make?

A) Increase the memory allocation to 1,024 MB

B) Increase the timeout to 15 minutes

C) Move the function into a VPC

D) Enable provisioned concurrency with 20 units

Correct answer: A

Explanation: Correct: At 128 MB the function gets a small fraction of a vCPU, and Lambda scales CPU with memory; the saturated CPU and durations hugging the 3-second ceiling show a CPU-starved function, so raising memory directly cuts execution time. Why others are wrong: raising the timeout to 15 minutes only lets

slow executions run longer at higher cost without fixing the cause; VPC attachment does not change CPU or duration; provisioned concurrency removes cold starts, but these are warm executions timing out under load.

Q13. A Lambda function is invoked asynchronously to generate reports. When the downstream report service is down for an hour, all events from that hour are lost because retries are exhausted. The developer must ensure failed asynchronous invocations are preserved for later reprocessing. What should be configured?

A) A dead-letter queue (SQS queue or SNS topic) on the function for failed asynchronous events

B) A higher reserved concurrency so retries happen faster

C) Synchronous invocation from the caller so errors surface immediately

D) Lambda destinations on success only, writing to an S3 bucket

Correct answer: A

Explanation: Correct: For asynchronous invocations, a dead-letter queue captures events that fail after Lambda's built-in retries so they can be inspected and reprocessed. Why others are wrong: higher reserved concurrency does not extend the retry window or preserve failed events; changing callers to synchronous invocation alters the architecture and shifts retry responsibility to every caller; on-success destinations do nothing for the failed events that are being lost.

Q14. A developer's Lambda function creates a new database connection inside the handler on every invocation, causing connection storms during bursts. Which coding pattern fixes this with the least effort?

A) Move connection creation outside the handler so it is created once per execution environment and reused across invocations

B) Store the connection object in an S3 bucket and fetch it for each request

C) Force a cold start before each invocation by publishing a new function version

D) Increase the database's maximum connection limit to absorb the storm

Correct answer: A

Explanation: Correct: Code outside the handler runs once when the execution environment initializes, and warm environments reuse it, so clients and connections defined there are shared across invocations. Why others are wrong: connections are live network objects and cannot be serialized into S3; deliberately forcing cold starts makes the problem worse; raising database limits treats the symptom and can be expensive or impossible at scale.

Q15. A healthcare application stores configuration in Lambda environment variables, but the combined size of the values now exceeds the 4 KB limit. The values include a database connection string that must stay encrypted. What should the developer do?

A) Compress the environment variables with gzip before setting them

B) Move the configuration to AWS Systems Manager Parameter Store or Secrets Manager and retrieve it at runtime with caching in the function

C) Split the configuration across two Lambda functions that call each other

D) Store the values in the function's tags instead of environment variables

Correct answer: B

Explanation: Correct: Parameter Store and Secrets Manager are built for externalized configuration and secrets, support encryption with KMS, and the function can cache retrieved values outside the handler to limit API calls. Why others are wrong: Lambda does not decompress environment variables for you, and the 4 KB

limit applies to the stored values; splitting functions complicates the design without removing the limit; tags are metadata for resource management, limited in size, and not readable as runtime configuration in the same way.

Q16. A developer wants a Lambda function to process very large product-catalog files (up to 2 GB) that several other services also need to read and update concurrently. Which storage option fits Lambda's model best?

A) Mount an Amazon EFS file system to the function and process the files in place

B) Use the function's /tmp storage and accept that other services cannot see the files

C) Store the files in DynamoDB items split into 400 KB chunks

D) Keep the files in the deployment package and rebuild it for each update

Correct answer: A

Explanation: Correct: Lambda supports mounting EFS, giving functions a shared, persistent file system that scales to gigabytes and is accessible concurrently by other services. Why others are wrong: /tmp is private to an execution environment and not shared; DynamoDB items are limited to 400 KB and chunking large files is fragile and slow; rebuilding the deployment package for data updates conflates code and data and cannot support concurrent writers.

Q17. A public API backed by Lambda returns recommendation payloads of up to 20 MB. Clients complain that responses are buffered and arrive all at once, and some large responses fail. Which Lambda feature addresses this?

A) Lambda response streaming, which progressively streams the response to the client

B) Raising the function's memory so the payload is generated faster

C) Returning the payload as a base64 string inside a JSON envelope

D) Increasing the API payload quota for the account through Service Quotas

Correct answer: A

Explanation: Correct: Lambda response streaming lets the function write the response progressively, improving time-to-first-byte and supporting payloads up to 200 MB, which fits the 20 MB responses. Why others are wrong: more memory speeds generation but the response is still buffered and subject to the standard 6 MB payload limit; base64 encoding inflates the payload by about a third and keeps the limit problem; the synchronous payload limit is not a customer-adjustable quota.

Q18. A developer invokes a Lambda function synchronously from a mobile backend and passes a 9 MB document for analysis. The invoke calls fail with a payload size error. What is the recommended pattern?

A) Split the document into 6 MB chunks and invoke the function once per chunk

B) Upload the document to S3 and pass the bucket and key in a small event payload for the function to fetch

C) Compress the document with base64 encoding before sending it

D) Switch the function to a container image so payload limits no longer apply

Correct answer: B

Explanation: Correct: The claim-check pattern - storing the large payload in S3 and passing a reference - is the standard way around Lambda's 6 MB synchronous request payload limit. Why others are wrong: chunking creates ordering and reassembly complexity and partial-failure headaches; base64 increases size by roughly 33 percent, making the error worse; the payload limit applies to the invocation API regardless of whether the function is a zip or container deployment.

Q19. A fintech team must prove that only Lambda code they built and approved runs in production. They need Lambda to verify a signature before deploying function code. Which feature provides this?

A) Lambda code signing with AWS Signer, configured to reject unsigned deployments

B) VPC endpoints for the Lambda API so deployments stay on the private network

C) Lambda layers with version pinning so old code cannot change

D) S3 Versioning on the bucket that stores deployment packages

Correct answer: A

Explanation: Correct: Lambda code signing uses AWS Signer signing profiles and a code signing configuration that checks signatures at deployment time and can warn or enforce rejection of unsigned code. Why others are wrong: VPC endpoints control network paths, not code authenticity; layer pinning controls dependency versions, not who built the function code; S3 Versioning preserves old packages but does not verify signatures.

Q20. A developer troubleshooting a Lambda function notices the same SQS message being processed repeatedly and suspects the function is calling itself indirectly through an S3 notification chain, creating a runaway loop. Which Lambda capability helps stop this class of problem, and what should the developer also do? (Select the best single answer.)

A) Use Lambda's recursive loop detection, which stops invocations in supported service chains, and redesign the chain to break the cycle

B) Disable the function's execution role so the loop cannot continue

C) Set the function's reserved concurrency to zero permanently and delete its event source mappings so no trigger can ever restart the loop

D) Turn off CloudWatch Logs so the loop is no longer visible

Correct answer: A

Explanation: Correct: Lambda includes recursive loop detection for common patterns involving services such as SQS, Lambda, and S3, and can stop runaway invocations, but the durable fix is redesigning the event chain so outputs cannot re-trigger the same function unintentionally. Why others are wrong: removing the execution role breaks the function entirely and is not a fix; zero reserved concurrency disables the function permanently rather than correcting the design; disabling logs only hides the symptom while the loop keeps running and billing.

Q21. A developer wants to minimize cost for a CPU-bound Lambda function by choosing the right memory setting. Which reasoning is CORRECT?

A) 10,240 MB, because maximum CPU always finishes the work fastest, and the shortest possible duration always produces the lowest overall bill

B) The smallest memory of 128 MB, because it carries the lowest per-GB-second rate and the lowest possible charge for each invocation

C) Provisioned concurrency, because pre-warmed environments make invocations free of duration charges and remove any need to tune memory

D) An intermediate setting such as 1,769 MB (one full vCPU), because cost is memory multiplied by duration and added CPU shortens the duration

Correct answer: D

Explanation: Correct: Because Lambda charges per GB-second and faster execution shortens duration, both configuration points are testable - the 1,769 MB point (one full vCPU) is a common sweet spot for CPU-bound work. Why others are wrong: 10,240 MB maximizes CPU but also multiplies the per-millisecond

price, often raising cost; the smallest memory minimizes the rate but stretches duration enough to cost more overall; provisioned concurrency removes cold starts but does not change per-invocation compute pricing.

Q22. A Java application uses Lambda SnapStart. After enabling it, sessions intermittently fail because every restored environment presents the same cached access token. What is the correct fix for SnapStart-based functions?

A) Hard-coding database credentials so no network call is needed at restore

B) Embedding the current timestamp in a static variable initialized before the snapshot

C) Declaring all variables static so they can never change after restore

D) Using the after-restore runtime hook to refresh values that must be unique per environment, such as credentials fetched at runtime

Correct answer: D

Explanation: Correct: SnapStart restores a snapshot of an already-initialized environment, so initialization hooks that must run per unique environment (like fetching per-host secrets) belong in after-restore hooks. Why others are wrong: static values such as file handles are restored from the snapshot; timestamps embedded 'before snapshot' are stale after restore; hardcoded credentials are insecure regardless of SnapStart.

Q23. A team is migrating its client applications to IPv6-only networking. The clients invoke an AWS Lambda function over the internet, and the team needs those invocations to work over IPv6 without changing the function code. Which Lambda capability supports this scenario?

A) Deploy the function behind CloudFront Functions instead

B) Use S3 as an intermediary to carry requests to the function

C) Invoke the function through its dual-stack endpoint name so IPv6 clients can reach it

D) Move the workload to a Region that has no IPv4-only services

Correct answer: C

Explanation: Correct: Lambda supports dual-stack (IPv4/IPv6) endpoints for invoking functions and for the function control plane, letting VPC workloads use IPv6 addressing where configured. Why others are wrong: CloudFront Functions are edge JavaScript handlers with a different runtime and limits, not an IPv6 access mechanism for Lambda; S3 is not a network path to invoke functions; switching regions does not enable IPv6 for the API.

Q24. A Lambda function consumes an SQS queue that carries many event types, but the function only cares about 'order.shipped' events; it still gets invoked for every batch. How can the developer reduce wasted invocations without adding code?

A) Filtering records inside the handler as the first line of code

B) A dead-letter queue on the source queue

C) Reducing the event source batch size to 1

D) Event filtering patterns on the event source mapping

Correct answer: D

Explanation: Correct: Event filtering on SQS, DynamoDB Streams, and Kinesis mappings filters records before invocation, so Lambda is only invoked (and billed) when matching records arrive. Why others are wrong: filtering inside the handler prevents downstream work but the function is still invoked and billed for every batch; DLQs act on failed messages, not irrelevant ones; reducing batch size increases invocation count and cost.

Q25. A developer sets a Lambda function's timeout to 25 minutes for a video analysis task and finds the setting rejected. Which statement is TRUE?

A) Request a Service Quotas increase for the function timeout

B) Switch the function from a zip package to a container image

C) The timeout cannot exceed 15 minutes; redesign the job as shorter steps, for example with Step Functions

D) Enable provisioned concurrency so the function is allowed to run longer

Correct answer: C

Explanation: Correct: Lambda's maximum timeout is 15 minutes for every function; jobs that run longer or must pause belong in Step Functions or on compute services like Fargate. Why others are wrong: quota increases cannot raise the timeout past 15 minutes; container images share the same 15-minute limit as zip deployments; provisioned concurrency affects cold starts and reserved capacity, not the timeout ceiling.

Q26. A developer adds a scheduled ping every 5 minutes to keep a Lambda function warm, but users still see cold starts when a burst of 300 requests arrives. Why is the warm-up ineffective? (Select the best single answer.)

A) The function's memory is too small for the warm-up to help

B) Provisioned concurrency is already configured for the function, so the remaining issue is that callers use the wrong invocation type for bursts of traffic

C) A burst of 300 concurrent requests needs roughly 300 execution environments, and one scheduled warm-up only ever warms a single environment

D) The function is attached to a VPC, which blocks scheduled warm-up events from reaching it

Correct answer: C

Explanation: Correct: The warm-up is doing nothing because Lambda scales the number of execution environments with concurrency; a concurrent burst of 300 requests creates hundreds of cold environments regardless of one warmed instance. Why others are wrong: memory size affects cold-start speed, not whether cold starts occur across many environments; provisioned concurrency must be provisioned at the expected concurrency level (300) to be warm for the burst; invocation type does not change the per-environment initialization model. A VPC attachment also does not explain it: scheduled invocations reach the function through the Lambda service API regardless of the function's network configuration.

Q27. A Lambda function renders customer statements. Profiling shows that most of the per-invocation time is spent loading and compiling a template file, even though the template changes only when code is redeployed. What is the most effective optimization?

A) Increase memory to speed repeated compilation

B) Schedule the function to restart hourly

C) Fetch and compile the template from S3 inside the handler on each request

D) Load and compile the template outside the handler at initialization time

Correct answer: D

Explanation: Correct: Compiling the template once and re-evaluating it per request amortizes the expensive initialization across all invocations that reuse the environment, which is exactly what warm reuse is for. Why others are wrong: global S3 fetches of the template repeat the expensive I/O every cold start plus per-request downloads; larger memory does not remove redundant initialization work; scheduled restarts would force more cold starts, the opposite of the goal.

Q28. A payments Lambda function can be invoked more than once for the same order because its

SQS trigger delivers at least once. The team wants a managed, low-code way to make handler execution idempotent. Which approach fits best?

A) Configure the SQS redrive policy with maxReceiveCount of 1

B) Turn on exactly-once delivery for the trigger in the Lambda console

C) Use the Lambda Powertools idempotency utility backed by DynamoDB to record completed request keys and short-circuit duplicates

D) Check the source IP address of each invocation and ignore repeats

Correct answer: C

Explanation: Correct: Lambda Powertools is an AWS-maintained toolkit for Lambda that includes an idempotency utility designed exactly to make retried Lambda invocations safe. Why others are wrong: the SQS redrive policy governs queue retries, not handler-level idempotency; 'at-least-once delivery' is a delivery guarantee of the trigger, and Lambda cannot switch triggers to exactly-once; checking the caller IP does not identify retries reliably.

Q29. A suite of Lambda functions fronts an Aurora MySQL database. At peak, hundreds of execution environments each open their own connections and exhaust the database's connection limit. Which service should the developer place in front of the database?

A) Store open connections in /tmp so later invocations reuse them

B) Cache the database password in the function to avoid Secrets Manager calls

C) Put Amazon RDS Proxy between the functions and the database so connection pooling is shared across execution environments

D) Raise the function's reserved concurrency so fewer environments exist

Correct answer: C

Explanation: Correct: RDS Proxy pools and multiplexes application connections, so thousands of Lambda environments share a small set of database connections, and IAM authentication keeps credentials out of the code path. Why others are wrong: caching connections in /tmp does not share them across environments; caching the password in memory still requires each environment to open its own connection; the connection limit is a database constraint the storm would still hit.

Q30. A developer wants better price-performance for a fleet of Python Lambda functions that have no compiled x86-specific dependencies. Which change does Lambda support?

A) SnapStart must be disabled before switching architectures

B) The function must be moved out of the VPC to run on Graviton

C) API callers must be updated to send an architecture header with every request so Lambda can route the calls to Graviton processors

D) Deploy the same code on arm64 by changing the function's architecture setting and rebuilding any native dependencies for arm64

Correct answer: D

Explanation: Correct: arm64 (Graviton2) Lambda functions offer better price-performance for many workloads, and Lambda supports arm64 for both zip and container-image deployments including common Python runtimes. Why others are wrong: clients invoke the function the same way regardless of architecture; SnapStart is a Java-focused startup feature unrelated to CPU architecture; VPC support is identical for both architectures.