**Timed Mode Set 1 – AWS Certified Developer Associate**

56 of 65 questions answered correctly

**Your time:00:00:05**

You have reached 56 of 65 point(s), (86.15%)

**Categories**

1. CDA – Deployment85.71%
2. CDA – Development with AWS Services88.89%
3. CDA – Security66.67%
4. CDA – Troubleshooting and Optimization90.91%

* Congratulations! You passed the test. We still highly encourage you to carefully read our detailed explanations including the references and [cheat sheets](https://tutorialsdojo.com/links-to-all-aws-cheat-sheets/). 🙂

**1. Question**

An application hosted in an Auto Scaling group of On-Demand EC2 instances is used to process data polled from an SQS queue and the generated output is stored in an S3 bucket. To improve security, you were tasked to ensure that all objects in the S3 bucket are encrypted at rest using server-side encryption with AWS KMS Keys (SSE-KMS).

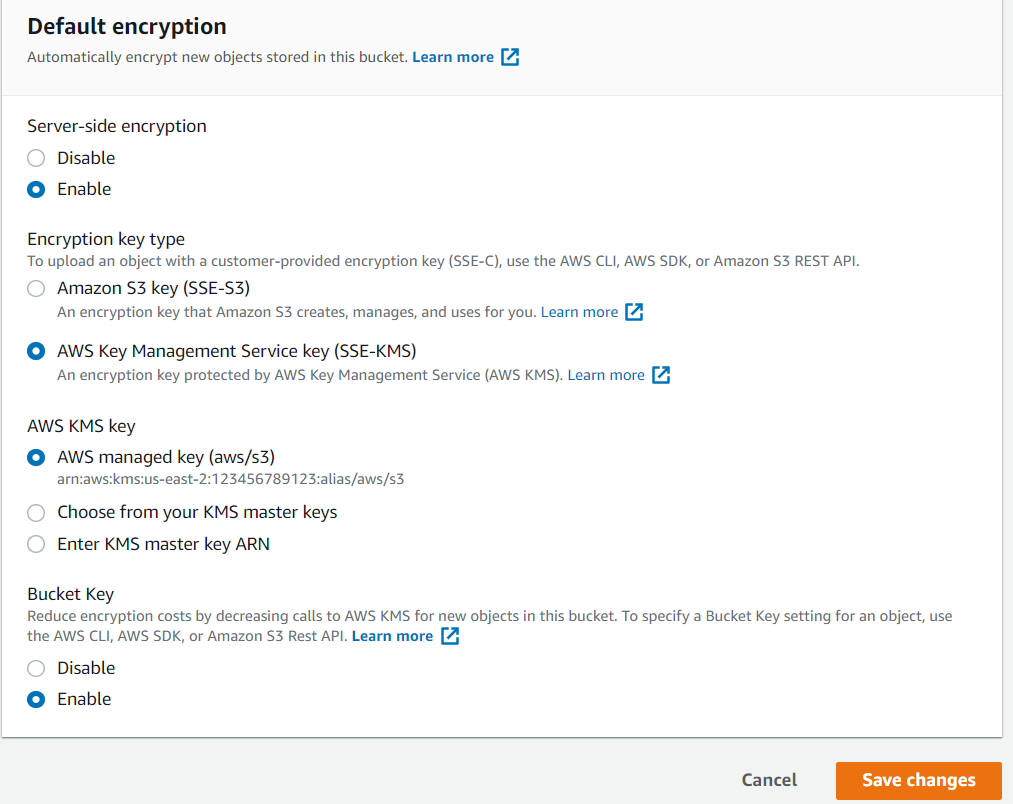
Which of the following is required to properly implement this requirement?

* + **Add a bucket policy which denies any s3:PutObject action unless the request includes the x-amz-server-side-encryption header.**
  + Add a bucket policy which denies any s3:PutObject action unless the request includes the x-amz-server-side-encryption-aws-kms-key-id header.
  + Add a bucket policy which denies any s3:PostObject action unless the request includes the x-amz-server-side-encryption header.
  + Add a bucket policy which denies any s3:PostObject action unless the request includes the x-amz-server-side-encryption-aws-kms-key-id header.

**Correct**

**Server-side encryption** is about protecting data at rest. AWS Key Management Service (**AWS KMS**) is a service that combines secure, highly available hardware and software to provide a key management system scaled for the cloud. AWS KMS uses KMS keys to encrypt your Amazon S3 objects. You use AWS KMS via the AWS Management Console or AWS KMS APIs to centrally create encryption keys, define the policies that control how keys can be used, and audit key usage to prove they are being used correctly. You can use these keys to protect your data in Amazon S3 buckets.

The first time you add an SSE-KMS–encrypted object to a bucket in a region, a default KMS key is created for you automatically. This key is used for SSE-KMS encryption unless you select a KMS key that you created separately using AWS Key Management Service. Creating your own KMS key gives you more flexibility, including the ability to create, rotate, disable, and define access controls, and to audit the encryption keys used to protect your data.



Amazon S3 supports bucket policies that you can use if you require server-side encryption for all objects that are stored in your bucket. For example, you can set a bucket policy that denies permission to upload an object (s3:PutObject) to everyone if the request does not include the x-amz-server-side-encryption header requesting server-side encryption with SSE-KMS.

When you upload an object, you can specify the KMS key using the x-amz-server-side-encryption-aws-kms-key-id header which you can use to require a specific KMS key for object encryption. If the header is not present in the request, Amazon S3 assumes the default KMS key. Regardless, the KMS key ID that Amazon S3 uses for object encryption must match the KMS key ID in the policy, otherwise Amazon S3 denies the request.

Therefore, the correct answer is: **Add a bucket policy which denies any s3:PutObject action unless the request includes the x-amz-server-side-encryption header.**

The option that says: **Adding a bucket policy which denies any s3:PutObject action unless the request includes the x-amz-server-side-encryption-aws-kms-key-id header** is incorrect because you have to use the x-amz-server-side-encryption header instead.

The option that says: **Adding a bucket policy which denies any s3:PostObject action unless the request includes the x-amz-server-side-encryption header** is incorrect because you have to deny s3:PutObject and not the s3:PostObject action.

The option that says: **Adding a bucket policy which denies any s3:PostObject action unless the request includes the x-amz-server-side-encryption-aws-kms-key-id header** is incorrect because you have to use the x-amz-server-side-encryption header instead. Moreover, you have to deny s3:PutObject and not the s3:PostObject action.

**References:**

<https://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html>

<https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html>

**Check out this Amazon S3 Cheat Sheet:**

<https://tutorialsdojo.com/amazon-s3/>

1. **2. Question**

An application hosted in an Amazon ECS Cluster processes a large data stream and stores the result in a DynamoDB table. There is an urgent requirement to detect new entries in the table and automatically trigger a Lambda function to run some verification tests on the processed data.

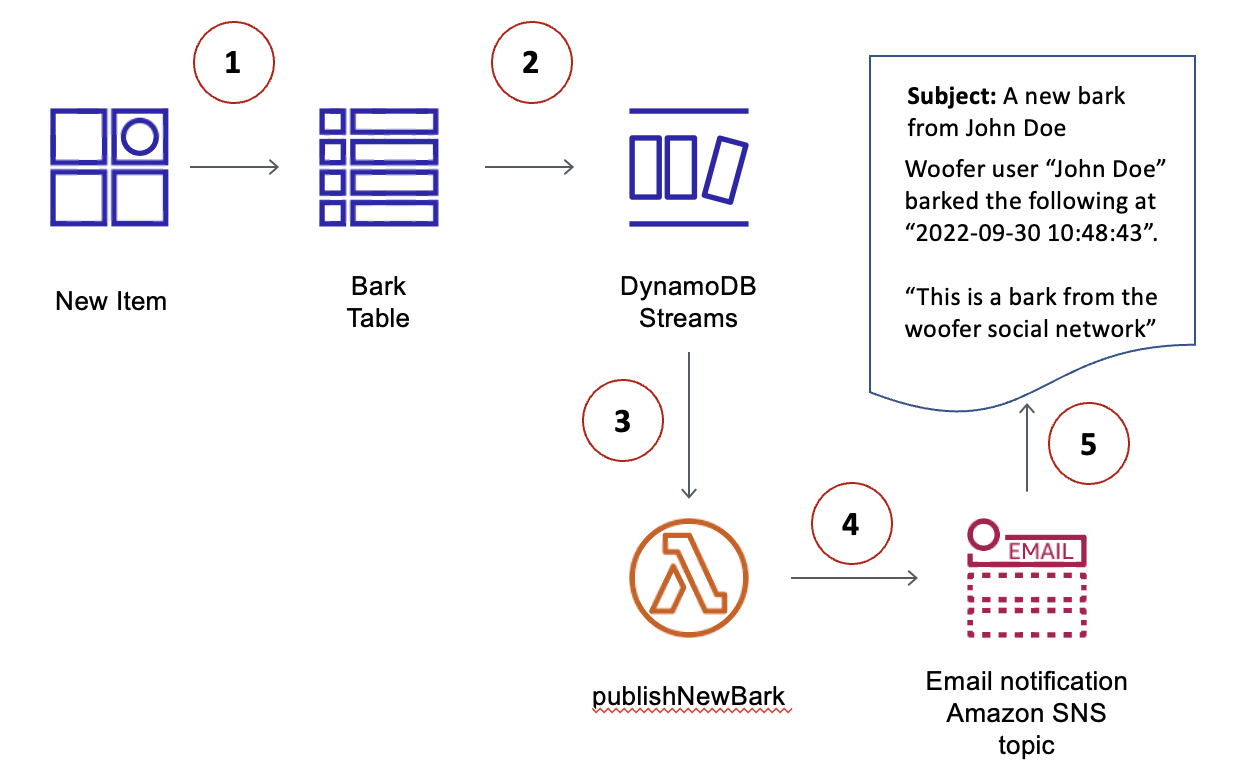
Which of the following options can satisfy the requirement with minimal configuration?

* + **Enable DynamoDB Streams to detect the new entries and automatically trigger the Lambda function.**
  + Run the Lambda function using SNS each time the ECS Cluster successfully processes financial data.
  + Set up an Amazon EventBridge (Amazon CloudWatch Events) rule to automatically trigger the Lambda function whenever a new entry is created in the DynamoDB table.
  + Detect the new entries in the DynamoDB table using AWS Copilot, then automatically invoke the Lambda function for processing.

**Correct**

Amazon DynamoDB is integrated with AWS Lambda so that you can create *triggers*—pieces of code that automatically respond to events in DynamoDB Streams. With triggers, you can build applications that react to data modifications in DynamoDB tables.

If you enable DynamoDB Streams on a table, you can associate the stream ARN with a Lambda function that you write. Immediately after an item in the table is modified, a new record appears in the table’s stream. AWS Lambda polls the stream and invokes your Lambda function synchronously when it detects new stream records.



You can create a Lambda function which can perform a specific action that you specify, such as sending a notification or initiating a workflow. For instance, you can set up a Lambda function to simply copy each stream record to persistent storage, such as EFS or S3, to create a permanent audit trail of write activity in your table.

Suppose you have a mobile gaming app that writes to a TutorialsDojoCourses table. Whenever the TopCourse attribute of the TutorialsDojoScores table is updated, a corresponding stream record is written to the table’s stream. This event could then trigger a Lambda function that posts a congratulatory message on a social media network. (The function would simply ignore any stream records that are not updates to TutorialsDojoCourses or that do not modify the TopCourse attribute.)

Hence, the correct answer is to **enable DynamoDB Streams to detect the new entries and automatically trigger the Lambda function**. In this way, the requirement can be met with minimal configuration change as DynamoDB streams can be used as an event source to automatically trigger Lambda functions whenever there is a new entry.

The option that says: **Set up an Amazon EventBridge (Amazon CloudWatch Events) rule to automatically trigger the Lambda function whenever a new entry is created in the DynamoDB table**is incorrect. An Amazon EventBridge (Amazon CloudWatch Events) rule is not capable of detecting table-level events from Amazon DynamoDB.

The option that says: **Run the Lambda function using SNS each time the ECS Cluster successfully processes financial data** is incorrect because you don’t need to create an SNS topic just to invoke Lambda functions. You can simply enable DynamoDB streams to meet the requirement with less configuration.

The option that says: **Detect new entries in the DynamoDB table using AWS Copilot then automatically invoke the Lambda function for processing** is incorrect because AWS Copilot is just a command-line tool that allows developers to quickly and easily build, release, and operate containerized applications on AWS. It does not have the capability to detect new entries in a DynamoDB table.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.Lambda.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>

**Check out this Amazon DynamoDB cheat sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

**AWS Lambda Integration with Amazon DynamoDB Streams:**

<https://tutorialsdojo.com/aws-lambda-integration-with-amazon-dynamodb-streams/>

1. **3. Question**

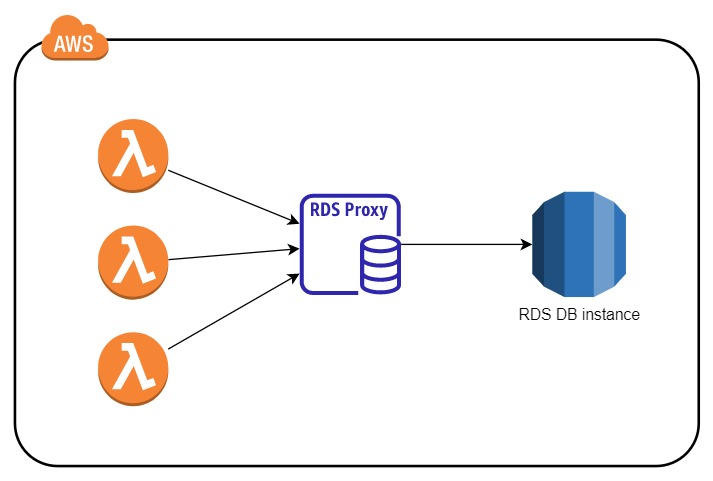
A developer has an application that uses a Lambda function to process data from an Aurora MySQL DB Instance in a Virtual Private Cloud (VPC). The database throws a MySQL: ERROR 1040: Too many connections error whenever there is a surge in incoming traffic.

Which is the most suitable solution for resolving the issue?

* + Increase the concurrency limit of the Lambda function
  + Increase the value of the max\_connections parameter of the Aurora MySQL DB Instance.
  + **Provision an RDS Proxy between the Lambda function and the RDS database instance**
  + Increase the allocated memory of your function.

**Correct**

If a “Too Many Connections” error happens to a client connecting to a MySQL database, it means all available connections are in use by other clients. Opening a connection consumes resources on the database server. Since Lambda functions can scale to tens of thousands of concurrent connections, your database needs more resources to open and maintain connections instead of executing queries. The maximum number of connections a database can support is largely determined by the amount of memory allocated to it. Upgrading to a database instance with higher memory is a straightforward way of solving the problem. Another approach would be to maintain a connection pool that clients can reuse. This is where RDS Proxy comes in.



RDS Proxy helps you manage a large number of connections from Lambda to an RDS database by establishing a warm connection pool to the database. Your Lambda functions interact with RDS Proxy instead of your database instance. It handles the connection pooling necessary for scaling many simultaneous connections created by concurrent Lambda functions. This allows your Lambda applications to reuse existing connections, rather than creating new connections for every function invocation.

Thus, the correct answer is: **Provision an RDS Proxy between the Lambda function and RDS database instance.**

The option that says: **Increase the concurrency limit of the Lambda function** is incorrect. The concurrency limit refers to the maximum requests AWS Lambda can handle simultaneously. Increasing the limit will allow for more requests to open a database connection, which could potentially worsen the problem.

The option that says: **Increase the value of the max\_connections parameter of the Aurora MySQL DB Instance** is incorrect. Although this may be a valid solution, it is not the most efficient since it simply increases the maximum number of connections that can be made to the database instance. Moreover, increasing the maximum number of connections alone, without considering the database size, may lead to other issues, such as slow response times, timeouts, and even crashes.

The option that says: **Increase the allocated memory of your function**is incorrect. Increasing the Lambda function’s memory can improve its performance, but it may not necessarily solve the underlying issue of the ‘too many connections’ error. This error is typically caused by a limit on the maximum number of connections the database can handle, so solutions that address the database’s connection management, such as creating a connection pool using RDS Proxy, are more likely to be effective.

**References:**

<https://aws.amazon.com/rds/proxy/>

<https://aws.amazon.com/blogs/compute/using-amazon-rds-proxy-with-aws-lambda/>

**Check out this Amazon RDS Cheat Sheet:**

<https://tutorialsdojo.com/amazon-relational-database-service-amazon-rds/>

1. **4. Question**

A developer is managing a distributed system that consists of an Application Load Balancer, an SQS queue, and an Auto Scaling group of EC2 instances. The system has been integrated with CloudFront to better serve clients worldwide. To enhance the security of the in-flight data, the developer was instructed to establish an end-to-end SSL connection between the origin and the end-users.

Which TWO options will allow the developer to meet this requirement using CloudFront? (Select TWO.)

* + Configure your ALB to only allow traffic on port 443 using an SSL certificate from AWS Config.
  + **Configure the Origin Protocol Policy to use HTTPS only**
  + Set up an Origin Access Control (OAC) setting
  + **Configure the Viewer Protocol Policy to use HTTPS only**
  + Associate a Web ACL using AWS Web Application Firewall (WAF) with your CloudFront Distribution.

**Correct**

For web distributions, you can configure CloudFront to require that viewers use HTTPS to request your objects, so connections are encrypted when CloudFront communicates with viewers. You can also configure CloudFront to use HTTPS to get objects from your origin, so connections are encrypted when CloudFront communicates with your origin.

If you configure CloudFront to require HTTPS both to communicate with viewers and to communicate with your origin, here’s what happens when CloudFront receives a request for an object. The process works basically the same way whether your origin is an Amazon S3 bucket or a custom origin such as an HTTP/S server:

1. A viewer submits an HTTPS request to CloudFront. There’s some SSL/TLS negotiation here between the viewer and CloudFront. In the end, the viewer submits the request in an encrypted format.

2. If the object is in the CloudFront edge cache, CloudFront encrypts the response and returns it to the viewer, and the viewer decrypts it.

3. If the object is not in the CloudFront cache, CloudFront performs SSL/TLS negotiation with your origin and, when the negotiation is complete, forwards the request to your origin in an encrypted format.

4. Your origin decrypts the request, encrypts the requested object, and returns the object to CloudFront.

5. CloudFront decrypts the response, re-encrypts it, and forwards the object to the viewer. CloudFront also saves the object in the edge cache so that the object is available the next time it’s requested.

6. The viewer decrypts the response.

You can configure one or more cache behaviors in your CloudFront distribution to require HTTPS for communication between viewers and CloudFront. You also can configure one or more cache behaviors to allow both HTTP and HTTPS, so that CloudFront requires HTTPS for some objects but not for others.

To implement this setup, you have to change the **Origin Protocol Policy** setting for the applicable origins in your distribution. If you’re using the domain name that CloudFront assigned to your distribution, such as dtut0rial5d0j0.cloudfront.net, you change the **Viewer Protocol Policy** setting for one or more cache behaviors to require HTTPS communication. With this configuration, CloudFront provides the SSL/TLS certificate.

Hence, the correct answers are: **Configure the Origin Protocol Policy to use HTTPS only** and **Configure the Viewer Protocol Policy to use HTTPS only**are correct answers in this scenario.

The option that says: **Configure your ALB to only allow traffic on port 443 using an SSL certificate from AWS Config**is incorrect because you can’t store a certificate in AWS Config.

The option that says: **Set up an Origin Access Control (OAC) setting**is incorrect because this CloudFront feature only allows you to secure S3 origins by granting access to S3 buckets for designated CloudFront distributions. This method is applicable only to S3 origins and cannot be used to establish end-to-end SSL connections for other origins.

The option that says: **Associate a Web ACL using AWS Web Application Firewall (WAF) with your CloudFront Distribution**is incorrect because AWS WAF is primarily used to protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources. This will not allow you to establish an SSL connection between your origin and your clients.

**References:**

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https.html>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-cloudfront-to-custom-origin.html#using-https-cloudfront-to-origin-certificate>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-viewers-to-cloudfront.html>

**Check out this Amazon CloudFront Cheat Sheet:**

<https://tutorialsdojo.com/amazon-cloudfront/>

1. **5. Question**

A Software Engineer is developing an application that will be hosted on an EC2 instance and read messages from a standard SQS queue. The average time that it takes for the producers to send a new message to the queue is 10 seconds.

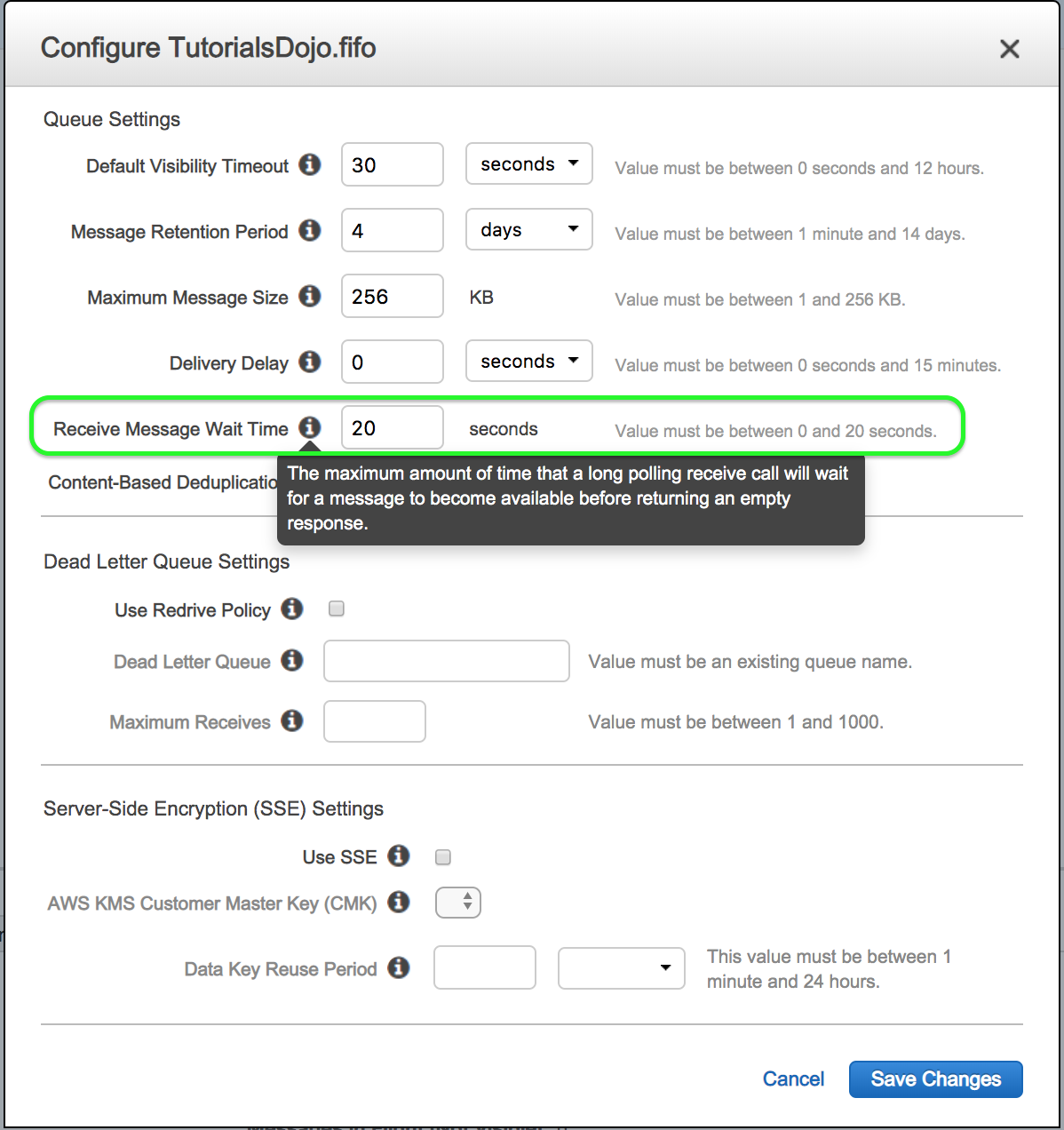
Which of the following is the MOST efficient way for the application to query the new messages from the queue?

* + **Configure the SQS queue to use Long Polling.**
  + Configure each message in the SQS queue to have a custom visibility timeout of 10 seconds.
  + Configure the SQS queue to use Short Polling.
  + Configure an SQS Delay Queue with a value of 10 seconds.

**Correct**

Amazon SQS long polling is a way to retrieve messages from your Amazon SQS queues. While the regular short polling returns immediately, even if the message queue being polled is empty, long polling doesn’t return a response until a message arrives in the message queue or the long poll times out.

Long polling helps reduce the cost of using Amazon SQS by eliminating the number of empty responses (*when there are no messages available for a ReceiveMessage request*) and false empty responses (*when messages are available but aren’t included in a response*). This type of polling is suitable if the new messages that are being added to the SQS queue arrive less frequently.



You can configure long polling to your SQS queue by simply setting the *“Receive Message Wait Time”* field to a value greater than **0**. Hence, the correct answer for this scenario is to **configure the SQS queue to use Long Polling.**

The option that says: **Configure each message in the SQS queue to have a custom visibility timeout of 10 seconds** is incorrect because a visibility timeout is primarily used to prevent other consumers from processing the message again for a period of time. This is normally used if your application takes a long time to process and delete a message from the SQS queue.

The option that says: **Configure the SQS queue to use Short Polling** is incorrect because it is inefficient to poll the queue every second if the average time that it takes for the producers to send a new message to the queue is 40 seconds. It is better to do Long Polling, which will query the queue every 15 or 20 seconds, considering that new messages are not being added every second.

The option that says:**Configure an SQS Delay Queue with a value of 10 seconds** is incorrect because this is primarily configured if you want to postpone the delivery of new messages to the SQS queue for a number of seconds. Having this SQS configuration which sets the new messages to remain invisible to the consumers for a duration of the delay period is not helpful in the given scenario. It is still better to use Long Polling instead of setting up a delay queue.

**References:**

<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-long-polling.html>

<https://aws.amazon.com/sqs/faqs/>

**Check out this Amazon SQS Cheat Sheet:**

<https://tutorialsdojo.com/amazon-sqs/>

**Amazon Simple Workflow (SWF) vs. AWS Step Functions vs. Amazon SQS:**

<https://tutorialsdojo.com/amazon-simple-workflow-swf-vs-aws-step-functions-vs-amazon-sqs/>

**Comparison of AWS Services Cheat Sheets:**

<https://tutorialsdojo.com/comparison-of-aws-services/>

1. **6. Question**

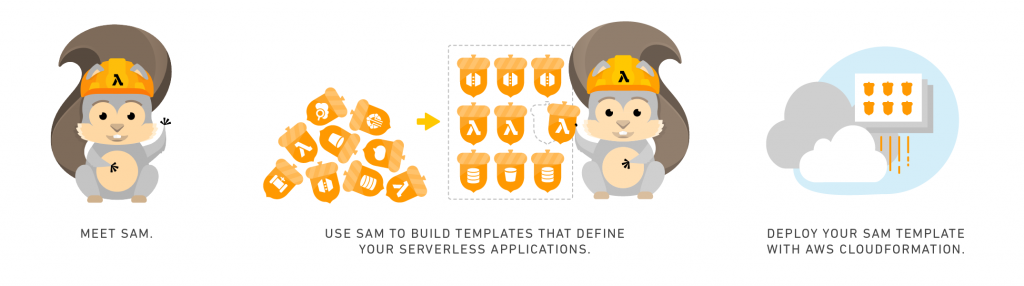
A developer is instructed to set up a new serverless architecture composed of AWS Lambda, API Gateway, and DynamoDB in a single stack. The new architecture should allow the developer to locally build, test, and debug serverless applications.

Which of the following should the developer use to satisfy the above requirement?

* + AWS CloudFormation
  + AWS Systems Manager
  + **AWS Serverless Application Model (AWS SAM)**
  + AWS Elastic Beanstalk

**Correct**

The **AWS Serverless Application Model (AWS SAM)** is an open-source framework that you can use to build serverless applications on AWS. It consists of the AWS SAM template specification that you use to define your serverless applications and the AWS SAM command line interface (AWS SAM CLI) that you use to build, test, and deploy your serverless applications.



Because AWS SAM is an extension of AWS CloudFormation, you get the reliable deployment capabilities of AWS CloudFormation. You can define resources by using AWS CloudFormation in your AWS SAM template. Also, you can use the full suite of resources, intrinsic functions, and other template features that are available in AWS CloudFormation.

You can use AWS SAM with a suite of AWS tools for building serverless applications. The AWS SAM CLI lets you locally build, test, and debug serverless applications that are defined by AWS SAM templates. The CLI provides a Lambda-like execution environment locally. It helps you catch issues upfront by providing parity with the actual Lambda execution environment. To step through and debug your code to understand what the code is doing, you can use AWS SAM with AWS toolkits like the AWS Toolkit for JetBrains, AWS Toolkit for PyCharm, AWS Toolkit for IntelliJ, and AWS Toolkit for Visual Studio Code. This tightens the feedback loop by making it possible for you to find and troubleshoot issues that you might run into in the cloud.

Therefore, the most suitable service to use in this scenario is**AWS Serverless Application Model (AWS SAM)**.

**AWS CloudFormation** is incorrect. Although this service can certainly be used to deploy Lambda, API Gateway, DynamoDB, and other AWS resources of your serverless application, it doesn’t have the capability to locally build, test, and debug your application like what AWS SAM has. In addition, AWS SAM is a more suitable service to use if you want to deploy and manage your serverless applications in AWS just as mentioned above.

**AWS Systems Manager** is incorrect because this service is primarily used for managing resources in your AWS environment, not for building, testing, and debugging serverless applications.

**AWS Elastic Beanstalk**is incorrect because this service is not suitable for deploying serverless applications. In addition, it doesn’t have the capability to locally build, test, and debug your serverless applications as effectively as what AWS SAM can do.

**References:**

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html>

**Check out this AWS Serverless Application Model Cheat Sheet:**

<https://tutorialsdojo.com/aws-serverless-application-model-sam/>

**Comparison of AWS Services Cheat Sheets:**

<https://tutorialsdojo.com/comparison-of-aws-services/>

1. **7. Question**

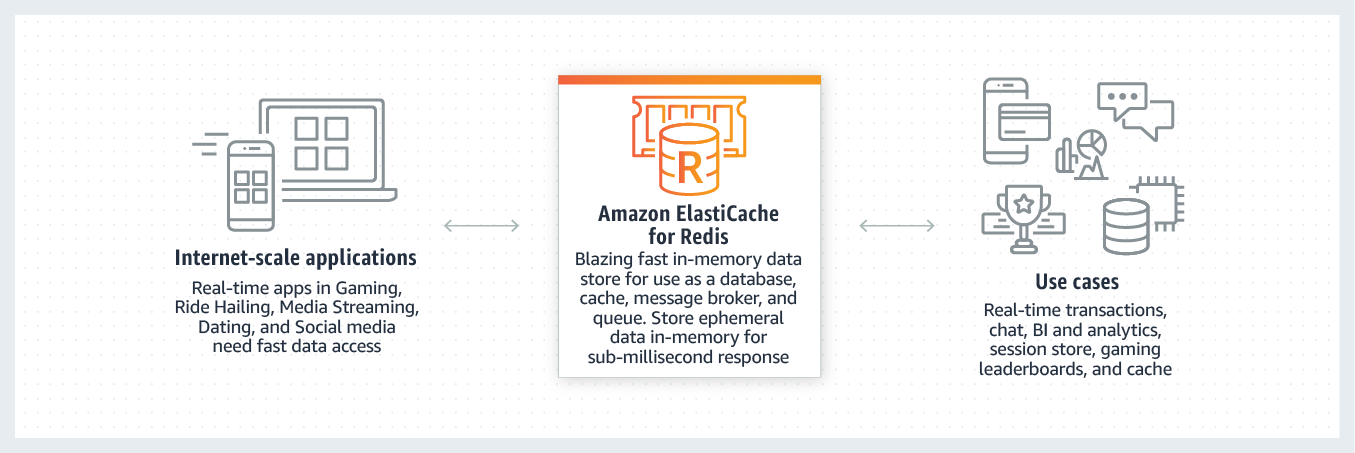
A developer is moving a legacy web application from their on-premises data center to AWS. The application is used simultaneously by thousands of users, and their session states are stored in memory. The on-premises server usually reaches 100% CPU Utilization every time there is a surge in the number of people accessing the application.

Which of the following is the best way to re-factor the performance and availability of the application’s session management once it is migrated to AWS?

* + Use Sticky Sessions with Local Session Caching.
  + Use an ElastiCache for Memcached cluster to store the user session state of the application.
  + Store the user session state of the application using CloudFront.
  + **Use an ElastiCache for Redis cluster to store the user session state of the application.**

**Correct**

**Amazon ElastiCache** for Redis is a blazing fast in-memory data store that provides sub-millisecond latency to power internet-scale real-time applications. Built on open-source Redis and compatible with the Redis APIs, ElastiCache for Redis works with your Redis clients and uses the open Redis data format to store your data. Your self-managed Redis applications can work seamlessly with ElastiCache for Redis without any code changes. ElastiCache for Redis combines the speed, simplicity, and versatility of open-source Redis with manageability, security, and scalability from Amazon to power the most demanding real-time applications in Gaming, Ad-Tech, E-Commerce, Healthcare, Financial Services, and IoT.



In order to address scalability and provide a shared data storage for sessions that can be accessible from any individual web server, you can abstract the HTTP sessions from the web servers themselves. A common solution to for this is to leverage an In-Memory Key/Value store such as Redis and Memcached. While Key/Value data stores are known to be extremely fast and provide sub-millisecond latency, the added network latency and added cost are the drawbacks. An added benefit of leveraging Key/Value stores is that they can also be utilized to cache any data, not just HTTP sessions, which can help boost the overall performance of your applications.

With Redis, you can keep your data on disk with a point in time snapshot which can be used for archiving or recovery. Redis also lets you create multiple replicas of a Redis primary. This allows you to scale database reads and to have highly available clusters. Hence, the correct answer for this scenario is to **use an ElastiCache for Redis cluster to store the user session state of the application.**

The option that says: **Store the user session state of the application using CloudFront**is incorrect because CloudFront is not suitable for storing user session data. It is primarily used as a content delivery network.

The option that says: **Use an ElastiCache for Memcached cluster to store the user session state of the application**is incorrect. Although using ElastiCache is a viable answer, Memcached is not as highly available as Redis.

The option that says: **Use Sticky Sessions with Local Session Caching** is incorrect. Although this is also a viable solution, it doesn’t offer durability and high availability compared to a distributed session management solution. The best solution for this scenario is to use an ElastiCache for Redis cluster.

**References:**

<https://aws.amazon.com/caching/session-management>

<https://aws.amazon.com/elasticache/redis-vs-memcached/>

<https://aws.amazon.com/elasticache/redis/>

**Check out this Amazon Elasticache Cheat Sheet:**

<https://tutorialsdojo.com/amazon-elasticache/>

1. **8. Question**

A developer is building an e-commerce application which will be hosted in an ECS Cluster. To minimize the number of instances in use, she must select a strategy which will place tasks based on the least available amount of CPU or memory.

Which of the following task placement strategy should the developer implement?

* + random
  + spread
  + distinctInstance
  + **binpack**

**Correct**

A **task placement strategy** is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new service.

Amazon ECS supports the following task placement strategies:

**binpack** – Place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use.

**random** – Place tasks randomly.

**spread** – Place tasks evenly based on the specified value. Accepted values are attribute key-value pairs, *instanceId*, or *host*.

The scenario states that the developer must select a task placement strategy which will place tasks based on the least available amount of CPU or memory. By using bin pack strategy with CPU as the field parameter, ECS is able to place tasks onto an instance with the least available amount of CPU first, before moving on to the other instances. Hence, the correct answer is to use the **binpack** task placement strategy.

**random** is incorrect because this will place the tasks randomly, rather than placing the tasks to the instances based on the least available amount of CPU or memory.

**spread** is incorrect because this will place tasks evenly to the instances based on a specified value.

**distinctInstance** is incorrect because this is not a valid task placement strategy, but a task placement constraint. This is primarily used as a constraint to place each task on a different container instance. It can be specified when either running a task or creating a new service.

**References:**

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-placement.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-placement-strategies.html>

<https://aws.amazon.com/blogs/compute/amazon-ecs-task-placement/>

**Check out this Amazon ECS Cheat Sheet:**

<https://tutorialsdojo.com/amazon-elastic-container-service-amazon-ecs/>

1. **9. Question**

A developer uses AWS SAM templates to deploy a serverless application. He needs to embed the application from the AWS Serverless Application Repository or from an S3 bucket as a nested application.

Which of the following resource type is the most SUITABLE one that the developer should use?

* + **AWS::Serverless::Application**
  + AWS::Serverless::LayerVersion
  + AWS::Serverless::Api
  + AWS::Serverless::Function

**Correct**

A serverless application can include one or more **nested applications**. You can deploy a nested application as a stand-alone artifact or as a component of a larger application.

As serverless architectures grow, common patterns emerge in which the same components are defined in multiple application templates. You can now separate out common patterns as dedicated applications, and then nest them as part of new or existing application templates. With nested applications, you can stay more focused on the business logic that’s unique to your application.

To define a nested application in your serverless application, use the [AWS::Serverless::Application](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-template.html#serverless-sam-template-application) resource type.

**AWS::Serverless::Function** is incorrect because this resource type describes configuration information for creating a Lambda function. You can describe any event source that you want to attach to the Lambda function—such as Amazon S3, Amazon DynamoDB Streams, and Amazon Kinesis Data Streams.

**AWS::Serverless::LayerVersion** is incorrect because this resource type creates a Lambda layer version (LayerVersion) that contains library or runtime code that’s needed by a Lambda function. When a serverless layer version is transformed, AWS SAM also transforms the logical ID of the resource so that old layer versions aren’t automatically deleted by AWS CloudFormation when the resource is updated.

**AWS::Serverless::Api** is incorrect because this resource type describes an API Gateway resource. It’s useful for advanced use cases where you want full control and flexibility when you configure your APIs. For most scenarios, it is recommended that you create APIs by specifying this resource type as an event source of your AWS::Serverless::Function resource.

**References:**

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-template.html#serverless-sam-template-application>

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-template-nested-applications.html>

1. **10. Question**

The users of a social media website must be authenticated using social identity providers such as Twitter, Facebook, and Google. Users can login to the site which will allow them to upload their selfies, memes, and other media files in an S3 bucket. As an additional feature, you should also enable guest user access to certain sections of the website.

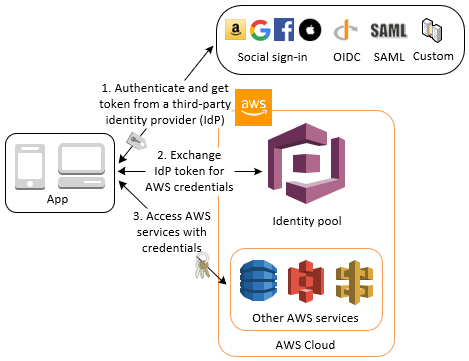
Which of the following should you do to accomplish this task?

* + Integrate AWS IAM Identity Center with your website.
  + **Create an Identity Pool in Amazon Cognito and enable access to unauthenticated identities.**
  + Create a custom identity broker which integrates with the AWS Security Token Service and supports unauthenticated access.
  + **Create a User Pool in Amazon Cognito and enable access to unauthenticated identities.**

**Incorrect**

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a username and password or through a third party such as Facebook, Amazon, or Google.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.



Amazon Cognito identity pools (federated identities) support user authentication through Amazon Cognito user pools, federated identity providers—including Amazon, Facebook, Google, and SAML identity providers—as well as unauthenticated identities. This feature also supports [Developer Authenticated Identities (Identity Pools)](https://docs.aws.amazon.com/cognito/latest/developerguide/developer-authenticated-identities.html), which lets you register and authenticate users via your own back-end authentication process.

Hence, the correct answer is: **Create an Identity Pool in Amazon Cognito and enabling access to unauthenticated identities**

The option that says: **Create a User Pool in Amazon Cognito and enable unauthenticated identities**is incorrect because you should have created an Identity Pool instead. Take note that a User Pool doesn’t have the option to enable unauthenticated identities. Moreover, you won’t be able to provide your users access to upload their media files to S3 using a User Pool.

The option that says: **Create a custom identity broker which integrates with the AWS Security Token Service and supports unauthenticated access**is incorrect because this is not a suitable solution in this scenario. You only need to build a custom identity broker application if your identity store is not compatible with SAML 2.0, which is required for identity federation.

The option that says: **Integrate AWS IAM Identity Center** is incorrect because this is only used to help you manage access and permissions to custom applications that support Security Assertion Markup Language (SAML) 2.0 and commonly used third-party software as a service (SaaS) applications. This is primarily used for existing corporate identities and not for social identity providers.

**References:**

<https://aws.amazon.com/premiumsupport/knowledge-center/cognito-user-pools-identity-pools/>

<https://docs.aws.amazon.com/cognito/latest/developerguide/getting-started-with-identity-pools.html>

**Check out this Amazon Cognito Cheat Sheet:**

<https://tutorialsdojo.com/amazon-cognito/>

1. **11. Question**

A new IT policy requires you to trace all calls that your [Node.js](http://node.js/) application sends to external HTTP web APIs as well as SQL database queries. You have to instrument your application, which is hosted in Elastic Beanstalk, in order to properly trace the calls via the X-Ray console .

What should you do to comply with the given requirement?

* + **Create a Docker image that runs the X-Ray daemon.**
  + Use a user data script to run the daemon automatically.
  + Enable active tracing in the Elastic Beanstalk by including the healthcheckurl.config configuration file in the .ebextensions directory of your source code.
  + **Enable the X-Ray daemon by including the xray-daemon.config configuration file in the .ebextensions directory of your source code.**

**Incorrect**

You can use the **AWS Elastic Beanstalk** console or a configuration file to run the AWS X-Ray daemon on the instances in your environment. X-Ray is an AWS service that gathers data about the requests that your application serves, and uses it to construct a service map that you can use to identify issues with your application and opportunities for optimization.



To relay trace data from your application to AWS X-Ray, you can run the X-Ray daemon on your Elastic Beanstalk environment’s Amazon EC2 instances. Elastic Beanstalk platforms provide a configuration option that you can set to run the daemon automatically. You can enable the daemon in a configuration file in your source code or by choosing an option in the Elastic Beanstalk console. When you enable the configuration option, the daemon is installed on the instance and runs as a service.

Hence, the correct answer is to: **enable the X-Ray daemon by including the xray-daemon.config configuration file in the .ebextensions directory of your source code**, just as shown above.

**Using a user data script to run the daemon automatically**is incorrect because this is only applicable if you want to enable X-Ray to your EC2 instances.

**Creating a Docker image that runs the X-Ray daemon**is incorrect because this is what you need to do if you want to enable X-Ray on ECS Cluster and not on Elastic Beanstalk.

**Enabling active tracing in the Elastic Beanstalk by including the healthcheckurl.config configuration file in the .ebextensions directory of your source code**is incorrect because this configuration file only sets the application health check URL and not X-Ray Tracing.

**References:**

<https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-beanstalk.html>

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-configuration-debugging.html>

**Check out these AWS X-Ray and Elastic Beanstalk Cheat Sheets:**

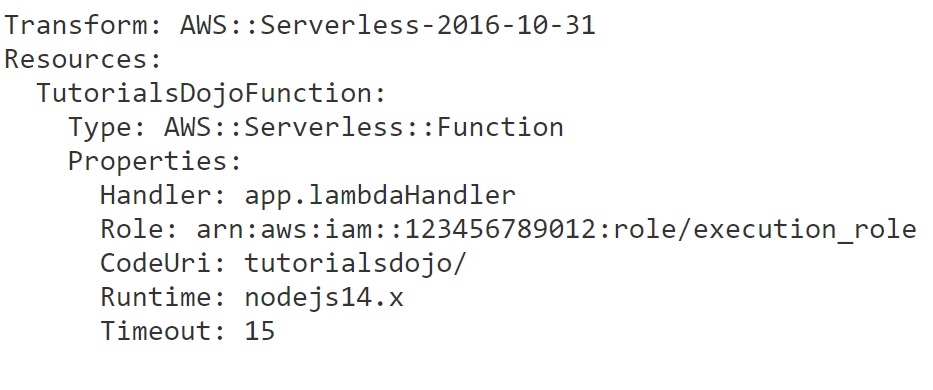
<https://tutorialsdojo.com/aws-x-ray/>

<https://tutorialsdojo.com/aws-elastic-beanstalk/>

1. **12. Question**

A development team has started using AWS CloudFormation for deploying Lambda functions. Their project structure places the source code for the Lambda function locally within a directory named “tutorialsdojo”. The lambda handler for the function is in a file called “app.js”.

Below is a snippet of their template.

What is the next step in order for the template to be deployed using the aws cloudformation deploy CLI?

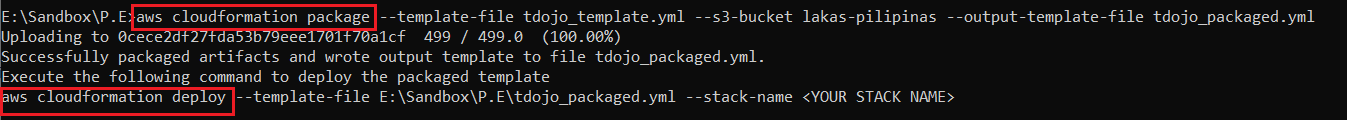
* + Use the Fn::Base64 intrinsic function inline with the CodeUri property to encode the content of the app.js file.
  + Package the Lambda function in a ZIP file. Specify the local path of the packaged file in the CodeUri property.
  + Upload the app.js file to an S3 bucket. Update the CodeUri property in the template to point to the S3 URI of the file.
  + **Use the aws cloudformation package command to upload the local artifacts of the Lambda function to an S3 bucket and produce a version of the template with references to the S3 URI of the file.**

**Correct**

The aws cloudformation package command packages the local artifacts (local paths) that your AWS CloudFormation template references. The command uploads local artifacts, such as source code for an AWS Lambda function or a Swagger file for an AWS API Gateway REST API, to an S3 bucket. The command returns a copy of your template, replacing references to local artifacts with the S3 location where the command uploaded the artifacts.

Use this command to quickly upload local artifacts that might be required by your template. After you package your template’s artifacts, run the aws cloudformation deploy command to deploy the returned template.

Since we have local artifacts (source code for the AWS Lambda functions), we should use the package command.



After running the package command, the modified template can now be deployed by using the deploy command.

Hence, the correct answer is: **Use the aws cloudformation package command to upload the local artifacts of the Lambda function to an S3 bucket and produce a version of the template with references to the S3 URI of the file.**

The option that says: **Upload the app.js file to an S3 bucket. Update the CodeUri property in the template to point to the S3 URI of the file**is incorrect. Simply uploading the app.js file to an S3 bucket without packaging it into a ZIP format is not adequate for deployment. AWS Lambda expects the deployment package to be in a ZIP format for the Node.js runtime.

The option that says: **Use the Fn::Base64 intrinsic function inline with the CodeUri property to encode the content of the app.js file** is incorrect. Embedding the base64 encoded content of the app.js file within the template would result in deployment errors. Note that the CodeUri property specifically expects either a local path to the Lambda function code or an S3 URI pointing to the zipped code.

The option that says: **Package the Lambda function in a ZIP file. Specify the local path of the packaged file in the CodeUri property**is incorrect. The CodeUri property expects either a local path to the directory containing the Lambda function source code or an S3 URI to the ZIP file. Specifying a local path to a ZIP file is wrong, as AWS CloudFormation wouldn’t be able to directly deploy the function with the ZIP file.

**References:**

<https://docs.aws.amazon.com/cli/latest/reference/cloudformation/package.html>

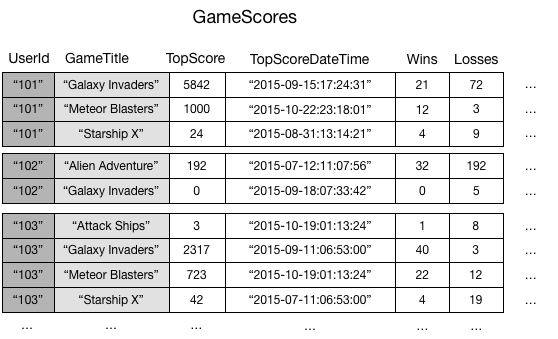
<https://docs.aws.amazon.com/cli/latest/reference/cloudformation/deploy/index.html>

**Check out this AWS CloudFormation Cheat Sheet:**

<https://tutorialsdojo.com/aws-cloudformation/>

1. **13. Question**

A mobile game is using a DynamoDB table named GameScore that keeps track of users and scores. Each item in the table is identified by a partition key (UserId) and a sort key (GameTitle). The diagram below shows how the items in the table are organized:



A developer wants to write a leaderboard application to display the top scores for each game.

How can the developer meet the requirement in the MOST efficient manner?

* + **Create a global secondary index. Assign the GameTitle attribute as the partition key and the TopScore attribute as the sort key.**
  + Create a local secondary index. Assign the TopScore attribute as the partition key and the GameTitle attribute as the sort key.
  + Create a local secondary index. Assign the GameTitle attribute as the partition key and the TopScore attribute as the sort key.
  + Use the Scan operation and filter the results based on a GameTitle value.

**Correct**

**Amazon DynamoDB** provides fast access to items in a table by specifying primary key values. However, many applications might benefit from having one or more secondary (or alternate) keys available, to allow efficient access to data with attributes other than the primary key. To address this, you can create one or more secondary indexes on a table, and issue Query or Scan requests against these indexes.

A ***secondary index*** is a data structure that contains a subset of attributes from a table, along with an alternate key to support Query operations. You can retrieve data from the index using a Query, in much the same way as you use Query with a table. A table can have multiple secondary indexes, which gives your applications access to many different query patterns.

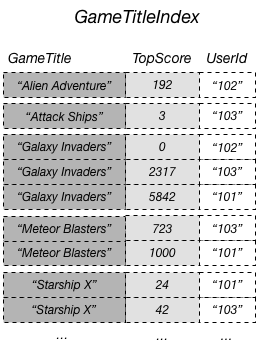
DynamoDB supports two types of secondary indexes:

[**Global secondary index**](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html)**—**an index with a partition key and a sort key that can be different from those on the base table. A global secondary index is considered “global” because queries on the index can span all of the data in the base table, across all partitions.

[**Local secondary index**](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/LSI.html)**—**an index that has the same partition key as the base table, but a different sort key. A local secondary index is “local” in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value.

To speed up queries on non-key attributes, you can create a global secondary index. A global secondary index contains a selection of attributes from the base table, but they are organized by a primary key that is different from that of the table. The index key does not need to have any of the key attributes from the table; it doesn’t even need to have the same key schema as a table.

In this scenario, you could create a global secondary index named *GameTitleIndex*, with a partition key of *GameTitle* and a sort key of *TopScore*. Since the base table’s primary key attributes are always projected into an index, the UserId attribute is also present. The following diagram shows what *GameTitleIndex* index would look like:



Hence, the correct answer in this scenario is to: **Create a global secondary index. Assign the GameTitle attribute as the partition key and the TopScore attribute as the sort key.**

**Create a local secondary index. Assign the TopScore attribute as the partition key and the GameTitle attribute as the sort key** is incorrect. You can’t add a local secondary index to an existing table. Moreover, even if it’s possible, making a query that returns the top scores for each game is impossible with the TopScore attribute as the partition key. When you issue a query, you must also specify a partition key. In this case, if you run a query with a partition key value of 500, the results might return different games with a score of 500 from various users. It does not tell if it’s the highest score in that game.

**Create a local secondary index. Assign the GameTitle attribute as the partition key and the TopScore attribute as the sort key**is incorrect because you can’t add this index to an already existing table. Additionally, a local secondary index has the same partition key as the base table, but has a different sort key. It is “local” in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value.

**Use the Scan operation and filter the results based on a GameTitle value**is incorrect. Technically, this also works but it is less efficient and slower compared to querying on secondary indexes. The Scan operation reads every item in a table. As the table grows, the slower the Scan operation would become.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

**DynamoDB Scan vs Query:**

<https://tutorialsdojo.com/dynamodb-scan-vs-query/>

1. **14. Question**

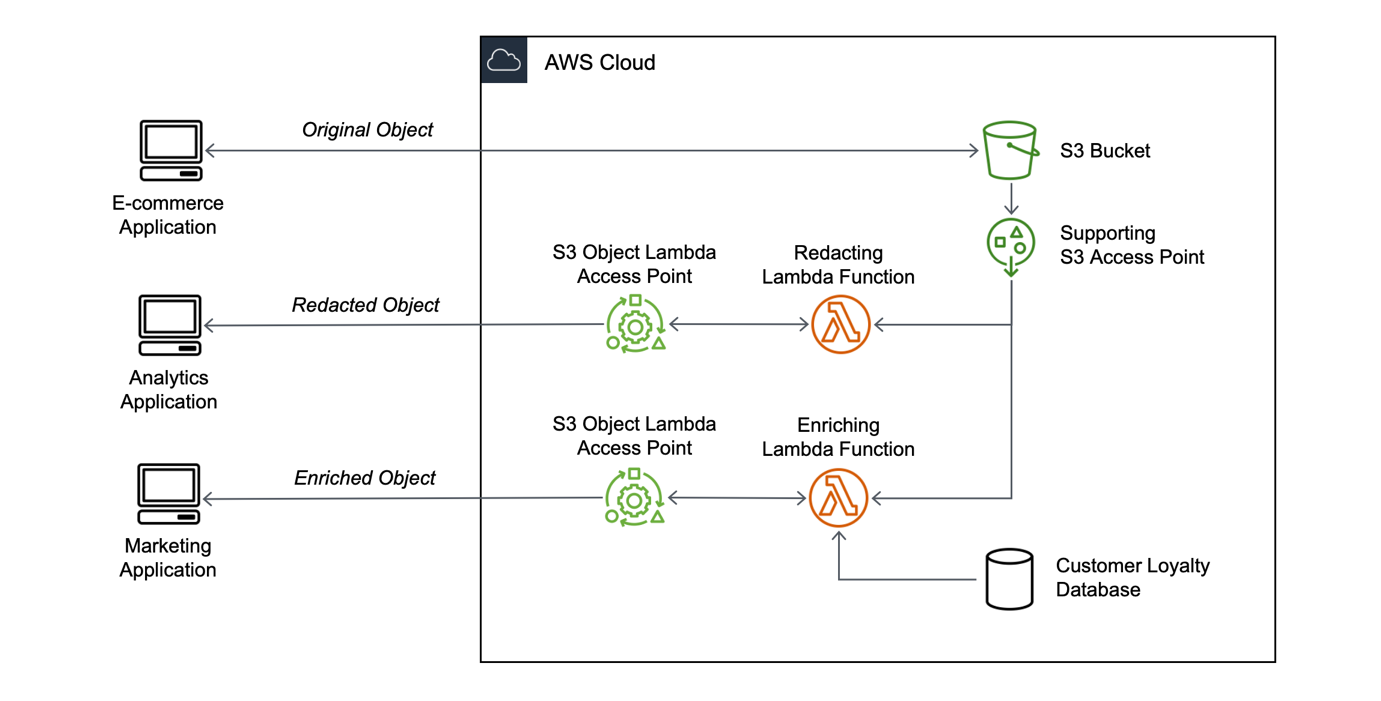
A recruitment agency has a large collection of resumes stored in an Amazon S3 bucket. The agency wants to perform an analysis on these files, but for privacy compliance reasons, they need to ensure that certain personally identifiable information (PII) is redacted before being processed by their internal service.

Which solution can meet the requirements in the most cost-effective way?

* + **Use Amazon S3 Object Lambda to redact PII before it is returned to the application.**
  + Implement a solution with AWS Glue to transform the data and redact PII before storing it in an S3 bucket.
  + **Configure an Amazon S3 Access Point and set up an Amazon CloudFront distribution with a Lambda@Edge function to redact the PII as data is fetched from the S3 bucket.**
  + Use a Lambda function to create a redacted copy of each file in a separate S3 bucket. Then, set up an Amazon S3 Access Point to serve these files.

**Incorrect**

S3 Object Lambda allows you to add your own code to S3 GET requests to modify and process data as it’s being returned to an application. This feature is designed for use cases where data needs to be transformed on-the-fly without the need to store a transformed copy of the data. It’s useful in scenarios like filtering rows, redacting confidential data, dynamically resizing images and other similar situations where data transformation or processing is required during data retrieval.



In the scenario, when the internal service fetches a file from the S3 bucket, S3 Object Lambda will run a Lambda function to redact the PII from the data as it is being retrieved before it is returned to the application. This eliminates the need to create and store a separate, redacted copy of each resume, thereby saving on storage costs. Plus, since the redaction happens during data retrieval, there’s no need to create a proxy for the internal service. This makes the solution efficient and cost-effective.

Hence, the correct answer is: **Use Amazon S3 Object Lambda to redact PII before it is returned to the application.**

The option that says: **Configure an Amazon S3 Access Point and set up an Amazon CloudFront distribution with a Lambda**[**@Edge**](https://portal.tutorialsdojo.com/members/edge/)**function to redact the PII as data is fetched from the S3 bucket** is incorrect. Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) functions operate at the CDN edge locations, making them ideal for use cases that need low-latency responses to end users. However, in the scenario, the requirement is to redact PII before processing by an internal service, not necessarily to serve end users quickly. Moreover, Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) is generally more expensive to run than a regular Lambda function.

The option that says: **Implement a solution with AWS Glue to transform the data and redact PII before storing it in the S3 bucket** is incorrect. AWS Glue is an ETL service designed for complex transformations and analytics. With this approach, you’d create redacted copies of resumes, leading to increased storage costs. Also, using AWS Glue to merely redact PII might be overkill and less cost-effective compared to simpler solutions like S3 Object Lambda.

The option that says: **Use a Lambda function to create a redacted copy of each file in a separate S3 bucket. Then, set up an Amazon S3 Access Point to serve these files** is incorrect. In this approach, storing redacted copies of data in S3 would increase storage costs. However, with S3 Object Lambda, on-the-fly redaction of PII becomes possible, eliminating the need for storing separate redacted copies of the data.

**References:**

<https://aws.amazon.com/s3/features/object-lambda/>

<https://aws.amazon.com/blogs/aws/introducing-amazon-s3-object-lambda-use-your-code-to-process-data-as-it-is-being-retrieved-from-s3/>

**Check out this Amazon S3 Cheat Sheet:**

<https://tutorialsdojo.com/amazon-s3/>

1. **15. Question**

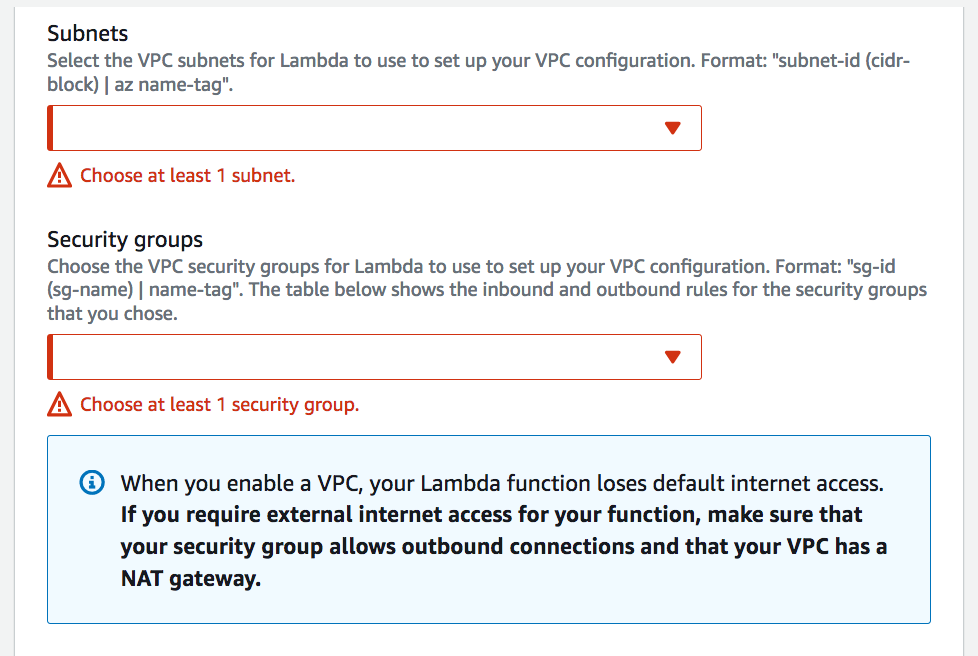
A developer is refactoring a Lambda function that currently processes data using a public GraphQL API. There’s a new requirement to store query results in a database hosted in a VPC. The function has been configured with additional VPC-specific information, and the database connection has been successfully established. However, the engineer has discovered that the function can no longer connect to the internet after testing.

Which of the following should the developer do to fix this issue? (Select TWO.)

* + Configure your function to forward payloads that were not processed to a dead-letter queue (DLQ) using Amazon SQS.
  + Set up elastic network interfaces (ENIs) to enable your Lambda function to connect securely to other resources within your private VPC.
  + **Ensure that the associated security group of the Lambda function allows outbound connections.**
  + **Add a NAT gateway to your VPC.**
  + Submit a limit increase request to AWS to raise the concurrent executions limit of your Lambda function.

**Correct**

AWS Lambda uses the VPC information you provide to set up ENIs that allow your Lambda function to access VPC resources. Each ENI is assigned a private IP address from the IP address range within the subnets you specify but is not assigned any public IP addresses.



Therefore, if your Lambda function requires Internet access (for example, to access AWS services that don’t have VPC endpoints ), you can configure a NAT instance inside your VPC, or you can use the Amazon VPC NAT gateway. You cannot use an Internet gateway attached to your VPC since that requires the ENI to have public IP addresses.

If your Lambda function needs Internet access, just as described in this scenario, do not attach it to a public subnet or to a private subnet without Internet access. Instead, attach it only to private subnets with Internet access through a NAT instance or **add a NAT gateway to your VPC**. You should also **ensure that the associated security group of the Lambda function allows outbound connections.**

The option that says: **Submit a limit increase request to AWS to raise the concurrent executions limit of your Lambda function**is incorrect because the root cause of the problem is that the function cannot connect to public GraphQL APIs over the Internet. The scenario doesn’t mention anything about a concurrency problem.

The option that says: **Configuring your function to forward payloads that were not processed to a dead-letter queue (DLQ) using Amazon SQS**is incorrect because it will only improve the error handling of your Lambda function. The issue here is the Internet connectivity of your function and not its error handling hence, this option will not solve the problem.

The option that says: **Setting up elastic network interfaces (ENIs) to enable your Lambda function to connect securely to other resources within your private VPC** is incorrect because this is already done automatically by AWS Lambda. It uses the VPC information you provide to automatically set up ENIs that allow your Lambda function to access VPC resources. You don’t need to do this step in order for your Lambda function to be integrated with your VPC.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/vpc.html>

<https://aws.amazon.com/premiumsupport/knowledge-center/internet-access-lambda-function/>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **16. Question**

A developer is planning to deploy a high-performance online trading application which requires a database that can scale globally and can handle frequent schema changes. The database should also support flexible schemas that enable faster and more iterative development.

Which is the MOST suitable database service that you should use to achieve this requirement?

* + Amazon Aurora
  + Amazon RDS
  + Amazon Redshift
  + **Amazon DynamoDB**

**Correct**

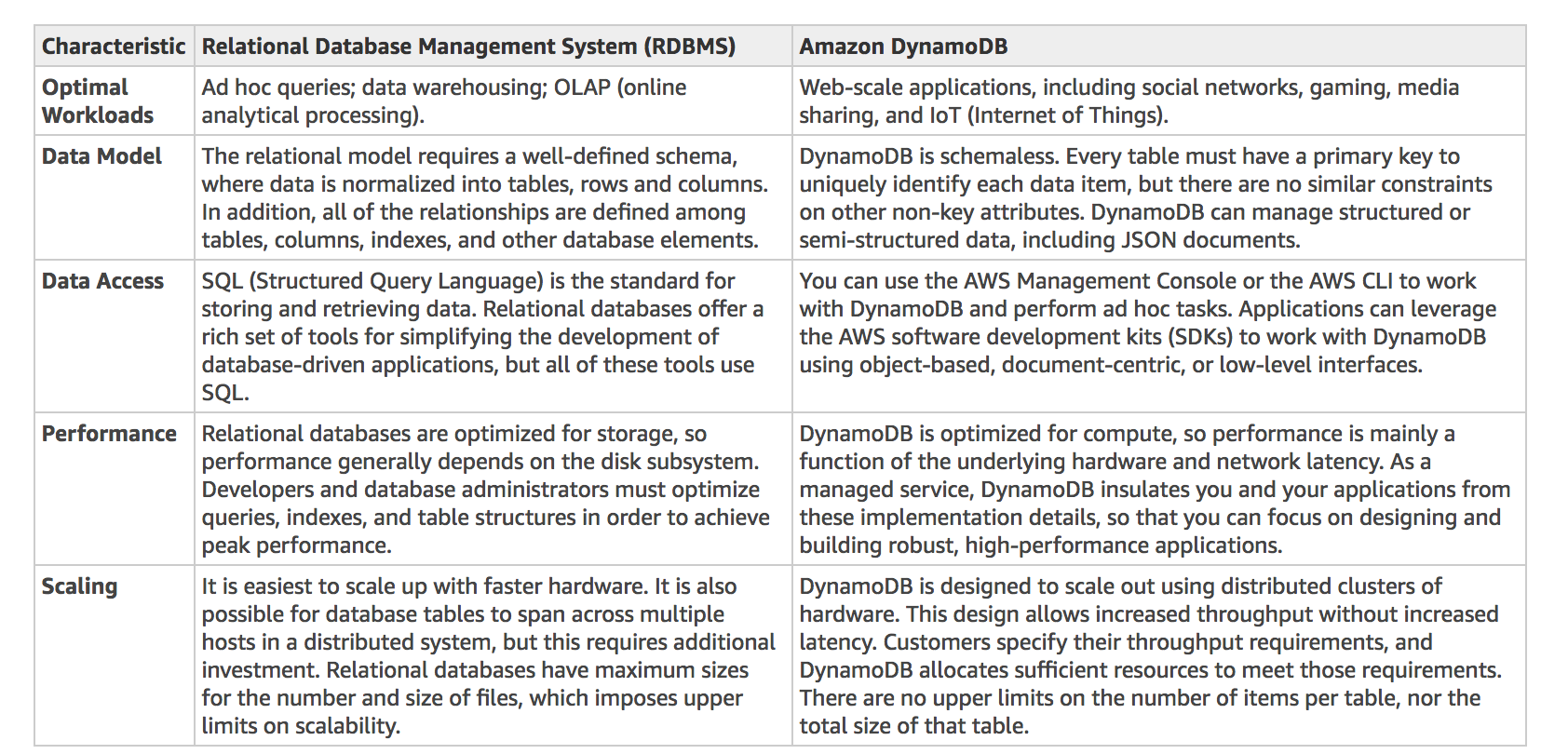
NoSQL databases are a great fit for many modern applications such as mobile, web, and gaming that require flexible, scalable, high-performance, and highly functional databases to provide great user experiences.

**Flexibility:** NoSQL databases generally provide flexible schemas that enable faster and more iterative development. The flexible data model makes NoSQL databases ideal for semi-structured and unstructured data.

**Scalability:** NoSQL databases are generally designed to scale out by using distributed clusters of hardware instead of scaling up by adding expensive and robust servers. Some cloud providers handle these operations behind-the-scenes as a fully managed service.

**High-performance:**NoSQL databases are optimized for specific data models (such as document, key-value, and graph) and access patterns that enable higher performance than trying to accomplish similar functionality with relational databases.

**Highly functional:** NoSQL databases provide highly functional APIs and data types that are purpose-built for each of their respective data models.



A relational database is known for having a rigid schema, with a lot of constraints and limits as to which (and what type of) data can be inserted or not. It is primarily used for scenarios where you have to support complex queries which fetch data across a number of tables. It is best for scenarios where you have complex table relationships but for use cases where you need to have a flexible schema, this is not a suitable database to use.

For NoSQL, it is not as rigid as a relational database because you can easily add or remove rows or elements in your table/collection entry. It also has a more flexible schema because it can store complex hierarchical data within a single item which, unlike a relational database, does not entail changing multiple related tables. Hence, the best answer to be used here is a NoSQL database, like DynamoDB. When your business requires a low-latency response to high-traffic queries, taking advantage of a NoSQL system generally makes technical and economic sense.

Amazon DynamoDB helps solve the problems that limit the relational system scalability by avoiding them. In DynamoDB, you design your schema specifically to make the most common and important queries as fast and as inexpensive as possible. Your data structures are tailored to the specific requirements of your business use cases.

Therefore, the correct answer is **Amazon DynamoDB**for this scenario.

**Amazon RDS** and **Amazon Aurora** are both incorrect because both of them are a type of relational database that doesn’t provide flexible schemas, unlike DynamoDB. Although it can scale globally, it would entail a lot of configuration and setup to do so. Moreover, a relational database has a rigid schema which is not suitable for frequent schema changes.

**Amazon Redshift** is incorrect because this is not a NoSQL database and is primarily used for OLAP systems.

**References:**

<https://aws.amazon.com/nosql/>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-general-nosql-design.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb>

**RDS vs DynamoDB:**

<https://tutorialsdojo.com/amazon-rds-vs-dynamodb/>

**Comparison of AWS Services Cheat Sheets:**

<https://tutorialsdojo.com/comparison-of-aws-services-for-udemy-students/>

1. **17. Question**

An online stock trading platform is hosted in an Auto Scaling group of EC2 instances with an Application Load Balancer in front to distribute the incoming traffic evenly. The developer must capture information about the IP traffic going to and from network interfaces in your VPC to comply with financial regulatory requirements.

Which of the following options should the developer do to meet the requirement?

* + Use AWS Inspector to capture information about the IP traffic going to and from the network interfaces of your EC2 instances.
  + Use CloudTrail logs to track all API calls and capture information about the IP traffic going to and from your VPC.
  + Install and run the AWS X-Ray daemon to your EC2 instances using an instance metadata script.
  + **Create a flow log in your VPC.**

**Correct**

VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from network interfaces in your VPC. Flow log data can be published to Amazon CloudWatch Logs and Amazon S3. After you’ve created a flow log, you can retrieve and view its data in the chosen destination.



Flow logs can help you with a number of tasks; for example, to troubleshoot why specific traffic is not reaching an instance, which in turn helps you diagnose overly restrictive security group rules. You can also use flow logs as a security tool to monitor the traffic that is reaching your instance. CloudWatch Logs charges apply when using flow logs, whether you send them to CloudWatch Logs or to Amazon S3.

Hence, you should **create a flow log in your VPC** to capture information about the IP traffic going to and from network interfaces in your VPC.

**Using CloudTrail logs to track all API calls and capture information about the IP traffic going to and from your VPC**is incorrect. Although you can indeed use CloudTrail to track the API call, it can’t capture information about the IP traffic of your VPC.

**Installing and running the AWS X-Ray daemon to your EC2 instances using an instance metadata script** is incorrect because you have to use a user data script and not a metadata. Alternatively, you can instrument your application which is running in an EC2 instance to capture the client’s IP address. However, it is much easier to just enable VPC Flow Logs to meet the requirement.

**Using AWS Inspector to capture information about the IP traffic going to and from the network interfaces of your EC2 instances** is incorrect because this service is just an automated security assessment service that helps improve the security and compliance of applications deployed on AWS. It doesn’t have the ability to capture IP traffic of your VPC.

**References:**

<https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html>

<https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-ec2.html>

<https://docs.aws.amazon.com/xray/latest/devguide/xray-concepts.html>

**Check out these Amazon VPC and AWS X-Ray Cheat Sheets:**

<https://tutorialsdojo.com/amazon-vpc/>

<https://tutorialsdojo.com/aws-x-ray/>

1. **18. Question**

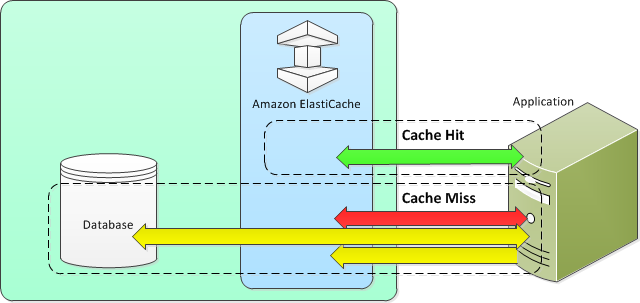
A web application is using an ElastiCache cluster that is suffering from cache churn. A developer needs to reconfigure the application so that data are retrieved from the database only in the event that there is a cache miss.

Which pseudocode illustrates the caching strategy that the developer needs to implement?

* + get\_item(item\_id):
  + item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)
  + if item\_value is None:
  + item\_value = cache.set(item\_id, item\_value)
  + cache.add(item\_id, item\_value)
  + return item\_value
  + **get\_item(item\_id):**
  + **item\_value = cache.get(item\_id)**
  + **if item\_value is None:**
  + **item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)**
  + **cache.add(item\_id, item\_value)**
  + **return item\_value**
  + get\_item(item\_id):
  + item\_value = cache.get(item\_id)
  + if item\_value is not None:
  + item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)
  + cache.add(item\_id, item\_value)
  + return item\_value
  + else:
  + return item\_value
  + get\_item(item\_id, item\_value):
  + item\_value = database.query("UPDATE Items WHERE id = ?", item\_id, item\_value)
  + cache.add(item\_id, item\_value)
  + return 'ok'

**Correct**

Lazy Loading is a caching strategy that loads data into the cache only when necessary. Here is how it works:



– If the data exists in the cache and is current, ElastiCache returns the data to your application. This event is also called “**Cache Hit**“.

– If there is a “**Cache Miss**“, or in other words, the data does not exist in the cache, or the data in the cache has expired, then your application requests the data from your data store, which returns the data to your application. Your application then writes the data received from the store to the cache so it can be more quickly retrieved the next time it is requested.

In the scenario, to implement lazy loading, you must first check if the item is already in the cache using the “cache.get(item\_id)” method. If the item is not in the cache (i.e. “item\_value is None”), the code then queries the database for the item and stores it in the cache using the “cache.set(item\_id, item\_value)” method so that it can be retrieved faster next time. This way, the application is not querying the database every time the item is needed and instead uses the cached version of the item if it’s available.

Hence, the correct answer is the option that says:

**get\_item(item\_id):**

**item\_value = cache.get(item\_id)**

**if item\_value is None:**

**item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)**

**cache.add(item\_id, item\_value)**

**return item\_value**

The option that says:

**get\_item(item\_id):**

**item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)**

**if item\_value is None:**

**item\_value = cache.set(item\_id, item\_value)**

**cache.add(item\_id, item\_value)**

**return item\_value**

is an incorrect implementation of lazy loading because it first queries the database and checks if the item is in the cache.

The option that says:

**get\_item(item\_id):**

**item\_value = cache.get(item\_id)**

**if item\_value is not None:**

**item\_value = database.query("SELECT \* FROM Items WHERE id = ?", item\_id)**

**cache.add(item\_id, item\_value)**

**return item\_value**

**else:**

**return item\_value**

does not implement lazy loading because the code is not utilizing the cache first and is querying the database every time the item is needed, this will make the application slow and inefficient.

The option that says:

**get\_item(item\_id, item\_value):**

**item\_value = database.query("UPDATE Items WHERE id = ?", item\_id, item\_value)**

**cache.add(item\_id, item\_value)**

**return 'ok'**

is incorrect because this is an implementation of a write-through caching strategy where data is written to both the cache and the primary storage (such as a database).

**References:**

<https://aws.amazon.com/caching/best-practices/>

<https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Strategies.html>

**Check out this Amazon Elasticache Cheat Sheet:**

<https://tutorialsdojo.com/amazon-elasticache/>

1. **19. Question**

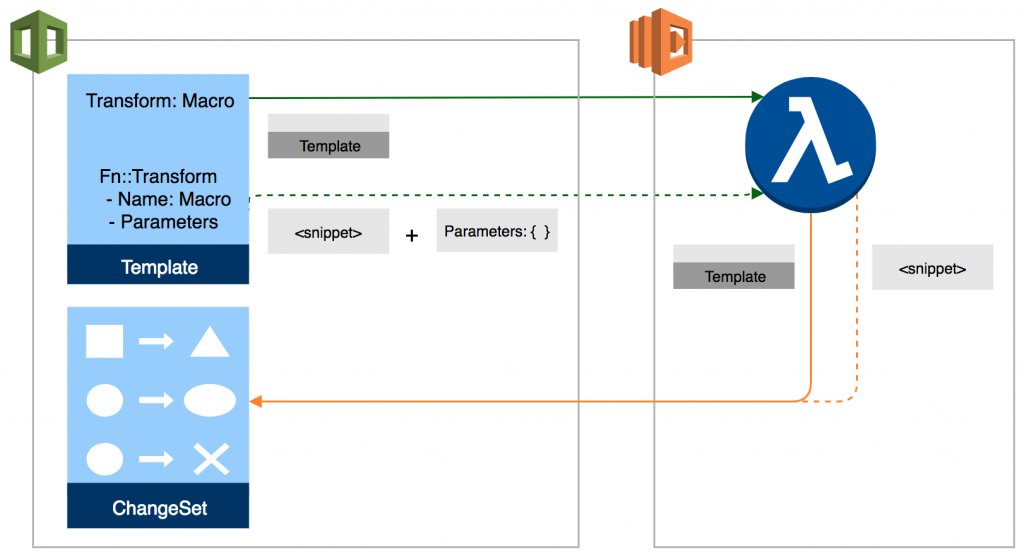
You are deploying a serverless application composed of Lambda, API Gateway, CloudFront, and DynamoDB using CloudFormation. The AWS SAM syntax should be used to declare resources in your template which requires you to specify the version of the AWS Serverless Application Model (AWS SAM).

Which of the following sections is required, aside from the Resources section, that should be in your CloudFormation template?

* + Mappings
  + Format Version
  + **Transform**
  + Parameters

**Correct**

For serverless applications (also referred to as Lambda-based applications), the optional **Transform** section specifies the version of the AWS Serverless Application Model (AWS SAM) to use. When you specify a transform, you can use AWS SAM syntax to declare resources in your template. The model defines the syntax that you can use and how it is processed.



This section specifies one or more macros that AWS CloudFormation uses to process your template. The Transform section builds on the simple, declarative language of AWS CloudFormation with a powerful macro system.

You can declare one or more macros within a template. AWS CloudFormation executes macros in the order that they are specified. When you create a change set, AWS CloudFormation generates a change set that includes the processed template content. You can then review the changes and execute the change set.

AWS CloudFormation also supports the AWS::Serverless and AWS::Include transforms, which are macros hosted by AWS CloudFormation. AWS CloudFormation treats these transforms the same as any macros you create in terms of execution order and scope.

Therefore, the **Transform** section should be the correct one to be added to your template.

**Mappings** section is incorrect because this is just a literal mapping of keys and associated values that you can use to specify conditional parameter values, similar to a lookup table.

**Parameters** section is incorrect because this only contains the values that will be passed to your template at runtime (when you create or update a stack). You can refer to parameters from the Resources and Outputs sections of the template, but this is not used to specify the AWS SAM version.

**Format Version** section is incorrect because this just refers to the AWS CloudFormation template version that the template conforms to, and not the version of the AWS Serverless Application Model (AWS SAM)

**References:**

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html>

<https://aws.amazon.com/blogs/aws/cloudformation-macros/>

**Check out this AWS CloudFormation Cheat Sheet:**

<https://tutorialsdojo.com/aws-cloudformation/>

**Check out this AWS Serverless Application Model Cheat Sheet:**

<https://tutorialsdojo.com/aws-serverless-application-model-sam/>

1. **20. Question**

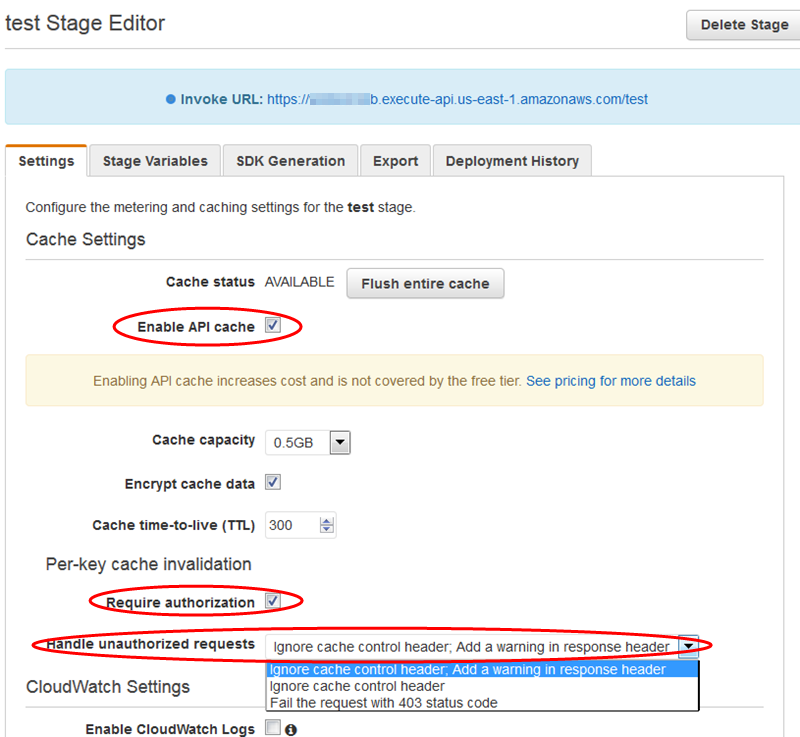
A web application is running in an ECS Cluster and updates data in DynamoDB several times a day. The clients retrieve data directly from the DynamoDB through APIs exposed by Amazon API Gateway. Although API caching is enabled, there are specific clients that want to retrieve the latest data from DynamoDB for every API request sent.

What should be done to only allow authorized clients to invalidate an API Gateway cache entry when submitting API requests? (Select TWO.)

* + The client must send a request which contains the Cache-Control: max-age=1 header.
  + **The client must send a request which contains the Cache-Control: max-age=0 header.**
  + Modify the cache settings to retrieve the latest data from DynamoDB if the request header's authorization signature matches your API's trusted clients list.
  + **Tick the Require Authorization checkbox in the Cache Settings of your API via the console.**
  + Provide your clients an authorization token from STS to query data directly from DynamoDB.

**Correct**

A client of your API can invalidate an existing cache entry and reload it from the integration endpoint for individual requests. The client must send a request that contains the Cache-Control: max-age=0 header. The client receives the response directly from the integration endpoint instead of the cache, provided that the client is authorized to do so. This replaces the existing cache entry with the new response, which is fetched from the integration endpoint.



Ticking the Require authorization checkbox ensures that not every client can invalidate the API cache. If most or all of the clients invalidate the API cache, this could significantly increase the latency of your API.

Hence, to only allow authorized clients to invalidate an API Gateway cache entry when submitting API requests, you can just **tick the Require Authorization checkbox in the Cache Settings of your API via the console and instruct the client to send a request which contains the Cache-Control: max-age=0 header.**

**Instructing the client to send a request which contains the Cache-Control: max-age=1 header** is incorrect because the value of the max-age should be 0 and not 1.

**Providing your clients an authorization token from STS to query data directly from DynamoDB** is incorrect because this will not enable your clients to invalidate the cache in API Gateway. Considering that your clients are using APIs to interact with DynamoDB, you should not provide them access to directly submit queries to your table but only through API Gateway.

**Modifying the cache settings to retrieve the latest data from DynamoDB if the request header’s authorization signature matches your API’s trusted clients list**is incorrect because this configuration can’t be done. There is no feature in API Gateway Cache Settings which would allow you to make a list of authorized signatures that are allowed to invalidate cache entries.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html#invalidate-method-caching>

<https://aws.amazon.com/api-gateway/faqs/#Throttling_and_Caching>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **21. Question**

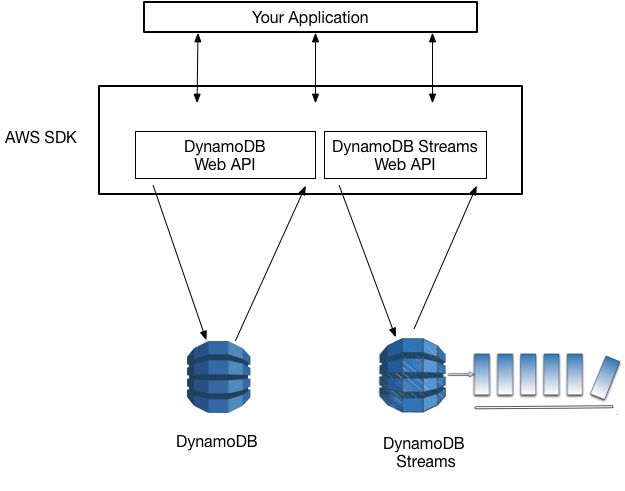
A developer is working on an application which stores data to an Amazon DynamoDB table with the DynamoDB Streams feature enabled. He set up an event source mapping with DynamoDB Streams and AWS Lambda function to monitor any table changes then store the original data of the overwritten item in S3. When an item is updated, it should only send a copy of the item’s previous value to an S3 bucket and maintain the new value in the DynamoDB table.

Which StreamViewType is the MOST suitable one to use in the DynamoDB configuration to fulfill this scenario?

* + KEYS\_ONLY
  + NEW\_AND\_OLD\_IMAGES
  + NEW\_IMAGE
  + **OLD\_IMAGE**

**Correct**

**DynamoDB Streams** provides a time-ordered sequence of item level changes in any DynamoDB table. The changes are de-duplicated and stored for 24 hours. Applications can access this log and view the data items as they appeared before and after they were modified, in near real time.



Amazon DynamoDB is also integrated with AWS Lambda so that you can create *triggers*—pieces of code that automatically respond to events in DynamoDB Streams. With triggers, you can build applications that react to data modifications in DynamoDB tables.

When an item in the table is modified, StreamViewType determines what information are written to the stream for this table. Valid values for StreamViewType are KEYS\_ONLY, NEW\_IMAGE, OLD\_IMAGE, and NEW\_AND\_OLD\_IMAGES.

For the **OLD\_IMAGE** type, the entire item which has the previous value as it appeared before it was modified is written to the stream. Hence, this is the correct answer in this scenario.

**KEYS\_ONLY** is incorrect because it will only write the key attributes of the modified item to the stream. This choice is wrong since the question states that values should be copied as well.

**NEW\_IMAGE** is incorrect because it will configure the stream to write the entire item with its new value as it appears after it was modified. This choice is wrong since the stream should capture the item’s pre-modified values.

**NEW\_AND\_OLD\_IMAGES** is incorrect because although it writes the new values of the item in the stream, it also includes the old one as well. Since this type will send both the new and the old item images of the item to the stream, this option is wrong. Remember that it should **only** send a copy of the item’s previous value to the S3 bucket, and not the new value in the DynamoDB table. The most suitable one to use here is the **OLD\_IMAGE**type.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>

<https://docs.amazonaws.cn/en_us/amazondynamodb/latest/developerguide/Streams.Lambda.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_StreamSpecification.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

**AWS Lambda Integration with Amazon DynamoDB Streams:**

<https://tutorialsdojo.com/aws-lambda-integration-with-amazon-dynamodb-streams/>

**Tutorials Dojo’s AWS Certified Developer Associate Exam Study Guide:**

<https://tutorialsdojo.com/aws-certified-developer-associate/>

1. **22. Question**

A developer has deployed a Lambda function that runs in DEV, UAT, and PROD environments. The function uses different parameters that varies based on the environment it is running in. The parameters are currently hardcoded in the function.

Which action should the developer do to reference the appropriate parameters without modifying the code every time the environment changes?

* + Create a stage variable called ENV and invoke the Lambda function by its alias name.
  + Create individual Lambda Layers for each environment
  + Publish three versions of the Lambda function. Assign the aliases DEV, UAT, and PROD to each version.
  + **Use environment variables to set the parameters per environment.**

**Correct**

**Environment variables for Lambda functions** enable you to dynamically pass settings to your function code and libraries without making changes to your code. Environment variables are key-value pairs that you create and modify as part of your function configuration using either the AWS Lambda Console, the AWS Lambda CLI, or the AWS Lambda SDK. AWS Lambda then makes these key-value pairs available to your Lambda function code using standard APIs supported by the language, like process.env for Node.js functions.

You can use environment variables to help libraries know what directory to install files in, where to store outputs, store connection and logging settings, and more. By separating these settings from the application logic, you don’t need to update your function code when changing the function behavior based on different settings.

Hence, the correct answer is: **Use environment variables to set the parameters per environment.**

The option that says: **Create a stage variable called ENV and invoke the Lambda function by its alias name** is incorrect because the stage variable is a feature of API Gateway, not AWS Lambda.

The option that says: **Create individual Lambda Layers for each environment** is incorrect because this feature is only used to pull in additional code and content in the form of layers. A layer is a ZIP archive that contains libraries, a custom runtime, or other dependencies.

The option that says: **Publish three versions of the Lambda function. Assign the aliases DEV, UAT, and PROD to each version** is incorrect because this is just like a pointer to a specific Lambda function version. By using aliases, you can access the Lambda function, which the alias is pointing to, without the caller knowing the specific version the alias is pointing to.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/env_variables.html>

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-configuration.html>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **23. Question**

An API gateway with a Lambda proxy integration takes a long time to complete its processing. There were also occurrences where some requests timed out. You want to monitor the responsiveness of your API calls as well as the underlying Lambda function.

Which of the following CloudWatch metrics should you use to troubleshoot this issue? (Select TWO.)

* + CacheHitCount
  + Count
  + **IntegrationLatency**
  + **Latency**
  + CacheMissCount

**Correct**

You can monitor API execution using CloudWatch, which collects and processes raw data from API Gateway into readable, near-real-time metrics. These statistics are recorded for a period of two weeks so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, API Gateway metric data is automatically sent to CloudWatch in one-minute periods.

The metrics reported by API Gateway provide information that you can analyze in different ways. The list below shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

 – Monitor the **IntegrationLatency** metrics to measure the responsiveness of the backend.

 – Monitor the **Latency** metrics to measure the overall responsiveness of your API calls.

 – Monitor the **CacheHitCount** and **CacheMissCount** metrics to optimize cache capacities to achieve a desired performance.

Hence, the correct metrics that you have to use in this scenario are **Latency** and**IntegrationLatency.**

**Count** is incorrect because this metric simply gets the total number of API requests in a given period.

**CacheMissCount** is incorrect because this metric just gets the number of requests served from the backend in a given period when API caching is enabled. The Sum statistic represents this metric, namely, the total count of the cache misses in the given period.

**CacheHitCount** is incorrect because this fetches the number of requests served from the API cache in a given period. The Sum statistic represents this metric, namely, the total count of the cache hits in the given period.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-metrics-and-dimensions.html>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/monitoring-cloudwatch.html>

**Check out this Amazon API Gateway Cheat Sheet:**

[https://tutorialsdojo.com/amazon-api-gateway](https://tutorialsdojo.com/amazon-api-gateway/)

1. **24. Question**

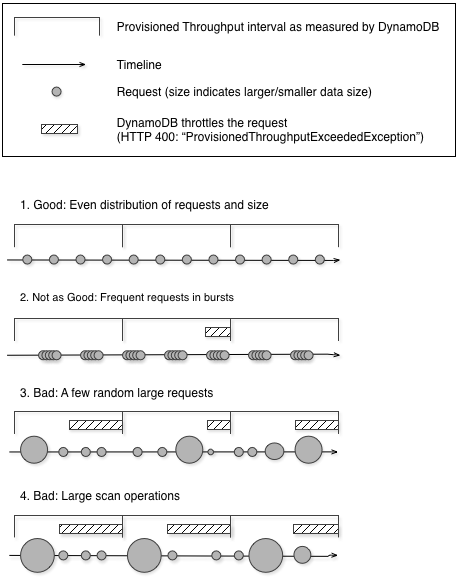
A reporting application is hosted in Elastic Beanstalk and uses DynamoDB as its database. If a user requests data, the application scans the entire table and returns the requested data. In the coming weeks, it is expected that the table will grow due to the surge of new users and requested reports.

Which of the following should be done as a preparation to improve the application’s performance with minimal cost? (Select TWO.)

* + **Reduce page size**
  + Increase page size
  + Use DynamoDB Accelerator (DAX)
  + Increase the Write Compute Unit (WCU) of the table
  + **Use Query operations instead**

**Correct**

In general, Scan operations are less efficient than other operations in DynamoDB. A Scan operation always scans the entire table or secondary index. It then filters out values to provide the result you want, essentially adding the extra step of removing data from the result set.



If possible, you should avoid using a Scan operation on a large table or index with a filter that removes many results. Also, as a table or index grows, the Scan operation slows. The Scan operation examines every item for the requested values and can use up the provisioned throughput for a large table or index in a single operation. For faster response times, design your tables and indexes so that your applications can use Query instead of Scan. For tables, you can also consider using the GetItem and BatchGetItem APIs.

Alternatively, you can refactor your application to use Scan operations in a way that minimizes the impact on your request rate.  Instead of using a large Scan operation, you can use the following techniques to minimize the impact of a scan on a table’s provisioned throughput.

**Reduce page size** – Because a Scan operation reads an entire page (by default, 1 MB), you can reduce the impact of the scan operation by setting a smaller page size. The Scan operation provides a *Limit* parameter that you can use to set the page size for your request. Each Query or Scan request that has a smaller page size uses fewer read operations and creates a “pause” between each request. For example, suppose that each item is 4 KB and you set the page size to 40 items. A Query request would then consume only 20 eventually consistent read operations or 40 strongly consistent read operations. A larger number of smaller Query or Scan operations would allow your other critical requests to succeed without throttling.

**Isolate scan operations** – DynamoDB is designed for easy scalability. As a result, an application can create tables for distinct purposes, possibly even duplicating content across several tables. You want to perform scans on a table that is not taking “mission-critical” traffic. Some applications handle this load by rotating traffic hourly between two tables—one for critical traffic, and one for bookkeeping. Other applications can do this by performing every write on two tables: a “mission-critical” table, and a “shadow” table.

Hence, using **Query operations instead of Scan and reducing the page size**are the correct answers.

**Using DynamoDB Accelerator (DAX)**is incorrect. Although this will improve the scalability and read performance of the application, it adds a significant cost in maintaining your application. Using Query operations and reducing the page size of your query are the more cost-effective solutions in this scenario.

**Increasing page size**is incorrect because it should be the other way around. You have to decrease the page size of the Scan operation using the *Limit* parameter to reduce the impact.

**Increasing the Write Compute Unit (WCU) of the table**is incorrect because the reporting application is heavily used for reading data and not for writing. In addition, increasing the WCU will increase the cost.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-scan.html#bp-query-scan-spikes>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-practices.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

1. **25. Question**

A developer is managing a serverless application orchestrated by AWS Step Functions. One of the Lambda functions sends an API call to a third-party payment service, which takes some time to complete. The Step Functions workflow needs to pause while the service validates the payment. It should only resume after the service sends a notification to a webhook endpoint.

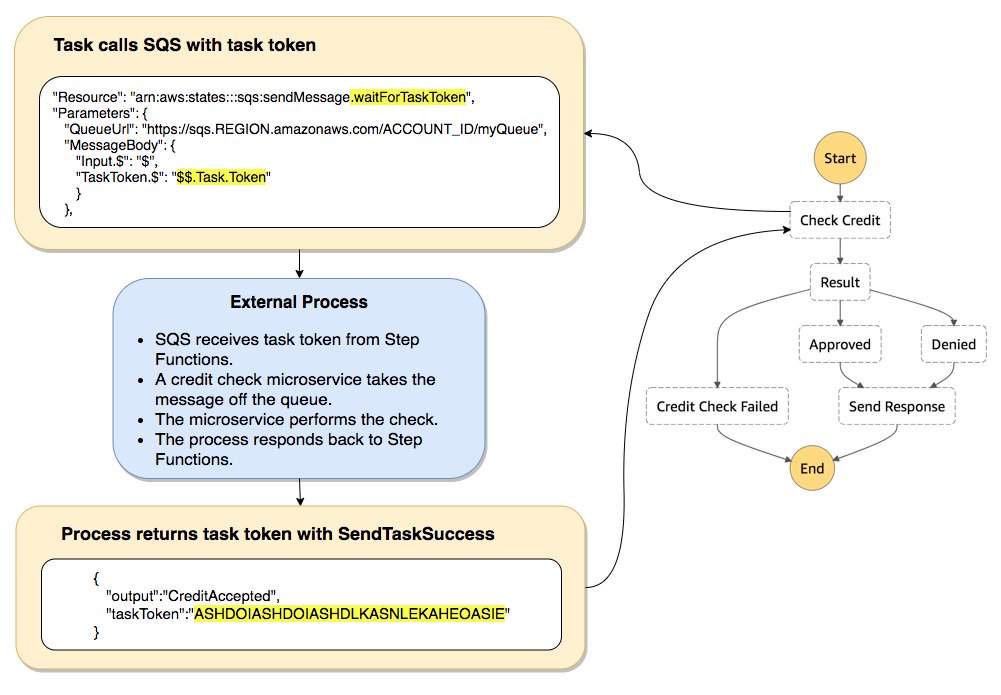
Which combination of actions will fulfill the requirements in the most cost-effective manner? (Select Two)

* + Set the invocation method of the Lambda function task state to asynchronous. Create an AWS SQS queue and configure the webhook handler to send the payment service’s response to the queue. Use a combination of Wait State and Choice State to poll the queue.
  + **Configure the Lambda function task state to use the waitForTaskToken option. Retrieve the task token from the context object of the state machine and include it as part of the Lambda function’s payload body.**
  + **Use a Wait State to pause the execution of the workflow. Configure the webhook handler to invoke the Lambda function synchronously.**
  + **Configure the webhook handler to call the SendTaskSuccess method after a successful notification.**
  + Configure the webhook handler to call the SendTaskHeartbeat method after a successful notification.

**Incorrect**

In AWS Step Functions, the waitForTaskToken option allows a task to be paused until an external system signals its completion. When a task is configured with this option, Step Functions generates a unique token, which can be retrieved from the context object of the state machine. This token, for instance, can be stored in a data store for reference.

*The diagram below depicts how waitForTaskToken is used for an SQS task state.*



An external system, such as a webhook handler can then reference the token and call the SendTaskSuccess or SendTaskFailure method to signal Step Functions to resume the workflow. When the workflow is in a paused state, you’re not billed for the time the workflow is paused, making it a cost-effective method for awaiting external processes or events.

Hence, the correct answers are:

* + **Configure the Lambda function task state to use the waitForTaskToken option. Retrieve the task token from the context object of the state machine and include it as part of the Lambda function’s payload body.**
  + **Configure the webhook handler to call the SendTaskSuccess method after a successful notification.**

The option that says: **Set the invocation method of the Lambda function task state to asynchronous. Create an AWS SQS queue and configure the webhook handler to send the payment service’s response to the queue. Use a combination of Wait State and Choice State to poll the queue**is incorrect. While this solution may work, every iteration involving the Wait State and Choice State incurs a cost as a state transition. If the third-party service takes an unpredictable amount of time, the state machine could go through multiple cycles of waiting and checking the SQS queue, resulting in a higher cost.

The option that says: **Use a Wait State to pause the execution of the workflow. Configure the webhook handler to invoke the Lambda function synchronously** is incorrect. A fixed Wait State is less cost-effective in scenarios where the waiting duration is unpredictable. If the third-party service finishes earlier than the wait duration, you’re paying for unused time. If it takes longer, the workflow might proceed before the task is complete.

The option that says: **Configure the webhook handler to call the SendTaskHeartbeat method after a successful notification**is incorrect because this method is simply used for keeping tasks alive and preventing them from timing out. It also does not signal completion.

**References:**

<https://aws.amazon.com/blogs/compute/building-cost-effective-aws-step-functions-workflows/>

<https://docs.aws.amazon.com/step-functions/latest/dg/callback-task-sample-sqs.html>

<https://docs.aws.amazon.com/step-functions/latest/dg/connect-to-resource.html>

**Check out this AWS Step Functions Cheat Sheet:**

<https://tutorialsdojo.com/aws-step-functions/>

1. **26. Question**

A developer is working on a Lambda function which has an event source mapping to process requests from API Gateway. The function will consistently have 10 requests per second and it will take a maximum of 50 seconds to complete each request.

What should the developer do to prevent the function from throttling?

* + **Do nothing since Lambda will automatically scale to handle the load.**
  + Submit a Service Limit Increase request to AWS to raise your concurrent executions limit.
  + Use Dead Letter Queues (DLQ) to reprocess failed requests.
  + Implement traffic shifting in Lambda using Aliases.

**Correct**

If you create a Lambda function to process events from event sources that aren’t poll-based (for example, Lambda can process every event from other sources, like Amazon S3 or API Gateway), each published event is a unit of work, in parallel, up to your account limits. Therefore, the number of invocations these event sources make influences the concurrency.

You can use this formula to estimate the capacity used by your function:

**concurrent executions = (invocations per second) x (average execution duration in seconds)**

For example, consider a Lambda function that processes Amazon S3 events. Suppose that the Lambda function takes on average three seconds and Amazon S3 publishes 10 events per second. Then, you will have 30 concurrent executions of your Lambda function. See the calculation shown below to visualize the process:

      = **(10 events per second) x (3 seconds average execution duration)**  
=  **30 concurrent executions**

In this scenario, it is expected that the Lambda function takes a maximum of 50 seconds for every execution with 10 requests per second. Using the formula above, the function will have 500 concurrent executions.

**= (10 events per second) x (50 seconds average execution duration)  
=  500 concurrent executions**

AWS Lambda dynamically scales function execution in response to increased traffic, up to your [concurrency limit](https://docs.aws.amazon.com/lambda/latest/dg/limits.html). Under sustained load, your function’s concurrency bursts to an initial level between 500 and 3000 concurrent executions that varies per region. After the initial burst, the function’s capacity increases by an additional 500 concurrent executions each minute until either the load is accommodated, or the total concurrency of all functions in the region hits the limit.

By default, AWS Lambda limits the total concurrent executions across all functions within a given region to 1000. This limit can be raised by requesting for AWS to increase the limit of the concurrent executions of your account.

Since the expected concurrent executions of the Lambda function is well within the default concurrency limit, the best thing to do here is to **do nothing since Lambda will automatically scale to handle the load.**

**Submitting a Service Limit Increase request to AWS to raise your concurrent executions limit**is incorrect because this is totally unnecessary given that the concurrent executions that will be used by your function are within the limits.

**Using Dead Letter Queues (DLQ) to reprocess failed requests**is incorrect because DLQs are primarily used to direct unprocessed events to an Amazon SQS queue or an Amazon SNS topic to analyze the failure.

**Implementing traffic shifting in Lambda using Aliases**is incorrect because this just allows you to point to two different versions of the Lambda function and dictate what percentage of incoming traffic is sent to each version.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/running-lambda-code.html>

<https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html#function-configuration>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **27. Question**

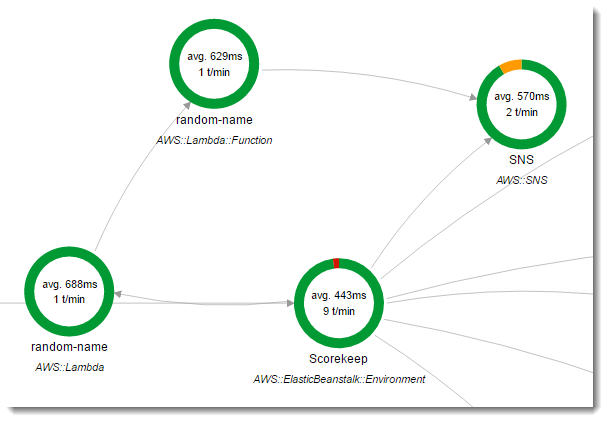
A recently deployed Lambda function has an intermittent issue in processing customer data. You enabled the active tracing option in order to detect, analyze, and optimize performance issues of your function using the X-Ray service.

Which of the following environment variables are used by AWS Lambda to facilitate communication with X-Ray? (Select TWO.)

* + **AWS\_XRAY\_CONTEXT\_MISSING**
  + **AWS\_XRAY\_TRACING\_NAME**
  + AUTO\_INSTRUMENT
  + **\_X\_AMZN\_TRACE\_ID**
  + AWS\_XRAY\_DEBUG\_MODE

**Incorrect**

AWS X-Ray is an AWS service that allows you to detect, analyze, and optimize performance issues with your AWS Lambda applications. X-Ray collects metadata from the Lambda service and any upstream or downstream services that make up your application. X-Ray uses this metadata to generate a detailed service graph that illustrates performance bottlenecks, latency spikes, and other issues that impact the performance of your Lambda application.



AWS Lambda uses environment variables to facilitate communication with the X-Ray daemon and configure the X-Ray SDK.

**\_X\_AMZN\_TRACE\_ID:** Contains the tracing header, which includes the sampling decision, trace ID, and parent segment ID. If Lambda receives a tracing header when your function is invoked, that header will be used to populate the \_X\_AMZN\_TRACE\_ID environment variable. If a tracing header was not received, Lambda will generate one for you.

**AWS\_XRAY\_CONTEXT\_MISSING:** The X-Ray SDK uses this variable to determine its behavior in the event that your function tries to record X-Ray data, but a tracing header is not available. Lambda sets this value to LOG\_ERROR by default.

**AWS\_XRAY\_DAEMON\_ADDRESS:** This environment variable exposes the X-Ray daemon’s address in the following format: *IP\_ADDRESS***:***PORT*. You can use the X-Ray daemon’s address to send trace data to the X-Ray daemon directly without using the X-Ray SDK.

Therefore, the correct answers for this scenario are the **\_X\_AMZN\_TRACE\_ID and AWS\_XRAY\_CONTEXT\_MISSING** environment variables.

**AWS\_XRAY\_TRACING\_NAME** is incorrect because this is primarily used in X-Ray SDK where you can set a service name that the SDK uses for segments.

**AWS\_XRAY\_DEBUG\_MODE** is incorrect because this is used to configure the SDK to output logs to the console without using a logging library.

**AUTO\_INSTRUMENT** is incorrect because this is primarily used in X-Ray SDK for Django Framework only. This allows the recording of subsegments for built-in database and template rendering operations.

**References**:

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html#viewing-lambda-xray-results>

<https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-nodejs-configuration.html>

<https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-python-configuration.html>

**Check out this AWS X-Ray Cheat Sheet:**

<https://tutorialsdojo.com/aws-x-ray/>

**Instrumenting your Application with AWS X-Ray:**

<https://tutorialsdojo.com/instrumenting-your-application-with-aws-x-ray/>

1. **28. Question**

A DynamoDB table has several top-level attributes such as id, course\_id, course\_title, price, rating and many others. The database queries of your application returns all of the item attributes by default but you only want to fetch specific attributes such as the course\_id and price per request.

As the developer, how can you refactor your application to accomplish this requirement?

* + Use condition expressions
  + **Use projection expression**
  + Use filter expressions
  + Use expression attribute names

**Correct**

To read data from a table, you use operations such as GetItem, Query, or Scan. DynamoDB returns all of the item attributes by default. To get just some, rather than all of the attributes, use a projection expression.

A *projection expression* is a string that identifies the attributes you want. To retrieve a single attribute, specify its name. For multiple attributes, the names must be comma-separated.

The following AWS CLI example shows how to use a projection expression with a GetItemoperation. This projection expression retrieves a top-level scalar attribute (Description), the first element in a list (RelatedItems[0]), and a list nested within a map (ProductReviews.FiveStar).

aws dynamodb get-item \

--table-name ProductCatalog \

--key '{"Id":{"N":"1"}}' \

**--projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"**

You can use any attribute name in a projection expression, provided that the first character is a-z or A-Z and the second character (if present) is a-z, A-Z, or 0-9. If an attribute name does not meet this requirement, you will need to define an expression attribute name as a placeholder.

Therefore, **using projection expression**is the correct answer in this scenario.

**Using condition expressions** is incorrect because this is primarily used to determine which items should be modified for data manipulation operations such as PutItem, UpdateItem, and DeleteItem calls.

**Using expression attribute names** is incorrect because this is a placeholder that you use in a projection expression as an alternative to an actual attribute name. An expression attribute name must begin with a #, and be followed by one or more alphanumeric characters.

**Using filter expressions**is incorrect because it simply determines which **items**(and not the **attributes**) within the Query results should be returned to you. All of the other results are discarded. Take note that the scenario says that you have to fetch specific attributes and not specific items.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.ProjectionExpressions.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.html>

1. **29. Question**

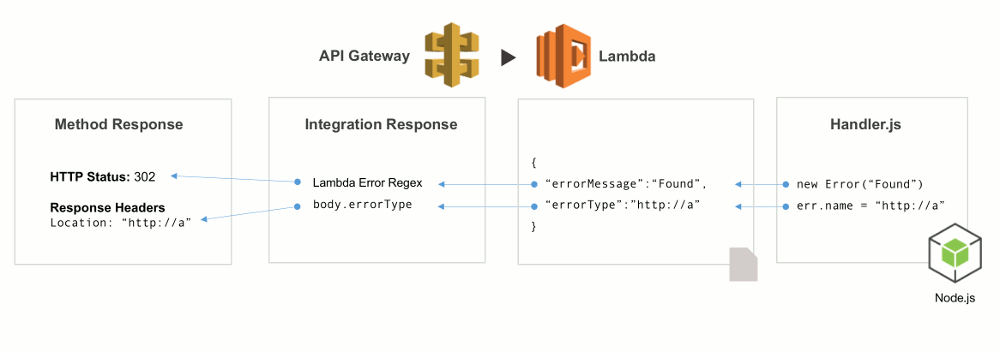
A developer configured an Amazon API Gateway proxy integration named MyAPI to work with a Lambda function. However, when the API is being called, the developer receives a 502 Bad Gateway error. She tried invoking the underlying function, but it properly returned the result in XML format.

What is the MOST likely root cause of this issue?

* + There has been an occasional out-of-order invocation due to heavy loads.
  + The API name of the Amazon API Gateway proxy is invalid.
  + **There is an incompatible output returned from a Lambda proxy integration backend.**
  + The endpoint request timed-out.

**Correct**

Amazon API Gateway Lambda proxy integration is a simple, powerful, and nimble mechanism to build an API with a setup of a single API method. The Lambda proxy integration allows the client to call a single Lambda function in the backend. The function accesses many resources or features of other AWS services, including calling other Lambda functions



In Lambda proxy integration, when a client submits an API request, API Gateway passes the raw request as-is to the integrated Lambda function, except that the order of the request parameters is not preserved. This [request data](https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format) includes the request headers, query string parameters, URL path variables, payload, and API configuration data. The configuration data can include current deployment stage name, stage variables, user identity, or authorization context (if any). The backend Lambda function parses the incoming request data to determine the response that it returns.

For API Gateway to pass the Lambda output as the API response to the client, the Lambda function must return the result in the following JSON format:

{

"isBase64Encoded": true|false,

"statusCode": httpStatusCode,

"headers": { "headerName": "headerValue", ... },

"body": "..."

}

Since the Lambda function returns the result in XML format, it will cause the 502 errors in the API Gateway. Hence, the correct answer is that **there is an incompatible output returned from a Lambda proxy integration backend.**

The option that says: **The API name of the Amazon API Gateway proxy** is invalid is incorrect because there is nothing wrong with its *MyAPI* name.

The option that says: **There has been an occasional out-of-order invocation due to heavy loads** is incorrect. Although this is a valid cause of a 502 error, the issue is most likely caused by the Lambda function’s XML response instead of JSON.

The option that says: **The endpoint request timed-out** is incorrect because this will likely result in 504 errors and not 502’s.

**References:**

<https://aws.amazon.com/premiumsupport/knowledge-center/malformed-502-api-gateway/>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-output-format>

<https://docs.aws.amazon.com/apigateway/api-reference/handling-errors/>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **30. Question**

A Lambda function downloads the same 250 MB file between invocations and stores it in memory for processing. This leads to frequent timeouts and negatively impacts the performance of the serverless application.

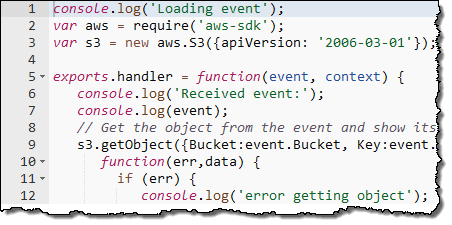
Which change should be made to resolve the issue most effectively?

* + **Store the file in the /tmp directory of the execution context and reuse it on succeeding invocations.**
  + Increase the memory allocation of the function.
  + Increase the ephemeral storage size of the function
  + Increase the timeout of the function.

**Correct**

When AWS Lambda executes your Lambda function, it provisions and manages the resources needed to run your Lambda function. When you create a Lambda function, you specify configuration information, such as the amount of memory and maximum execution time that you want to allow for your Lambda function. When a Lambda function is invoked, AWS Lambda launches an execution context based on the configuration settings you provide. The execution context is a temporary runtime environment that initializes any external dependencies of your Lambda function code, such as database connections or HTTP endpoints. This affords subsequent invocations better performance because there is no need to “cold-start” or initialize those external dependencies, as explained below.

It takes time to set up an execution context and do the necessary “bootstrapping”, which adds some latency each time the Lambda function is invoked. You typically see this latency when a Lambda function is invoked for the first time or after it has been updated because AWS Lambda tries to reuse the execution context for subsequent invocations of the Lambda function.



After a Lambda function is executed, AWS Lambda maintains the execution context for some time in anticipation of another Lambda function invocation. In effect, the service freezes the execution context after a Lambda function completes and thaws the context for reuse if AWS Lambda chooses to reuse the context when the Lambda function is invoked again.

Each execution context provides **512 MB – 10,240 MB** of additional disk space in the /tmp directory. The directory content remains when the execution context is frozen, providing a transient cache that can be used for multiple invocations. You can add extra code to check if the cache has the data that you stored.

Hence, the correct answer in this scenario is to **Store the file in the /tmp directory of the execution context and reuse it on succeeding invocations.**

The option that says: **Increase the memory allocation of the function**is incorrect. The actual processing time may be reduced by allocating more memory, but there would still be a lot of time wasted in downloading the 250 MB file every time the function is invoked.

The option that says: **Increase the timeout of the function** is incorrect because this doesn’t solve the root cause of the problem. You may configure your function with a maximum timeout of 15 minutes. However, the fact still remains that the function repeatedly downloads the file at every invocation.

The option that says: **Increase the ephemeral storage size of the function**is incorrect. This won’t have any effect at all on solving the issue. The change must be made at the code level. Instead of storing the file directly on memory, storing it in the /tmp directory provides a more cost-effective and scalable solution, as the function can read and write the file from disk as needed. The /tmp directory also persists between invocations of the function, allowing it to access the file more quickly between invocations.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/running-lambda-code.html>

<https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html#function-configuration>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **31. Question**

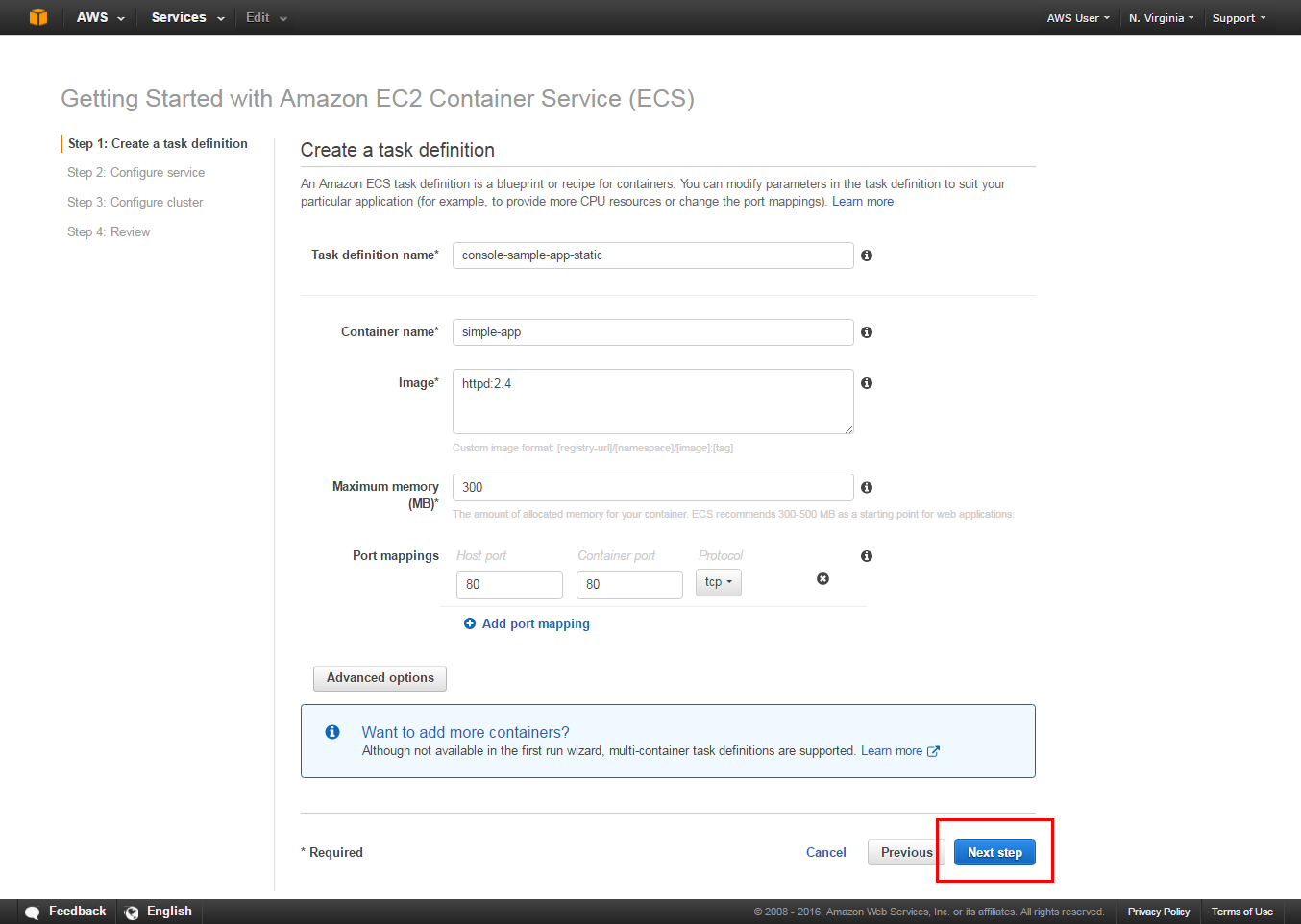
A developer is building a new Docker application using ECS. She needs to allow containers to access ports on the host container instance to send or receive traffic using port mapping.

Which component of ECS should the developer configure to properly implement this task?

* + Service scheduler
  + Container instance
  + **Task definition**
  + Container Agent

**Correct**

Port mappings allow containers to access ports on the host container instance to send or receive traffic. Port mappings are specified as part of the container definition which can be configured in the task definition.



For task definitions that use the awsvpc network mode, you should only specify the containerPort. The hostPort can be left blank or it must be the same value as the containerPort.

Port mappings on Windows use the NetNAT gateway address rather than localhost. There is no loopback for port mappings on Windows, so you cannot access a container’s mapped port from the host itself.

Hence, the correct component that the developer should configure is **Task Definition.**

**Service scheduler** is incorrect because this only provides you the ability to run tasks manually (for batch jobs or single run tasks), with Amazon ECS placing tasks on your cluster for you. The service scheduler is ideally suited for long running stateless services and applications but not to configure port mappings.

**Container instance** is incorrect because this is just an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into a cluster. When you run tasks with Amazon ECS, your tasks using the EC2 launch type are placed on your active container instances. However, you can’t manually configure the port mappings directly on your container instances but through task definitions.

**Container Agent**is incorrect because this only allows container instances to connect to your cluster. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs, but you can also install it on any Amazon EC2 instance that supports the Amazon ECS specification. Same as the other incorrect options, you can’t configure port mappings with this component.

**References:**

<https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_PortMapping.html>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html#container_definition_portmappings>

**Check out this Amazon ECS Cheat Sheet:**

<https://tutorialsdojo.com/amazon-elastic-container-service-amazon-ecs/>

1. **32. Question**

A developer has recently completed a new version of a serverless application that is ready to be deployed using AWS SAM. There is a requirement that the traffic should shift from the previous Lambda function to the new version in the shortest time possible, but you still don’t want to shift traffic all-at-once immediately.

Which deployment configuration is the MOST suitable one to use in this scenario?

* + CodeDeployDefault.LambdaLinear10PercentEvery2Minutes
  + CodeDeployDefault.HalfAtATime
  + CodeDeployDefault.LambdaLinear10PercentEvery1Minute
  + **CodeDeployDefault.LambdaCanary10Percent5Minutes**

**Correct**

If you use AWS SAM to create your serverless application, it comes built-in with CodeDeploy to help ensure safe Lambda deployments. There are various deployment preference types that you can choose from.

For example:

If you choose **Canary10Percent10Minutes**then **10** percent of your customer traffic is immediately shifted to your new version. After **10** minutes, all traffic is shifted to the new version.

However, if your pre-hook/post-hook tests fail, or if a CloudWatch alarm is triggered, CodeDeploy rolls back your deployment. The following table outlines other traffic-shifting options that are available:

* + **Canary**: **Traffic is shifted in two increments**. You can choose from predefined canary options. The options specify the percentage of traffic that’s shifted to your updated Lambda function version in the first increment, and the interval, in minutes, before the remaining traffic is shifted in the second increment.
  + **Linear**: **Traffic is shifted in equal increments with an equal number of minutes between each increment.** You can choose from predefined linear options that specify the percentage of traffic that’s shifted in each increment and the number of minutes between each increment.
  + **All-at-once**: All traffic is shifted from the original Lambda function to the updated Lambda function version at once.

Hence, the **CodeDeployDefault.LambdaCanary10Percent5Minutes** option is correct because **10** percent of your customer traffic is **immediately** shifted to your new version. After **5** minutes, all traffic is shifted to the new version. This means that the entire deployment time will only take **5** minutes

**CodeDeployDefault.HalfAtATime**is incorrect because this is only applicable for EC2/On-premises compute platform and not for Lambda.

**CodeDeployDefault.LambdaLinear10PercentEvery1Minute** is incorrect because it will add 10 percent of the traffic linearly to the new version every minute. Hence, all traffic will be shifted to the new version only after **10** minutes

**CodeDeployDefault.LambdaLinear10PercentEvery2Minutes** is incorrect because it will add 10 percent of the traffic linearly to the new version every 2 minutes. Hence, all traffic will be shifted to the new version only after **20** minutes.

**References:**

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html>

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-deploying.html>

1. **33. Question**

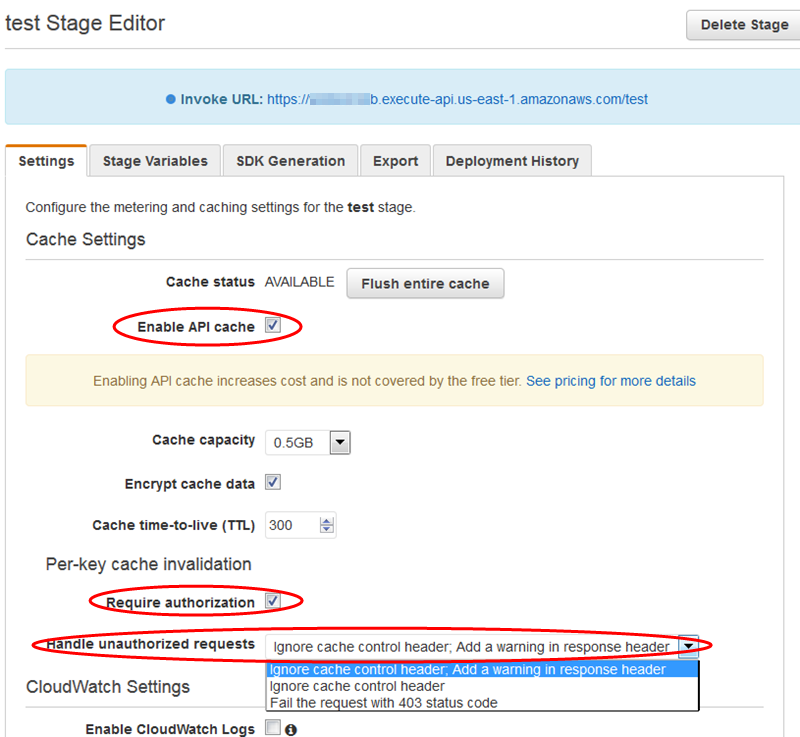
Your serverless AWS Lambda functions are integrated with Amazon API gateway using Lambda proxy integration. The API caching feature is enabled in the API Gateway with a TTL value of 300 seconds. A client would like to fetch the latest data from your endpoints every time a request is sent and invalidate the existing cache.

What should the client do in order to get the latest data?

* + Have the client send a request with the Cached: false header.
  + Override API caching by allowing the client to send requests to the endpoint directly.
  + Modify cache TTL value to a shorter period.
  + **Have the client send a request with the Cache-Control: max-age=0 header.**

**Correct**

A client of your API can invalidate an existing cache entry and reload it from the integration endpoint for individual requests. The client must send a request that contains the **Cache-Control: max-age=0** header.



The client receives the response directly from the integration endpoint instead of the cache, provided that the client is authorized to do so. This replaces the existing cache entry with the new response, which is fetched from the integration endpoint.

**Modifying the TTL value for the cached data to a lower value** is incorrect because there is still no guarantee that the client will submit a request after the cache has expired. Also, you will not be fully utilizing the purpose of API caching since new data will be fetched from the endpoint more often. The best solution for this scenario is to use the Cache-Control header instead.

**Allowing the client to access the endpoint directly** is incorrect because the purpose of placing API Gateway in-front of your endpoints is to not expose your endpoints to the public and risk security issues. It also provides you the additional benefits of not burdening your endpoints with a massive number of requests and allowing developer-friendly data exchanges through APIs.

**Having the client send a request with the Cached: false header** is incorrect because this is a custom header. The correct way is to configure the **Cache-Control: max-age=0** header instead.

**Reference:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html#override-api-gateway-stage-cache-for-method-cache>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **34. Question**

A website hosted in AWS has a custom CloudWatch metric to track all HTTP server errors in the site every minute, which occurs intermittently. An existing CloudWatch Alarm has already been configured for this metric but you would like to re-configure this to properly monitor the application. The alarm should only be triggered when all three data points in the most recent three consecutive periods are above the threshold.

Which of the following options is the MOST appropriate way to monitor the website based on the given threshold?

* + Use high-resolution metrics.
  + Use metric math in CloudWatch to properly compute the threshold.
  + **Set both the Evaluation Period and Datapoints to Alarm to 3.**
  + **Set both the Period and Datapoints to Alarm to 3.**

**Incorrect**

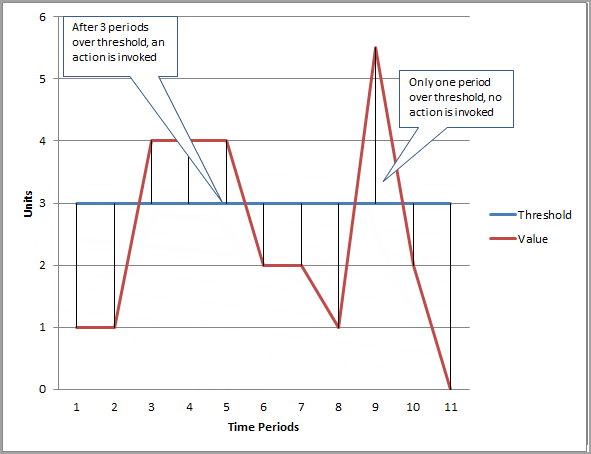
When you create an alarm, you specify three settings to enable CloudWatch to evaluate when to change the alarm state:

**– Period** is the length of time to evaluate the metric or expression to create each individual data point for an alarm. It is expressed in seconds. If you choose one minute as the period, there is one datapoint every minute.

**– Evaluation Period** is the number of the most recent periods, or data points, to evaluate when determining alarm state.

**– Datapoints to Alarm** is the number of data points within the evaluation period that must be breaching to cause the alarm to go to the ALARM state. The breaching data points do not have to be consecutive, they just must all be within the last number of data points equal to **Evaluation Period**.

In the following figure, the alarm threshold is set to three units. The alarm is configured to go to the ALARM state and both **Evaluation Period** and **Datapoints to Alarm** are 3. That is, when all three datapoints in the most recent three consecutive periods are above the threshold, the alarm goes to the ALARM state. In the figure, this happens in the third through fifth time periods. At period six, the value dips below the threshold, so one of the periods being evaluated is not breaching, and the alarm state changes to OK. During the ninth time period, the threshold is breached again, but for only one period. Consequently, the alarm state remains OK.



Hence, the option that says:**Set both the Evaluation Period and Datapoints to Alarm to 3** is the correct answer.

The option that says: **Use high-resolution metrics** is incorrect because the scenario says that it only needs to monitor the HTTP server errors every minute, and not its sub-minute activity. If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds. Hence, this option is irrelevant in this scenario.

The option that says:**Set both the Period and Datapoints to Alarm to 3** is incorrect because you should set the ***Evaluation Period*** and not the ***Period*** setting.

The option that says: **Use metric math in CloudWatch to properly compute the threshold** is incorrect because the Metric Math feature is only applicable for scenarios where you need to query multiple CloudWatch metrics or if you want to use math expressions to create new time series based on selected metrics.

**References:**

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html>

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ConsoleAlarms.html>

**Check out this Amazon CloudWatch Cheat Sheet:**

<https://tutorialsdojo.com/amazon-cloudwatch/>

1. **35. Question**

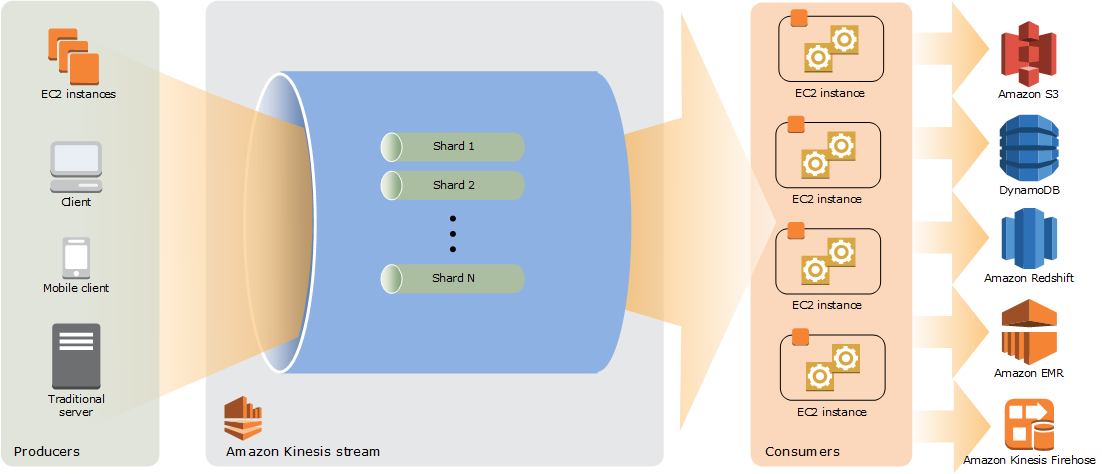
Your application is processing one Kinesis data stream which has four shards, and each instance has one KCL worker. To scale up processing in your application, you reshard your stream to increase the number of open shards to six.

What is the MAXIMUM number of EC2 instances that you should launch to achieve optimum performance?

* + 3
  + 5
  + **6**
  + 12

**Correct**

Resharding enables you to increase or decrease the number of shards in a stream in order to adapt to changes in the rate of data flowing through the stream. The Kinesis Client Library (KCL) ensures that for every shard there is a record processor running and processing that shard. It also tracks the shards in the stream using an Amazon DynamoDB table.



Typically, when you use the KCL, you should ensure that the number of instances does not exceed the number of shards (except for failure standby purposes). Each shard is processed by exactly one KCL worker and has exactly one corresponding record processor, so you never need multiple instances to process one shard. However, one worker can process any number of shards, so it’s fine if the number of shards exceeds the number of instances.

To scale up processing in your application, you should test a combination of these approaches:

 – Increasing the instance size (because all record processors run in parallel within a process)

 – Increasing the number of instances up to the maximum number of open shards (because shards can be processed independently)

 – Increasing the number of shards (which increases the level of parallelism)

Thus, the maximum number of instances you can launch is **6**, to match the number of open shards in a ratio of 1:1.

Although you can launch **3** instances in which each instance handles 2 shards, this is not the maximum number of instances you can deploy for your application. Hence, this option is incorrect. Take note that the maximum number of your instances is not half the number of open shards.

Just like the above option, you can also launch **5** instances in which each instance handles 3 shards. However, this is not the maximum number of instances you can launch. Keep in mind that the maximum number of your instances can be equal to the number of open shards of the Kinesis stream. Therefore, this option is also incorrect.

Launching **12** instances is incorrect because you should ensure that the number of instances does not exceed the number of open shards. The maximum number of instances that you should deploy is 6.

**References:**

<https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html>

<https://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-scaling.html>

**Check out this Amazon Kinesis Cheat Sheet:**

<https://tutorialsdojo.com/amazon-kinesis/>

**Kinesis Scaling, Resharding and Parallel Processing:**

<https://tutorialsdojo.com/kinesis-scaling-resharding-and-parallel-processing/>

1. **36. Question**

A mobile game has a serverless backend consisting of an API Gateway backed by Lambda functions and a DynamoDB table in provisioned capacity mode, where player data is stored. While the game has maintained a consistent level of traffic, recent growth in the player base has caused response times to slow down. To improve performance, the developer wants to reduce the number of database queries for data that rarely change.

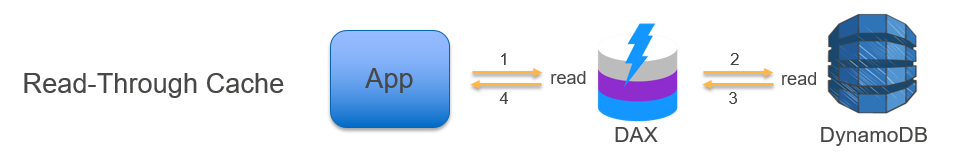
What approach can the developer take to achieve this goal cost-effectively and with less development overhead?

* + Create an Amazon MemoryDB for Redis database in front of the DynamoDB table to cache data.
  + **Set up an Amazon DynamoDB Accelerator (DAX) caching layer in front of the DynamoDB table.**
  + Use DynamoDB Session Handler to handle the saving and retrieval of player data.
  + Switch the DynamoDB table’s capacity mode to On-demand.

**Correct**

Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to a 10x performance improvement – from milliseconds to **microseconds** – even at millions of requests per second. DAX does all the heavy lifting required to add in-memory acceleration to your DynamoDB tables without requiring developers to manage cache invalidation, data population, or cluster management.

This will enable you to focus on building great applications for your customers without worrying about performance at scale. You do not need to modify application logic since DAX is compatible with existing DynamoDB API calls. You can enable DAX with just a few clicks in the AWS Management Console or using the AWS SDK. Just as with DynamoDB, you only pay for the capacity you provision.



One of the key benefits of using DAX is that it works transparently with existing DynamoDB API calls, and the application code does not need to be modified to take advantage of the caching layer. This means that the developer can improve the performance of their DynamoDB applications without having to modify the code, making it easier and more cost-effective to improve application performance.

Hence, the correct answer is: **Set up an Amazon DynamoDB Accelerator (DAX) caching layer in front of the DynamoDB table.**

The option that says: **Create an Amazon MemoryDB for Redis database in front of the DynamoDB table to cache data** is incorrect. While this might work, it is not the most cost-effective approach because it involves setting up an entirely new database in addition to the existing DynamoDB table. Additionally, this method requires modifying the application code to use the Redis cache for data queries, which can be time-consuming.

The option that says: **Use DynamoDB Session Handler to handle the saving and retrieval of player data** is incorrect because this is just a custom session handler for PHP and does not provide any caching functionality. It is not a suitable approach to reduce the number of database queries for data that rarely change.

The option that says: **Switch the DynamoDB table’s capacity mode to On-demand** is incorrect. While it may help improve performance by automatically scaling the read capacity in response to traffic changes, it does not address the main objective of reducing the number of database queries for data that rarely change.

**References:**

<https://aws.amazon.com/dynamodb/dax>

<https://aws.amazon.com/caching/aws-caching/>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

1. **37. Question**

A Lambda function has been integrated with DynamoDB Streams as its event source. There has been a new version of the function that needs to be deployed using CodeDeploy where the traffic must be shifted in two increments. It should shift 10 percent of the incoming traffic to the new version in the first increment and then the remaining 90 percent should be deployed five minutes later.

Which of the following deployment configurations is the MOST suitable to satisfy this requirement?

* + Linear
  + **Canary**
  + Rolling with additional batch
  + All-at-once

**Correct**

**CodeDeploy** is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services. CodeDeploy can deploy application content that runs on a server and is stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories. CodeDeploy can also deploy a serverless Lambda function. You do not need to make changes to your existing code before you can use CodeDeploy.

CodeDeploy supports the following deployment configurations:

**-In-place (for EC2/On-premises)**– the application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated.

**-Canary (for Lambda/ECS) –** traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated Lambda function or ECS task set in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.

**-Linear (for Lambda/ECS) –** traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.

**-All-at-once (for Lambda/ECS) –** all traffic is shifted from the original Lambda function or ECS task set to the updated function or task set all at once.



In a **Canary** deployment configuration, the traffic is shifted in **two** increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated Lambda function version in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment. Hence, this is the correct answer which will satisfy the requirement for the given scenario.

**Linear** is incorrect because this will cause the traffic to be shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.

**All-at-once**is incorrect because with this deployment configuration, the traffic is shifted from the original Lambda function to the updated Lambda function version all at once.

**Rolling with additional batch** is incorrect because this is only applicable in Elastic Beanstalk and not for Lambda.

**References:**

<https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-configurations.html>

<https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html>

**Check out this AWS CodeDeploy Cheat Sheet:**

<https://tutorialsdojo.com/aws-codedeploy/>

1. **38. Question**

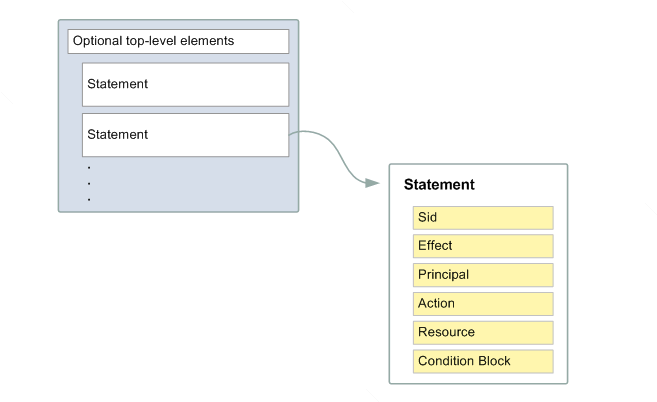
A company has created a private S3 bucket named tdojo. The Developer IAM role must be granted read access to all objects within this bucket. However, objects stored under the qa folder should be restricted to the QA IAM role only.

Which S3 bucket policy will effectively implement the principle of least privilege access while satisfying the given requirements?

* + {  
      “Version”: “2012-10-17”,  
      “Statement”: [  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/Developer”,  
              “arn:aws:iam::123456789123:role/QA”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”  
        ],  
        “Resource”: [  
           “arn:aws:s3:::tdojo/\*”,  
           “arn:aws:s3:::tdojo/qa/\*”  
        ]  
        }  
      ]  
    }
  + {  
      “Version”: “2012-10-17”,  
      “Statement”: [  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/Developer”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”,  
        “s3:PutObject”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/\*”  
        },  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/QA”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/qa/\*”  
        }  
      ]  
    }
  + {  
      “Version”: “2012-10-17”,  
      “Statement”: [  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/Developer”  
          ]  
        },  
        “Action”: [  
        “s3:\*”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/\*”  
        },  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/QA”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/qa/\*”  
        }  
      ]  
    }
  + **{  
      “Version”: “2012-10-17”,  
      “Statement”: [  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/Developer”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/\*”  
        },  
        {  
        “Effect”: “Allow”,  
        “Principal”: {  
          “AWS”: [  
              “arn:aws:iam::123456789123:role/QA”  
          ]  
        },  
        “Action”: [  
        “s3:GetObject”  
        ],  
        “Resource”: “arn:aws:s3:::tdojo/qa/\*”  
        }  
      ]  
    }**

**Correct**

An S3 bucket policy is a JSON document that defines access controls for an Amazon S3 bucket. It allows you to grant or deny access to different AWS identities (such as IAM users and roles or federated users) for different S3 bucket actions (such as GetObject, PutObject, and ListBucket) on specific S3 objects or prefixes within a bucket. It can be used to control who can access your S3 data and how they can access it.



An S3 bucket policy statement is composed of several elements, and the following are required to create a valid policy:

– **Effect**: The effect can be **Allow** or **Deny**.

– **Action**: The specific API action for which you are granting or denying permission.

– **Principal**: The user, account, service, or other entity that is allowed or denied access to the bucket or objects within the bucket.

– **Resource**: The resource that’s affected by the action. You specify a resource using an Amazon Resource Name (ARN).

In the scenario, you can create separate statements for the Developer and QA roles. This allows for granular control over access to the bucket and its contents. Each statement can have its own set of conditions, allowing for different permissions to be granted to different identities or groups under different circumstances. Additionally, separating statements can make it easier to read, understand and manage the policy as it becomes more complex.

Hence, the correct answer is:

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/Developer"**

**]**

**},**

**"Action": [**

**"s3:GetObject"**

**],**

**"Resource": "arn:aws:s3:::tdojo/\*"**

**},**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/QA"**

**]**

**},**

**"Action": [**

**"s3:GetObject"**

**],**

**"Resource": "arn:aws:s3:::tdojo/qa/\*"**

**}**

**]**

**}**

The policy below is not compliant with the principle of least privilege, as it uses a wildcard in the Action element. The scenario mentioned that the Developer IAM role must be given permission to read all objects in the bucket. Therefore, you have to explicitly mention the specific API to allow that operation, which is GetObject.

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/Developer"**

**]**

**},**

**"Action": [**

**"s3:\*"**

**],**

**"Resource": "arn:aws:s3:::tdojo/\*"**

**},**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/QA"**

**]**

**},**

**"Action": [**

**"s3:GetObject"**

**],**

**"Resource": "arn:aws:s3:::tdojo/qa/\*"**

**}**

**]**

**}**

The policy below does not follow the principle of least privilege since it grants excessive permissions to the Developer IAM role. Note that the Developer IAM role only needs read access.

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/Developer"**

**]**

**},**

**"Action": [**

**"s3:GetObject",**

**"s3:PutObject"**

**],**

**"Resource": "arn:aws:s3:::tdojo/\*"**

**},**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/QA"**

**]**

**},**

**"Action": [**

**"s3:GetObject"**

**],**

**"Resource": "arn:aws:s3:::tdojo/qa/\*"**

**}**

**]**

**}**

The IAM Policy shown below is not right as it grants the QA role access to read all objects within the S3 bucket, which is beyond its necessary permissions:

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Principal": {**

**"AWS": [**

**"arn:aws:iam::123456789123:role/Developer",**

**"arn:aws:iam::123456789123:role/QA"**

**]**

**},**

**"Action": [**

**"s3:GetObject"**

**],**

**"Resource": [**

**"arn:aws:s3:::tdojo/\*",**

**"arn:aws:s3:::tdojo/qa/\*"**

**]**

**}**

**]**

**}**

**References:**

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-iam-policies.html>

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html>

**Check out this Amazon S3 Cheat Sheet:**

<https://tutorialsdojo.com/amazon-s3/>

1. **39. Question**

A developer is building a photo-sharing application that automatically enhances images uploaded by users to Amazon S3. When a user uploads an image, its S3 path is sent to an image-processing application hosted on AWS Lambda. The Lambda function applies the selected filter to the image and stores it back to S3.

If the upload is successful, the application will return a prompt telling the user that the request has been accepted. The entire processing typically takes an average of 5 minutes to complete, which causes the application to become unresponsive.

Which of the following is the MOST suitable and cost-effective option which will prevent the application from being unresponsive?

* + Use a combination of Lambda and Step Functions to orchestrate service components and asynchronously process the requests.
  + Configure the application to asynchronously process the requests and use the default invocation type of the Lambda function.
  + Use AWS Serverless Application Model (AWS SAM) to allow asynchronous requests to your Lambda function.
  + **Configure the application to asynchronously process the requests and change the invocation type of the Lambda function to Event.**

**Correct**

**AWS Lambda** supports synchronous and asynchronous invocation of a Lambda function. You can control the invocation type only when you invoke a Lambda function (referred to as *on-demand invocation*). The following examples illustrate on-demand invocations:

– Your custom application invokes a Lambda function.

– You manually invoke a Lambda function (for example, using the AWS CLI) for testing purposes.

In both cases, you invoke your Lambda function using the [Invoke](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html) operation, and you can specify the invocation type as synchronous or asynchronous.

When you use AWS services as a trigger, the invocation type is predetermined for each service. You have no control over the invocation type that these event sources use when they invoke your Lambda function.

In the Invoke API, you have 3 options to choose from for the **InvocationType:**

**RequestResponse** (default) – Invoke the function synchronously. Keep the connection open until the function returns a response or times out. The API response includes the function response and additional data.

**Event** – Invoke the function asynchronously. Send events that fail multiple times to the function’s dead-letter queue (if it’s configured). The API response only includes a status code.

**DryRun** – Validate parameter values and verify that the user or role has permission to invoke the function.

By configuring the application to asynchronously process requests by changing the invocation type of the Lambda function to “Event,” the function can run in the background without blocking the main application. When the processing is complete, Lambda can store it back to S3 and trigger another event, such as a notification to the user that the image is ready.

Hence, the correct answer is to **configure the application to asynchronously process the requests and change the invocation type of the Lambda function to Event.**

**Configuring the application to asynchronously process the requests and use the default invocation type of the Lambda function** is incorrect because this will invoke your Lambda function synchronously. The default invocation type is **RequestResponse** which invokes the function synchronously and keeps the connection open until the function returns a response or times out.

**Using AWS Serverless Application Model (AWS SAM) to allow asynchronous requests to your Lambda function** is incorrect because AWS SAM just is an open-source framework that you can use to build serverless applications on AWS.

**Using a combination of Lambda and Step Functions to orchestrate service components and asynchronously process the requests** is incorrect because the AWS Step Functions service just lets you coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. Although this can be a valid solution, it is not cost-effective since the application does not have a lot of components to orchestrate. Lambda functions can effectively meet the requirements in this scenario without using Step Functions by processing the requests asynchronously.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/invocation-options.html>

<https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **40. Question**

A developer is working with an AWS Serverless Application Model (AWS SAM) application composed of several AWS Lambda functions. The developer runs the application locally on his laptop using sam local commands. While testing, one of the functions returns Access denied errors. Upon investigation, the developer discovered that the Lambda function is using the AWS SDK to make API calls within a sandbox AWS account.

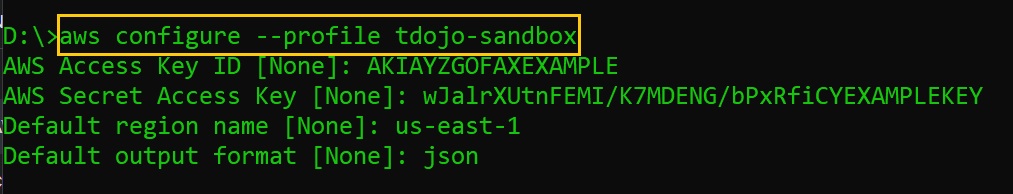
Which combination of steps must the developer do to resolve the issue? (Select TWO)

* + Add the AWS credentials of the sandbox AWS account to the Globals section of the template.yml file and reference them in the AWS::Serverless::Function properties section of the Lambda function.
  + Create an AWS SAM CLI configuration file at the root of the SAM project folder. Add the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables to it.
  + Run the function using sam local invoke with the --parameter-overrides parameter.
  + **Run the function using sam local invoke with the --profile parameter.**
  + **Use the aws configure command with the --profile parameter to add a named profile with the sandbox AWS account’s credentials.**

**Correct**

AWS Lambda functions have an associated execution role that provides permissions to interact with other AWS services. However, when you run AWS Lambda functions locally using the SAM CLI, you’re simulating the execution environment of the Lambda function but not replicating the AWS execution context, including the IAM execution role. This means that the function won’t automatically assume any IAM execution role and instead will rely on the credentials stored in ~/.aws/credentials file.

When testing locally with AWS SAM, you can specify a named profile from your AWS CLI configuration using the --profile parameter with the sam local invoke command. This will instruct the SAM CLI to use the credentials from the specified profile when invoking the Lambda function. You can run the aws configure  with the --profile option to set the credentials for a named profile.



In the scenario, the developer must first set up the sandbox AWS account’s credentials using aws configure --profile sandbox. This creates a named profile ‘sandbox’ (note that you can use any name for the profile). For local testing with the SAM CLI, the developer can then specify this profile using the command sam local invoke --profile sandbox. This ensures that the locally executed Lambda function utilizes the correct credentials to access resources in the sandbox AWS account.

Hence, the correct answers are:

**– Use the aws configure command with the --profile parameter to add a named profile with the sandbox AWS account’s credentials.**

**– Run the function using sam local invoke with the --profile parameter.**

The option that says: **Create an AWS SAM CLI configuration file at the root of the SAM project folder. Add the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables to it**is incorrect. The SAM CLI relies on the AWS credentials stored in the /.aws/credentials file, which can be set through the aws configure command. While it’s technically possible to place application credentials in a configuration file, SAM CLI doesn’t support sourcing AWS credentials from it for authentication.

The option that says: **Add the AWS credentials of the sandbox AWS account to the Globals section of the template.yml file and reference them in the AWS::Serverless::Function properties section of the Lambda function** is incorrect. The Globals section in a SAM template.yaml is primarily used for setting properties that apply to all AWS resources of a certain type. It’s not a storage location for AWS credentials. Moreover, the AWS::Serverless::Function resource property does not have fields for AWS credentials. Even if you were to add the credentials as environment variables, it still wouldn’t grant the locally running function the permissions associated with those credentials.

The option that says: **Run the function using sam local invoke with the --parameter-overrides parameter**is incorrect. The --parameter-overrides option is typically used to change template parameters during local testing. For instance, if you had a parameter in your SAM template for setting an environment variable, the --parameter-overrides option would allow you to test with different values for those parameters. Still, it does not interact with nor modify AWS credentials.

**References:**

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-cli-command-reference-sam-local-invoke.html>

<https://aws.amazon.com/blogs/aws/aws-serverless-application-model-sam-command-line-interface-build-test-and-debug-serverless-apps-locally/>

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html>

**Check out this AWS SAM Cheat Sheet:**

<https://tutorialsdojo.com/aws-serverless-application-model-sam/>

1. **41. Question**

An application hosted in a multicontainer Docker platform in Elastic Beanstalk uses DynamoDB to handle the session data of its users. These data are only used in a particular timeframe and the stale data can be deleted after the user logged out of the system.

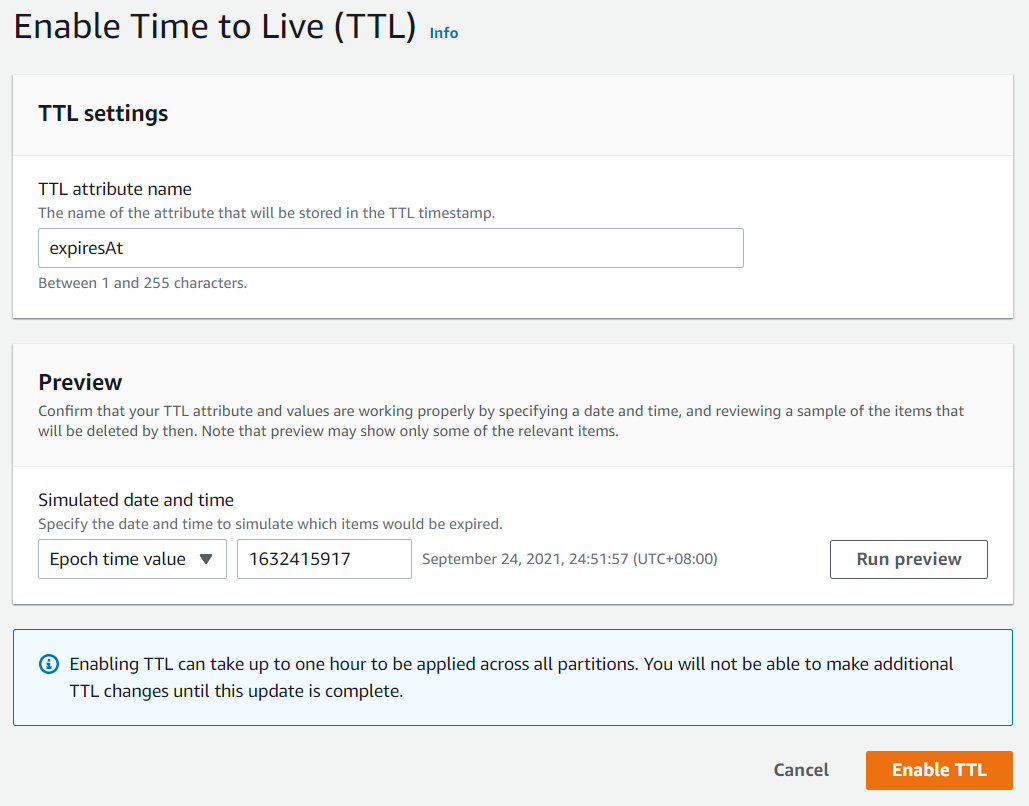
Which of the following is the most suitable way to delete the session data?

* + Use conditional writes to add the session data to the DynamoDB table and then automatically delete it based on the condition you specify.
  + **Enable TTL for the session data in the DynamoDB table.**
  + Use atomic counters to track the validity of the session data and delete once it becomes stale.
  + Delete the stale data by regularly performing a scan on the table.

**Correct**

**Time To Live (TTL)** for DynamoDB allows you to define when items in a table expire so that they can be automatically deleted from the database.

TTL is provided at no extra cost as a way to reduce storage usage and reduce the cost of storing irrelevant data without using provisioned throughput. With TTL enabled on a table, you can set a timestamp for deletion on a per-item basis, allowing you to limit storage usage to only those records that are relevant.



TTL is useful if you have continuously accumulated data that lose relevance after a specific time period. For example session data, event logs, usage patterns, and other temporary data. If you have sensitive data that must be retained only for a certain amount of time according to contractual or regulatory obligations, TTL helps you ensure that it is removed promptly and as scheduled.

Therefore, the correct answer in this scenario is to: **Enable TTL for the session data in the DynamoDB table**.

The option that says: **Delete the stale data by regularly performing a scan on the table**is incorrect because the Scan operation uses *eventually consistent* reads when accessing the data in a table and therefore, the result set might not include the changes to data in the table immediately before the operation began. This is an inefficient option that can simply be replaced by using TTL.

The option that says: **Use atomic counters to track the validity of the session data and deleting it once becomes stale** is incorrect because atomic counters are primarily used in updating data and for scenarios where you want the updates to not be idempotent.

The option that says: **Use conditional writes to add the session data to the DynamoDB table and then automatically deleting it based on the condition you specify** is incorrect because conditional writes are only helpful in cases where multiple users attempt to modify the same item.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithItems.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

**Tutorials Dojo’s AWS Certified Developer Associate Exam Study Guide:**

<https://tutorialsdojo.com/aws-certified-developer-associate/>

1. **42. Question**

An application performs various workflows and processes long-running tasks that take a long time to complete. The users are complaining that the application is unresponsive since the workflow substantially increased the time it takes to complete a user request.

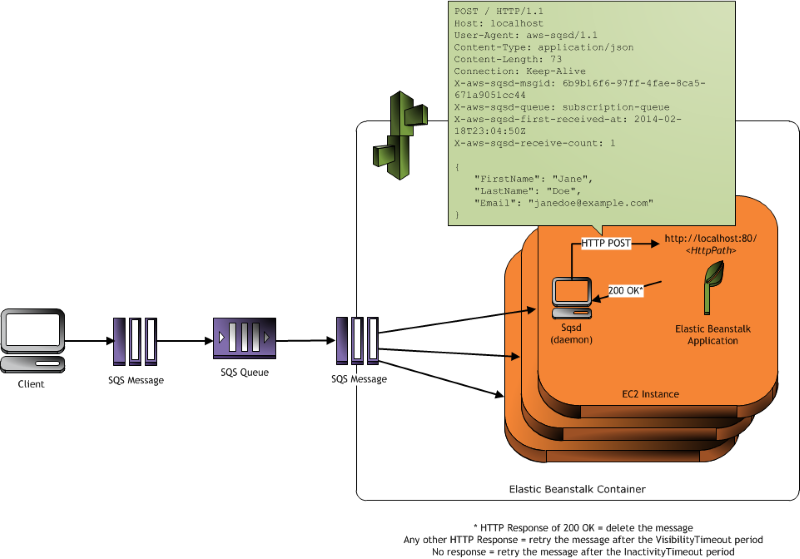
Which of the following is the BEST way to improve the performance of the application?

* + **Use an Elastic Beanstalk worker environment to process the tasks asynchronously.**
  + Use a multicontainer docker environment in Elastic Beanstalk to process the long-running tasks asynchronously.
  + Spawn a worker process locally in the EC2 instances and process the tasks asynchronously.
  + Use an Amazon ECS Cluster with a Fargate launch type to process the tasks asynchronously.

**Correct**

If your application performs operations or workflows that take a long time to complete, you can offload those tasks to a dedicated *worker environment*. Decoupling your web application front end from a process that performs blocking operations is a common way to ensure that your application stays responsive under load.

A long-running task is anything that substantially increases the time it takes to complete a request, such as processing images or videos, sending emails, or generating a ZIP archive. These operations can take only a second or two to complete, but a delay of a few seconds is a lot for a web request that would otherwise complete in less than 500 ms.



One option is to spawn a worker process locally, return success, and process the task asynchronously. This works if your instance can keep up with all of the tasks sent to it. Under high load, however, an instance can become overwhelmed with background tasks and become unresponsive to higher-priority requests. If individual users can generate multiple tasks, the increase in load might not correspond to an increase in users, making it hard to scale out your web server tier effectively.

To avoid running long-running tasks locally, you can use the AWS SDK for your programming language to send them to an Amazon Simple Queue Service (Amazon SQS) queue and run the process that performs them on a separate set of instances. You then design these worker instances to take items from the queue only when they have the capacity to run them, preventing them from becoming overwhelmed.

Elastic Beanstalk worker environments simplify this process by managing the Amazon SQS queue and running a [daemon process](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html#worker-daemon) on each instance that reads from the queue for you. When the daemon pulls an item from the queue, it sends an HTTP POST request locally to http://localhost/ on port 80 with the contents of the queue message in the body. All that your application needs to do is perform the long-running task in response to the POST. You can [configure the daemon](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html#using-features-managing-env-tiers-worker-settings) to post to a different path, use a MIME type other than application/JSON, connect to an existing queue, or customize connections (maximum concurrent requests), timeouts, and retries.

Hence, the best solution to meet the requirements of this scenario is to **use an Elastic Beanstalk worker environment to process the tasks asynchronously.**

**Spawning a worker process locally in the EC2 instances then processing the tasks asynchronously**is incorrect. Although this is a valid solution, it is not scalable and hence, it’s not the best one. Under high load, an instance can become overwhelmed with background tasks and become unresponsive to higher priority requests. This makes it hard to scale out your web server tier effectively.

**Using a multicontainer docker environment in Elastic Beanstalk to process the long-running tasks asynchronously** is incorrect because this is primarily used to support multiple containers per Amazon EC2 instance with multicontainer Docker platform. This is not applicable when processing long-running tasks and it is not scalable since it’s not using an SQS queue.

**Using an Amazon ECS Cluster with a Fargate launch type to process the tasks asynchronously**is incorrect because Fargate just allows you to run your containerized applications without the need to provision and manage the backend infrastructure.

**References:**

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html>

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs.html>

**Check out this AWS Elastic Beanstalk Cheat Sheet:**

<https://tutorialsdojo.com/aws-elastic-beanstalk/>

1. **43. Question**

A developer is instructed to configure a worker daemon to queue messages based on a specific schedule using a worker environment hosted in Elastic Beanstalk. Periodic tasks should be defined to automatically add jobs to your worker environment’s queue at regular intervals.

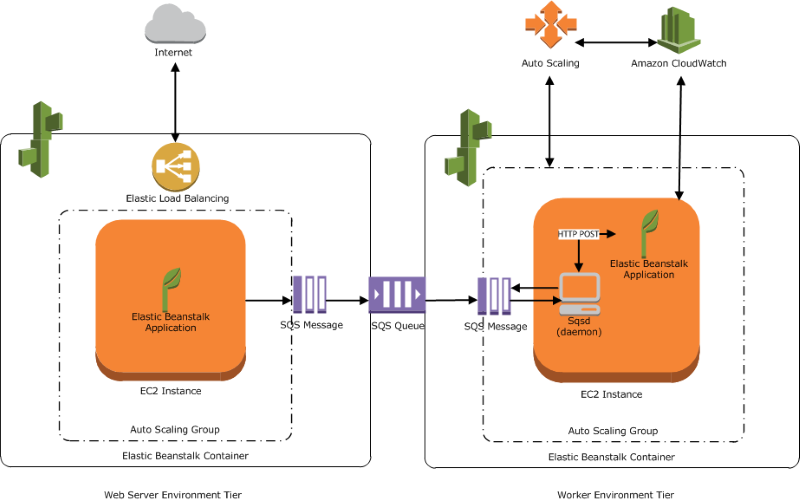
Which configuration file should the developer add to the source bundle to meet the above requirement?

* + Dockerrun.aws.json
  + **cron.yaml**
  + env.yaml
  + appspec.yml

**Correct**

AWS resources created for a worker environment tier include an Auto Scaling group, one or more Amazon EC2 instances, and an IAM role. For the worker environment tier, Elastic Beanstalk also creates and provisions an Amazon SQS queue if you don’t already have one. When you launch a worker environment tier, Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each EC2 instance in the Auto Scaling group.

The daemon is responsible for pulling requests from an Amazon SQS queue and then sending the data to the web application running in the worker environment tier that will process those messages. If you have multiple instances in your worker environment tier, each instance has its own daemon, but they all read from the same Amazon SQS queue.



You can define periodic tasks in a file named **cron.yaml** in your source bundle to add jobs to your worker environment’s queue automatically at a regular intervals. For example, you can configure and upload a cron.yaml file, which creates two periodic tasks: one that runs every 12 hours and a second that runs at 11 pm UTC every day.

Hence, using the **cron.yaml**is the correct configuration file to be used in this scenario.

**Dockerrun.aws.json**is incorrect because this configuration file is primarily used in multi-container Docker environments that are hosted in Elastic Beanstalk. This can be used alone or combined with source code and content in a source bundle to create an environment on a Docker platform.

**env.yaml**is incorrect because this is primarily used to configure the environment name, solution stack, and environment links to use when creating your environment in Elastic Beanstalk.

**appspec.yml**is incorrect because this is used to manage each application deployment as a series of lifecycle event hooks in CodeDeploy and not in Elastic Beanstalk.

**References:**

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html#worker-periodictasks>

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/applications-sourcebundle.html>

**Check out this AWS Elastic Beanstalk Cheat Sheet:**

<https://tutorialsdojo.com/aws-elastic-beanstalk/>

1. **44. Question**

To improve their information security management system (ISMS), a company recently released a new policy which requires all database credentials to be encrypted and be automatically rotated to avoid unauthorized access.

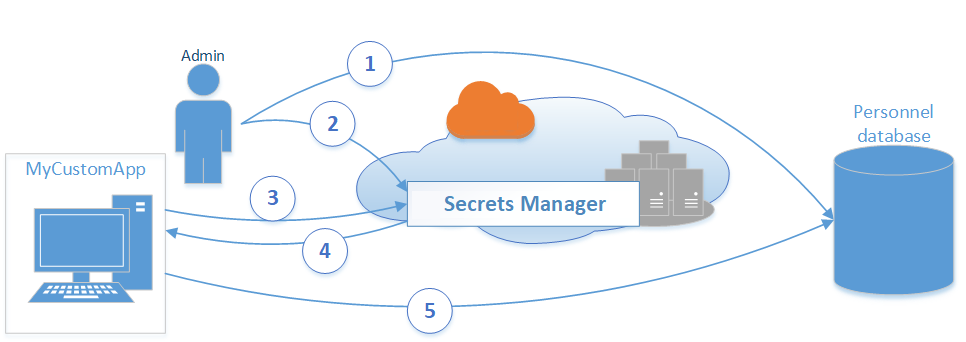
Which of the following is the MOST appropriate solution to secure the credentials?

* + **Create a secret in AWS Secrets Manager and enable automatic rotation of the database credentials.**
  + Create an IAM Role which has full access to the database. Attach the role to the services which require access.
  + Enable IAM DB authentication which rotates the credentials by default.
  + Create a parameter to the Systems Manager Parameter Store using the PutParameter API with a type of SecureString.

**Correct**

**AWS Secrets Manager** is an AWS service that makes it easier for you to manage secrets. *Secrets* can be database credentials, passwords, third-party API keys, and even arbitrary text. You can store and control access to these secrets centrally by using the Secrets Manager console, the Secrets Manager command line interface (CLI), or the Secrets Manager API and SDKs.

In the past, when you created a custom application that retrieves information from a database, you typically had to embed the credentials (the secret) for accessing the database directly in the application. When it came time to rotate the credentials, you had to do much more than just create new credentials. You had to invest time to update the application to use the new credentials. Then you had to distribute the updated application. If you had multiple applications that shared credentials and you missed updating one of them, the application would break. Because of this risk, many customers have chosen not to regularly rotate their credentials, which effectively substitutes one risk for another.



Secrets Manager enables you to replace hardcoded credentials in your code (including passwords), with an API call to Secrets Manager to retrieve the secret programmatically. This helps ensure that the secret can’t be compromised by someone examining your code, because the secret simply isn’t there. Also, you can configure Secrets Manager to automatically rotate the secret for you according to a schedule that you specify. This enables you to replace long-term secrets with short-term ones, which helps to significantly reduce the risk of compromise.

Hence, **creating a secret in AWS Secrets Manager and enabling automatic rotation of the database credentials** is the most appropriate solution for this scenario.

The option that says: **Create a parameter to the Systems Manager Parameter Store using the PutParameter API with a type of SecureString**is incorrect because, by default, Systems Manager Parameter Store doesn’t rotate its parameters.

The option that says: **Enable IAM DB authentication which rotates the credentials by default** is incorrect because this solution only enables the service to connect to Amazon RDS with IAM credentials. It doesn’t have the capability to rotate the credentials like what AWS Secrets Manager does to its secrets.

The option that says:**Create an IAM Role which has full access to the database. Attach the role to the services which requires access** is incorrect because although IAM Role is a preferred way to grant access to certain services, this solution doesn’t rotate the keys/credentials.

**References:**

<https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-paramstore.html>

<https://aws.amazon.com/blogs/compute/sharing-secrets-with-aws-lambda-using-aws-systems-manager-parameter-store/>

**Check out this AWS Secrets Manager Cheat Sheet:**

<https://tutorialsdojo.com/aws-secrets-manager/>

1. **45. Question**

A programmer is developing a shell script that uses AWS CLI to list all objects of a given bucket. However, the script is timing out if the bucket has tens of thousands of objects.

Which solution would most likely rectify the issue?

* + **Add pagination parameters in the AWS CLI command**
  + Enable Amazon S3 Transfer Acceleration
  + Use S3 Select
  + Enable CORS

**Correct**

For commands that can return a large list of items, the AWS CLI provides options that you can use to control the number of items included in the output when the AWS CLI calls a service’s API to populate the list.

By default, the AWS CLI uses a page size of 1000 and retrieves all available items. If you see issues when running list commands on a large number of resources, the default page size of 1000 might be too high. This can cause calls to AWS services to exceed the maximum allowed time and generate a “timed out” error.

One of the pagination options you can use is the --page-size option. This option tells the AWS CLI to request a smaller number of items from each call to the AWS service.

**aws s3api list-objects --bucket tdbucket --page-size 10**0

The CLI still retrieves the full list but performs a larger number of service API calls in the background and retrieves a smaller number of items with each call. This gives the individual calls a better chance of succeeding without a timeout.

Hence, the correct answer is: **Add pagination parameters in the AWS CLI command.**

The option that says: **Using S3 Select** is incorrect because Amazon S3 Select only provides you with the capability to use simple structured query language (SQL) statements to filter the contents of Amazon S3 objects and retrieve just the subset of data that you need. It also does not paginate the results from your AWS CLI request.

The option that says: **Enabling Amazon S3 Transfer Acceleration** is incorrect because this is only a bucket-level feature that enables faster data transfers to and from Amazon S3. Although this will improve the retrieval times of your objects, this feature will still not paginate the result, which may still cause time-out errors.

The option that says: **Enabling CORS** is incorrect because the Cross-origin resource sharing (CORS) feature just defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. This is not useful in paginating the results from an AWS CLI call.

**References:**

<https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-pagination.html>

<https://docs.aws.amazon.com/cli/latest/reference/s3api/list-objects.html>

1. **46. Question**

An application, which already uses X-Ray, generates thousands of trace data every hour. The developer wants to use a filter expression that will limit the results based on custom attributes or keys that he specifies.

How should the developer refactor the application in order to filter the results in the X-Ray console?

* + Add the custom attributes as metadata in your segment document.
  + Include the custom attributes as new segment fields in the segment document.
  + **Add the custom attributes as annotations in your segment document.**
  + Create a new sampling rule based on the custom attributes.

**Correct**

Even with sampling, a complex application generates a lot of data. The AWS X-Ray console provides an easy-to-navigate view of the service graph. It shows health and performance information that helps you identify issues and opportunities for optimization in your application. For advanced tracing, you can drill down to traces for individual requests or use **filter expressions** to find traces related to specific paths or users.



When you instrument your application, the X-Ray SDK records information about incoming and outgoing requests, the AWS resources used, and the application itself. You can add other information to the segment document as annotations and metadata.

**Annotations** are simple key-value pairs that are indexed for use with [filter expressions](https://docs.aws.amazon.com/xray/latest/devguide/xray-console-filters.html). Use annotations to record data that you want to use to group traces in the console, or when calling the [GetTraceSummaries](https://docs.aws.amazon.com/xray/latest/api/API_GetTraceSummaries.html" \t "_blank) API. X-Ray indexes up to 50 annotations per trace.

**Metadata** are key-value pairs with values of any type, including objects and lists, but that are not indexed. Use metadata to record data you want to store in the trace but don’t need to use for searching traces. You can view annotations and metadata in the segment or subsegment details in the X-Ray console.

Hence, **adding the custom attributes as annotations in your segment document** is the correct answer.

**Including the custom attributes as new segment fields in the segment document** is incorrect because a segment field can’t be used as a filter expression. You have to add the custom attributes as annotations to the segment document that you’ll send to X-Ray, just as mentioned above.

**Creating a new sampling rule based on the custom attributes**is incorrect because sampling is primarily used to ensure efficient tracing and to provide a representative sample of the requests that your application serves.

**Adding the custom attributes as metadata in your segment document**is incorrect because metadata is primarily used to record custom data that you want to store in the trace but not for searching traces.

**References:**

<https://docs.aws.amazon.com/xray/latest/devguide/xray-concepts.html#xray-concepts-annotations>

<https://docs.aws.amazon.com/xray/latest/devguide/xray-console-filters.html>

**Check out this AWS X-Ray Cheat Sheet:**

<https://tutorialsdojo.com/aws-x-ray/>

1. **47. Question**

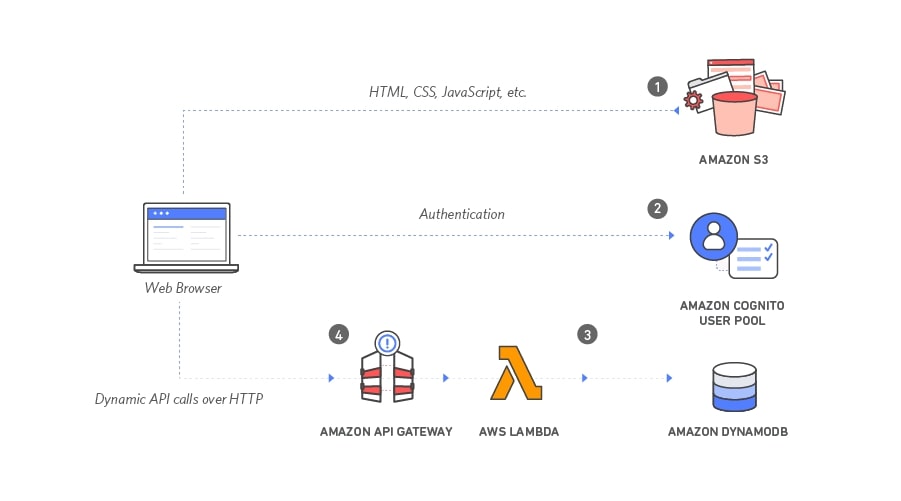
A serverless application consisting of Lambda functions integrated with API Gateway, and DynamoDB processes ad hoc requests that its users send. Due to the recent spike in incoming traffic, some of your customers are complaining that they are getting HTTP 504 errors from time to time.

Which of the following is the MOST likely cause of this issue?

* + **API Gateway request has timed out because the underlying Lambda function has been running for more than 29 seconds.**
  + The usage plan quota has been exceeded for the Lambda function.
  + An authorization failure occurred between API Gateway and the Lambda function.
  + Since the incoming requests are increasing, the API Gateway automatically enabled throttling which caused the HTTP 504 errors.

**Correct**

A gateway response is identified by a response type defined by API Gateway. The response consists of an HTTP status code, a set of additional headers that are specified by parameter mappings, and a payload that is generated by a non-VTL (Apache Velocity Template Language) mapping template.



You can set up a gateway response for a supported response type at the API level. Whenever API Gateway returns a response of the type, the header mappings and payload mapping templates defined in the gateway response are applied to return the mapped results to the API caller.

The following are the Gateway response types which are associated with the HTTP 504 error in API Gateway:

**INTEGRATION\_FAILURE** – The gateway response for an integration failed error. If the response type is unspecified, this response defaults to the DEFAULT\_5XX type.

**INTEGRATION\_TIMEOUT** – The gateway response for an integration timed out error. If the response type is unspecified, this response defaults to the DEFAULT\_5XX type.

For the integration timeout, the range is from 50 milliseconds to 29 seconds for all integration types, including Lambda, Lambda proxy, HTTP, HTTP proxy, and AWS integrations.

In this scenario, there is an issue where the users are getting HTTP 504 errors in the serverless application. This means the Lambda function is working fine at times but there are instances when it throws an error. Based on this analysis, the most likely cause of the issue is the **INTEGRATION\_TIMEOUT** error since you will only get an **INTEGRATION\_FAILURE** error if your AWS Lambda integration does not work at all in the first place.

Hence, the root cause of this issue is that the**API Gateway request has timed out because the underlying Lambda function has been running for more than 29 seconds**.

The option that says: **Since the incoming requests are increasing, the API Gateway automatically enabled throttling which caused the HTTP 504 errors** is incorrect because a large number of incoming requests will most likely produce an HTTP 502 or 429 error but not a 504 error. If executing the function would cause you to exceed a concurrency limit at either the account level (***ConcurrentInvocationLimitExceeded***) or function level (***ReservedFunctionConcurrentInvocationLimitExceeded***), Lambda may return a ***TooManyRequestsException*** as a response. For functions with a long timeout, your client might be disconnected during synchronous invocation while it waits for a response and returns an HTTP 504 error.

The option that says: **An authorization failure occurred between API Gateway and the Lambda function** is incorrect because an authentication issue usually produces HTTP 403 errors and not 504s. The gateway response for authorization failures for missing authentication token error, invalid AWS signature error, or Amazon Cognito authentication problems is HTTP 403, which is why this option is unlikely to be the cause of this issue.

The option that says: **The usage plan quota has been exceeded for the Lambda function** is incorrect. Although this is a possible root cause for this scenario, this option has the least chance to produce HTTP 504 errors. The scenario says that the issue happens from *time to time* and not *all the time* which suggests that this happens intermittently. If the usage plan indeed exceeded the quota, then the 504 error should always show up and not just from*time to time*.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html>

<https://aws.amazon.com/about-aws/whats-new/2017/11/customize-integration-timeouts-in-amazon-api-gateway/>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/supported-gateway-response-types.html>

**Check out this API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **48. Question**

A company operates an e-commerce website on Amazon Elastic Container Service (ECS) behind an Application Load Balancer (ALB). They’ve set their ALB as an origin for an Amazon CloudFront distribution. Users interact with the website via a custom domain linked to the CloudFront distribution, all maintained within a public hosted zone in Amazon Route 53.

The company wants to display region-specific pricing tables to its users. For example, when a user from the UK visits the site, they should be redirected to https://tutorialsdojo.com/uk/, while users from the Philippines should view prices on https://tutorialsdojo.com/ph/

How can a developer incorporate this feature in the least amount of development overhead?

* + Use AWS Web Application Firewall (WAF's) geo-matching rule to identify the user country and attach it to the ALB. Configure ALB listener rules with path conditions to route traffic based on the identified country.
  + Configure the Route 53 record to use the geolocation routing policy.
  + Forward the CloudFront-Viewer-Address header to the web server running on the ECS cluster. Implement a custom logic that matches the header’s value against a GeoIP database to determine user location. Based on the resolved location, redirect users to the appropriate region-specific URL.
  + **Implement a CloudFront function that returns the appropriate URL based on the CloudFront-Viewer-Country. Configure the distribution to trigger the function on Viewer request events.**

**Correct**

You can configure CloudFront to add specific HTTP headers to the requests that CloudFront receives from viewers and forwards to your origin or [edge function](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/edge-functions.html). The values of these HTTP headers are based on the characteristics of the viewer or the viewer request. The headers provide information about the viewer’s device type, IP address, geographic location, request protocol (HTTP or HTTPS), HTTP version, TLS connection details, and JA3 fingerprint.

With these headers, your origin or your edge function can receive information about the viewer without the need for you to write your own code to determine this information. If your origin returns different responses based on the information in these headers, you can include them in the *cache key* so that CloudFront caches the responses separately. For example, your origin might respond with content in a specific language based on the country that the viewer is in or with content tailored to a specific device type. Your origin might also write these headers to log files, which you can use to determine information about where your viewers are, which device types they’re on, and more.

In the given scenario, when a user initiates a request to the website, a CloudFront function can be triggered to inspect the CloudFront-Viewer-Country header, which pinpoints the originating country of a user. CloudFront functions are lightweight Javascript code that you can use to manipulate web requests at the CloudFront edge locations. By using a CloudFront function triggered on “Viewer request” events, you can assess and act upon incoming requests even before they reach the origin or CloudFront retrieves a cached response.

Here’s a representation using a CloudFront function code snippet:



Hence, the correct answer is: **Implement a CloudFront function that returns the appropriate URL based on the CloudFront-Viewer-Country. Configure the distribution to trigger the function on Viewer request events.**

The option that says: **Forward the CloudFront-Viewer-Address header to the web server running on the ECS cluster. Implement a custom logic that matches the header’s value against a GeoIP database to determine user location. Based on the resolved location, redirect users to the appropriate region-specific URL**is incorrect. While this approach is technically valid, it’s more complex since you have to handle the IP-to-location translation on the backend and maintain an up-to-date GeoIP database.

The option that says: **Configure the Route 53 record to use the geolocation routing policy** is incorrect. Route 53 geolocation routing is primarily used for directing traffic to specific resources based on user location for performance or regulatory reasons, not for content personalization based on geolocation.

The option that says: **Use AWS Web Application Firewall (WAF’s) geo-matching rule to identify the user country and attach it to the ALB. Configure ALB listener rules with path conditions to route traffic based on the identified country**is incorrect. AWS WAF’s geo-matching rule identifies a user’s country based on their IP. However, it’s primarily designed to allow or block access, not for redirection. Even if used in conjunction with ALB, the ALB’s listener rules can’t inherently make routing decisions based on a viewer’s geolocation

**References:**

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/adding-cloudfront-headers.html>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-functions.html>

**Check out this Amazon CloudFront Cheat Sheet:**

<https://tutorialsdojo.com/amazon-cloudfront/>

1. **49. Question**

A company offers a Generative Artificial Intelligence (AI) service exposed through a REST API managed by Amazon API Gateway. They recently rolled out a subscription tier where users receive API keys to access premium features. The company uses the CreateApiKey API for generating these keys.

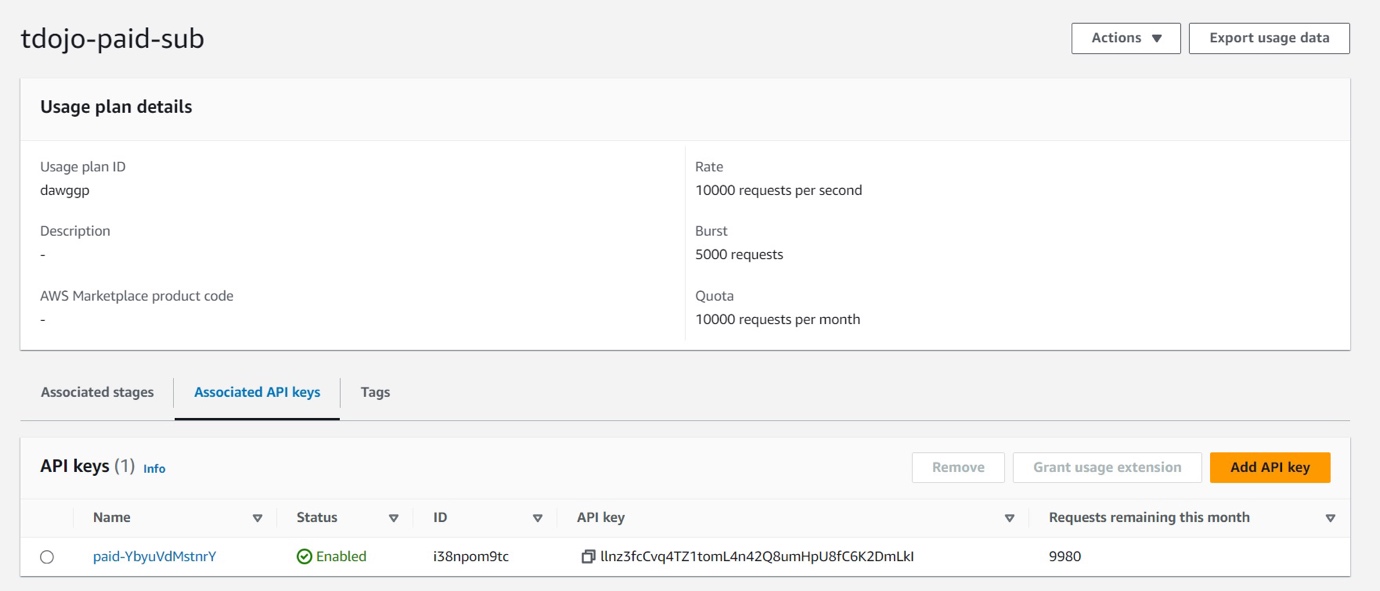
During testing, developers noticed that while existing users can access the service without issues, new premium subscribers get a 403 Forbidden error when using their API keys.

What must be done to give new users access to the service?

* + Instruct users to send their API key in a custom header. In the integration request, adjust the mapping template to extract and evaluate this header to distinguish between free-tier and premium subscribers.
  + **Associate the API keys for the premium users with the intended usage plan using the CreateUsagePlanKey operation.**
  + Use the ImportApiKeys operation to import the premium users’ keys, then apply the UpdateUsagePlan operation to set the new tier access.
  + Use the UpdateAuthorizer operation to modify the authorization settings. Promote the changes to the production stage by calling the CreateDeployment operation.

**Correct**

In Amazon API Gateway, API keys by themselves do not grant access to execute an API. They need to be associated with a usage plan, and that usage plan then determines which API stages and methods the API key can access.



If the API key is not associated with a usage plan, it will not have permission to access any of the resources, which will result in a “403 Forbidden” error.

In the given scenario, existing users can access the service, but new premium subscribers cannot. This indicates that while the API keys were created for new users, they might not have been associated with the appropriate usage plan. Hence, after generating an API key, it must be added to a usage plan by calling the **CreateUsagePlanKey**method**.**

Hence, the correct answer is: **Associate the API keys for the premium users with the intended usage plan using the CreateUsagePlanKey operation.**

The option that says: **Use the ImportApiKeys operation to import the premium users’ keys, then apply the UpdateUsagePlan operation to set the new tier access**is incorrect. The importApiKeys API is primarily used for bulk importing API keys, not for associating them with a usage plan. Although the updateUsagePlan API modifies properties of a usage plan; it doesn’t handle direct association of API keys.

The option that says: **Use the UpdateAuthorizer operation to modify the authorization settings. Promote the changes to the production stage by calling the CreateDeployment operation** is incorrect. The updateAuthorizer operation is only used to modify the settings of an existing custom authorizer, which handles custom authorization logic for APIs. In the scenario, the issue is not related to custom authorization but rather to the association of API keys with a usage plan.

The option that says: **Instruct users to send their API key in a custom header. In the integration request, adjust the mapping template to extract and evaluate this header to distinguish between free-tier and premium subscribers**is incorrect. Changing the way users provide their API key adds unnecessary complexity and won’t solve the issue at hand. The problem isn’t with how the API key is being sent but with the API key not having appropriate permissions because it’s not associated with a usage plan.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/api/API_UpdateUsagePlan.html>

<https://docs.aws.amazon.com/apigateway/latest/api/API_CreateApiKey.html>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **50. Question**

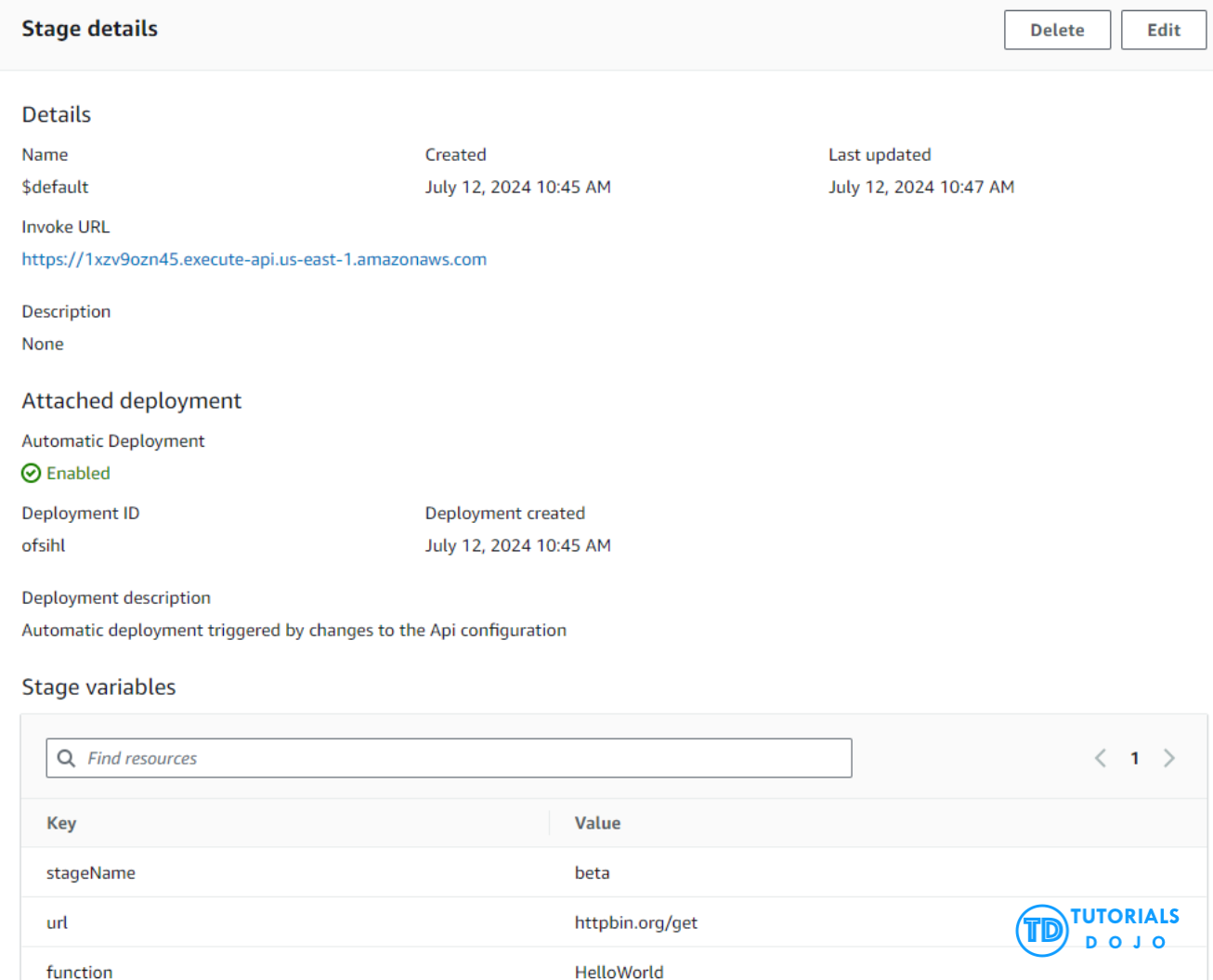
You have several API Gateway APIs with Lambda Integration for each release life cycle of your application. There is a requirement to consolidate multiple releases into a single API Gateway for the ALPHA, BETA, RC (Release Candidate), and PROD releases. For example, their clients can connect to their ALPHA release by using the alpha.tutorialsdojo.com endpoint and beta release through the beta.tutorialsdojo.com endpoint.

As the AWS developer, how can you satisfy this requirement?

* + Modify the Integration Response of the API Gateway to add different endpoints for each release.
  + **Set up Stage Variables for each release.**
  + Use Layers to the underlying Lambda functions of the API Gateway.
  + Modify the Integration Request of the API Gateway to manage different endpoints for each release.

**Correct**

**Amazon API Gateway** is a service that simplifies the creation, publishing, maintenance, monitoring, and securing of APIs at any scale. API Gateway enables you to define and manage the interface between front-end clients and backend services. It handles all tasks associated with processing hundreds of thousands of concurrent API calls, including traffic management, authorization, access control, monitoring, and API version management. One of its key features is the ability to create different stages for an API, allowing developers to maintain multiple versions of the same API, such as development, testing, and production.



Stage Variables in API Gateway are name-value pairs that can be used to configure different settings for each stage of an API. These variables allow developers to reference different backend resources and settings dynamically without changing the API deployment. For instance, stage variables can be used to specify different Lambda function ARNs, database endpoints, or other environment-specific configurations. This capability is handy for managing multiple environments (like ALPHA, BETA, RC, and PROD) within the same API Gateway, ensuring that each environment can be independently configured and managed using a single consolidated API Gateway.

Hence, the correct answer is: **Set up Stage Variables for each release.**

The options that say:

**–** **Modify the Integration Response of the API Gateway to add different endpoints for each release.**

**– Modify the Integration Request of the API Gateway to manage different endpoints for each release.**

are both **incorrect** because these two are primarily used to map the response or the request data to and from your backend. There is no way to use the Integration Request or Response as a variable that can be a part of the URL string of an HTTP integration for a method in your REST API. Moreover, these two can’t be part of a custom API URL: **<*alpha|beta|rc|prod*>.tutorialsdojo.com**, which was mentioned in the scenario.

The option that says: **Use Layers to the underlying Lambda functions of the API Gateway**is incorrect because this is only applicable if you want to configure your Lambda function to pull in additional code and content in the form of layers. Remember that a layer is just a ZIP archive that contains libraries, a [custom runtime](https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html), or other dependencies.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/stage-variables.html>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-stages.html>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **51. Question**

A programmer is developing a Node.js application that will be run on a Linux server in their on-premises data center. The application will access various AWS services such as S3, DynamoDB, and ElastiCache using the AWS SDK.

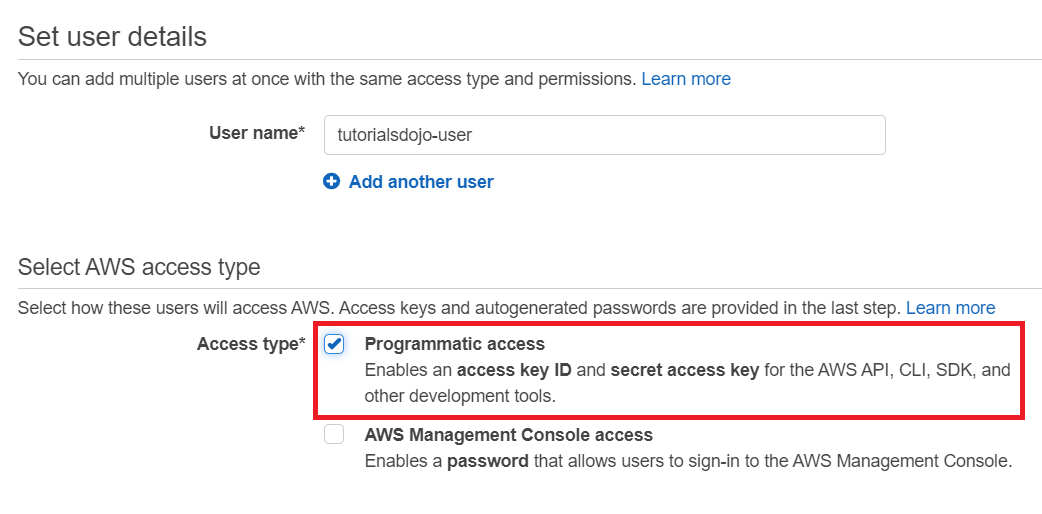
Which of the following is the MOST suitable way to provide access for the developer to accomplish the specified task?

* + **Go to the AWS Console and create a new IAM user with programmatic access. In the application server, create the credentials file at ~/.aws/credentials with the access keys of the IAM user.**
  + Go to the AWS Console and create a new IAM User with the appropriate permissions. In the application server, create the credentials file at ~/.aws/credentials with the username and the hashed password of the IAM User.
  + Create an IAM role with the appropriate permissions to access the required AWS services. Assign the role to the on-premises Linux server.
  + **Create an IAM role with the appropriate permissions to access the required AWS services and assign the role to the on-premises Linux server. Whenever the application needs to access any AWS services, request for temporary security credentials from STS using the AssumeRole API.**

**Incorrect**

If you have resources that are running inside AWS that need programmatic access to various AWS services, then the best practice is always to use IAM roles. However, applications running outside of an AWS environment will need access keys for programmatic access to AWS resources. For example, monitoring tools running on-premises and third-party automation tools will need access keys.

Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK).



In order to use the AWS SDK for your application, you have to create your credentials file first at ~/.aws/credentials for Linux servers or at C:\Users\USER\_NAME\.aws\credentials for Windows users and then save your access keys.

Hence, the correct answer is: **Go to the AWS Console and create a new IAM user with programmatic access. In the application server, create the credentials file at ~/.aws/credentials with the access keys of the IAM user.**

The option that says: **Create an IAM role with the appropriate permissions to access the required AWS services and assign the role to the on-premises Linux server. Whenever the application needs to access any AWS services, request for temporary security credentials from STS using the AssumeRole API** is incorrect because the scenario says that the application is running in a Linux server on-premises and not on an EC2 instance. You cannot directly assign an IAM Role to a server on your on-premises data center. Although it may be possible to use a combination of STS and IAM Role, the use of access keys for AWS SDK is still preferred, especially if the application server is on-premises.

The option that says: **Create an IAM role with the appropriate permissions to access the required AWS services. Assign the role to the on-premises Linux server** is also incorrect because, just as mentioned above, the use of an IAM Role is not a suitable solution for this scenario.

The option that says: **Go to the AWS Console and create a new IAM User with the appropriate permissions. In the application server, create the credentials file at ~/.aws/credentials with the username and the hashed password of the IAM User** is incorrect. An IAM user’s username and password can only be used to interact with AWS via its Management Console. These credentials are intended for human use and are not suitable for use in automated systems, such as applications and scripts that make programmatic calls to AWS services.

**References:**

<https://aws.amazon.com/developers/getting-started/nodejs/>

<https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys>

<https://aws.amazon.com/blogs/security/guidelines-for-protecting-your-aws-account-while-using-programmatic-access/>

**Check out this AWS IAM Cheat Sheet:**

<https://tutorialsdojo.com/aws-identity-and-access-management-iam/>

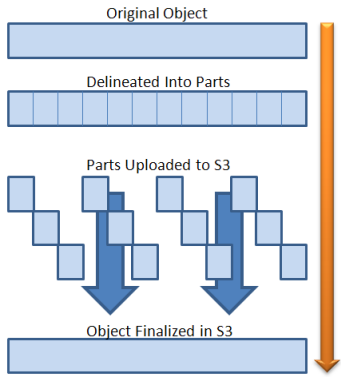
1. **52. Question**

A web application is uploading large files, which are over 4 GB in size, in an S3 bucket called [data.tutorialsdojo.com](http://data.tutorialsdojo.com/) every 30 minutes. You want to minimize the time required to upload each file. Which of the following should you do to minimize upload time?

* + Enable Transfer Acceleration in the bucket.
  + **Use the Multipart upload API.**
  + Use the BatchWriteItem API.
  + Use the Putltem API.

**Correct**

Multipart upload allows you to upload a single object as a set of parts. Each part is a contiguous portion of the object’s data. You can upload these object parts independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of your object are uploaded, Amazon S3 assembles these parts and creates the object. In general, when your object size reaches 100 MB, you should consider using multipart uploads instead of uploading the object in a single operation.



Using multipart upload provides the following advantages:

    – **Improved throughput** – You can upload parts in parallel to improve throughput.

– **Quick recovery from any network issues** – Smaller part size minimizes the impact of restarting a failed upload due to a network error.

– **Pause and resume object uploads** – You can upload object parts over time. Once you initiate a multipart upload, there is no expiry; you must explicitly complete or abort the multipart upload.

– **Begin an upload before you know the final object size** – You can upload an object as you are creating it.

Hence, **using the Multipart Upload API** is the correct answer in this scenario.

The options: **Use the BatchWriteItem API** and **Use the Putltem API** are incorrect because these are DynamoDB APIs and not S3.

The option that says: **Enable Transfer Acceleration in the bucket**is incorrect because although Transfer Acceleration will significantly reduce the upload time to S3, the bucket in the scenario won’t be able to turn on this feature. Take note that the name of the bucket used for Transfer Acceleration must be DNS-compliant and must not contain periods (“.”).

**Reference:**

<https://docs.aws.amazon.com/AmazonS3/latest/dev/uploadobjusingmpu.html>

<https://docs.aws.amazon.com/AmazonS3/latest/dev/transfer-acceleration.html>

**Check out this Amazon S3 Cheat Sheet:**

<https://tutorialsdojo.com/amazon-s3/>

**S3 Transfer Acceleration vs Direct Connect vs VPN vs Snowball vs Snowmobile:**

<https://tutorialsdojo.com/s3-transfer-acceleration-vs-direct-connect-vs-vpn-vs-snowball-vs-snowmobile/>

1. **53. Question**

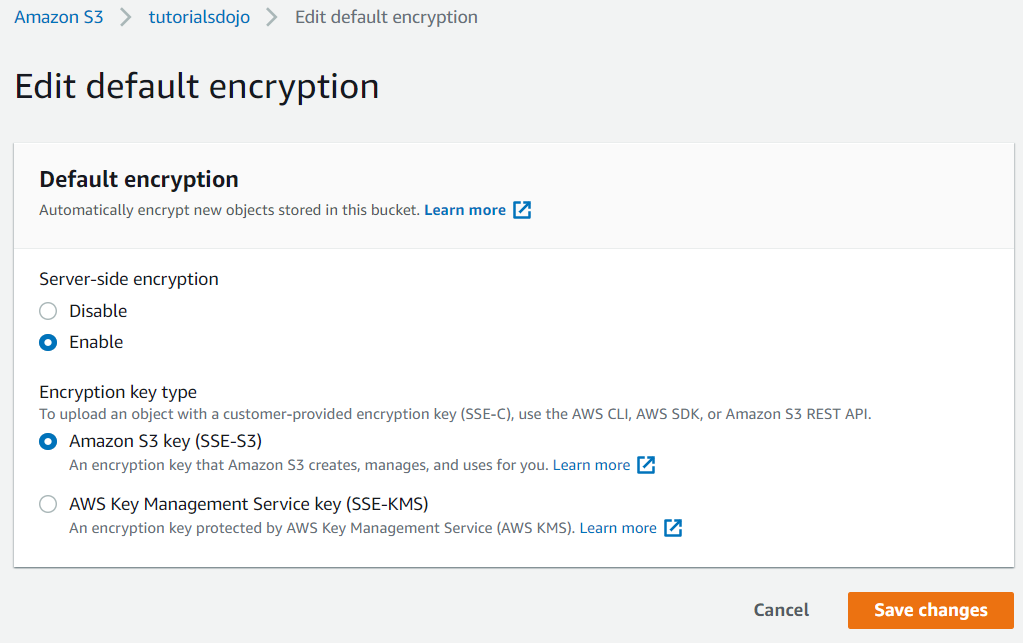
A developer needs to encrypt all objects being uploaded by their application to the S3 bucket to comply with the company’s security policy. The bucket will use server-side encryption with Amazon S3-Managed encryption keys (SSE-S3) to encrypt the data using 256-bit Advanced Encryption Standard (AES-256) block cipher.

Which of the following request headers should the developer use?

* + x-amz-server-side-encryption-customer-key
  + x-amz-server-side-encryption-customer-algorithm
  + **x-amz-server-side-encryption**
  + x-amz-server-side-encryption-customer-key-MD5

**Correct**

**Server-side encryption** protects data at rest. If you use Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3), Amazon S3 will encrypt each object with a unique key and as an additional safeguard, it encrypts the key itself with a master key that it rotates regularly. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data.



If you need server-side encryption for all of the objects that are stored in a bucket, use a bucket policy. For example, the following bucket policy denies permissions to upload an object unless the request includes the x-amz-server-side-encryption header to request server-side encryption:



However, if you chose to use server-side encryption with customer-provided encryption keys (SSE-C), you must provide encryption key information using the following request headers:

**x-amz-server-side​-encryption​-customer-algorithm**

**x-amz-server-side​-encryption​-customer-key**

**x-amz-server-side​-encryption​-customer-key-MD5**

Hence, using the **x-amz-server-side-encryption** header is correct as this is the one being used for Amazon S3-Managed Encryption Keys (SSE-S3). All other options are incorrect since they are used for SSE-C.

**References:**

<https://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html>

<https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingServerSideEncryption.html>

<https://docs.aws.amazon.com/AmazonS3/latest/dev/ServerSideEncryptionCustomerKeys.html>

**Check out this Amazon S3 Cheat Sheet:**

<https://tutorialsdojo.com/amazon-s3/>

1. **54. Question**

You developed a shell script which uses AWS CLI to create a new Lambda function. However, you received an InvalidParameterValueException after running the script.

What is the MOST likely cause of this issue?

* + **You provided an IAM role in the CreateFunction API which AWS Lambda is unable to assume.**
  + The resource already exists.
  + The AWS Lambda service encountered an internal error.
  + You have exceeded your maximum total code size per account.

**Correct**

To create a Lambda function, you need a deployment package and an execution role. The deployment package contains your function code. The execution role grants the function permission to use AWS services, such as Amazon CloudWatch Logs for log streaming and AWS X-Ray for request tracing. You can use the CreateFunction API via the AWS CLI or the AWS SDK of your choice.

A function has an unpublished version, and can have published versions and aliases. The unpublished version changes when you update your function’s code and configuration. A published version is a snapshot of your function code and configuration that can’t be changed. An alias is a named resource that maps to a version, and can be changed to map to a different version.

The InvalidParameterValueException will be returned if one of the parameters in the request is invalid. For example, if **you provided an IAM role in the CreateFunction API which AWS Lambda is unable to assume**. Hence, this option is the most likely cause of the issue in this scenario.

If ***y*ou have exceeded your maximum total code size per account**, the CodeStorageExceededException will be returned, which is why this option is incorrect.

If **the resource already exists**, the ResourceConflictException will be returned and not InvalidParameterValueException. Therefore, this option is also incorrect.

If **the AWS Lambda service encountered an internal error**, the ServiceException will be returned hence, this option is incorrect.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/API_CreateFunction.html>

<https://docs.aws.amazon.com/cli/latest/reference/lambda/create-function.html>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **55. Question**

A technical manager needs permission to create new repositories and delete them in CodeCommit. This will enable her to manage all of the code repositories of each development teams and remove duplicate or unused ones.

Which of the following permissions should be given in order to comply with the standard security advice of granting least privilege?

* + codecommit:GitPull and codecommit:GitPush
  + codecommit:CreateBranch and codecommit:DeleteBranch
  + codecommit:\*
  + **codecommit:CreateRepository and codecommit:DeleteRepository**

**Correct**

Access to AWS CodeCommit requires credentials. Those credentials must have permissions to access AWS resources, such as CodeCommit repositories, and your IAM user, which you use to manage your Git credentials or the SSH public key that you use for making Git connections.

In CodeCommit, the primary resource is a repository. Each resource has a unique Amazon Resource Names (ARN) associated with it. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. To manage access to AWS resources, you attach permissions policies to IAM identities. You use identity-based policies to control access to repositories.

The *codecommit:CreateRepository and codecommit:DeleteRepository* permissions enable you to create and delete a CodeCommit repository respectively. Hence, these two permissions are the correct permissions that should be given to the technical manager.

*codecommit:GitPull and codecommit:GitPush* permissions are incorrect because the codecommit:GitPull permission simply pulls information from a CodeCommit repository to a local repo and the codecommit:GitPush permission is just used to push information from a local repo to a CodeCommit repository.

*codecommit:CreateBranch and codecommit:DeleteBranch*permissions are incorrect because these enable you to create and delete a specific branch and not a repository. For this scenario, you have to use the codecommit:CreateRepository and codecommit:DeleteRepository permissions instead.

*codecommit:\** is incorrect because although this permission will allow all actions to the code repository. This entails a security risk and violates the standard security practice of granting only the permissions required to perform certain tasks.

**References:**

<https://docs.aws.amazon.com/codecommit/latest/userguide/auth-and-access-control-permissions-reference.html>

<https://docs.aws.amazon.com/codecommit/latest/userguide/auth-and-access-control-iam-access-control-identity-based.html>

**Check out this AWS CodeCommit Cheat Sheet:**

<https://tutorialsdojo.com/aws-codecommit/>

**AWS CodeCommit Repository:**

<https://tutorialsdojo.com/aws-codecommit-repository/>

1. **56. Question**

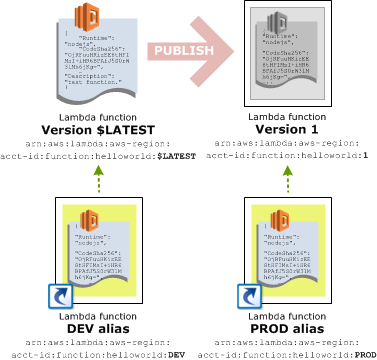
You want to update a Lambda function on your production environment and ensure that when you publish the updated version, you still have a quick way to roll back to the older version in case you encountered a problem. To prevent any sudden user interruptions, you want to gradually increase the traffic going to the new version.

Which of the following implementation is the BEST option to use?

* + Use stage variables in your Lambda function.
  + **Use Traffic Shifting with Lambda Aliases.**
  + Use Route 53 weighted routing to two Lambda functions.
  + Use ELB to route traffic to both Lambda functions.

**Correct**

By default, an alias points to a single Lambda function version. When the alias is updated to point to a different function version, incoming request traffic in turn instantly points to the updated version. This exposes that alias to any potential instabilities introduced by the new version. To minimize this impact, you can implement the routing-config parameter of the Lambda alias that allows you to point to two different versions of the Lambda function and dictate what percentage of incoming traffic is sent to each version.



For example, you can specify that only 2 percent of incoming traffic is routed to the new version while you analyze its readiness for a production environment, while the remaining 98 percent is routed to the original version. As the new version matures, you can gradually update the ratio as necessary until you have determined that the new version is stable. You can then update the alias to route all traffic to the new version.

You can point an alias to a maximum of two Lambda function versions. In addition:

– Both versions must have the same IAM execution role.

– Both versions must have the same [AWS Lambda Function Dead Letter Queues](https://docs.aws.amazon.com/lambda/latest/dg/dlq.html) configuration, or no DLQ configuration.

– When pointing an alias to more than one version, the alias cannot point to $LATEST.

Hence, **using Traffic Shifting for Lambda Aliases**is the correct answer.

**Using Route 53 weighted routing to two Lambda functions** is incorrect. Although you may configure 2 different endpoints for your Lambda versions and use Route 53 Weighted Routing, this is still not a manageable and convenient way of handling the failover of your serverless function. The best way is to use Lambda Aliases for the different versions of your function and do traffic shifting on these two versions.

**Using ELB to route traffic to both Lambda functions** is incorrect because this is not the recommended way to gradually deploy the new version of your Lambda function. It is still best to use Lambda Aliases instead of an Application Load Balancer.

**Using stage variables in your Lambda function** is incorrect because stage variables are primarily used in API Gateway and not in Lambda. Although this solution may work, you are still required to create an API Gateway and create a stage variable that will point to the new and old versions of the Lambda function. This entails extra configuration, compared with just doing traffic shifting in Lambda.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/aliases-intro.html>

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-traffic-shifting-using-aliases.html>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **57. Question**

You are developing a Lambda function which processes event notifications from Amazon S3. It is expected that the function will have:

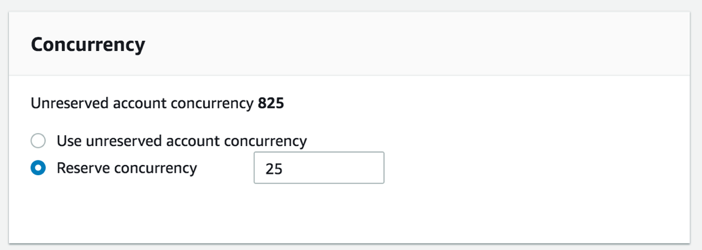
* + 50 requests per second
  + 100 seconds to complete each request

What should you do to prevent any issues when the function has been deployed and becomes operational?

* + No additional action needed since Lambda will automatically scale based on the incoming requests.
  + **Request for AWS to increase the limit of your concurrent executions.**
  + Implement exponential backoff in your application.
  + Increase the concurrency limit of the function.

**Correct**

Concurrent executions refers to the number of executions of your function code that are happening at any given time. You can estimate the concurrent execution count, but the concurrent execution count will differ depending on whether or not your Lambda function is processing events from a poll-based event source.



If you create a Lambda function to process events from event sources that aren’t poll-based (for example, Lambda can process every event from other sources, like Amazon S3 or API Gateway), each published event is a unit of work, in parallel, up to your account limits. Therefore, the number of invocations these event sources make influences the concurrency.

You can use this formula to estimate the capacity used by your function:

**concurrent executions = (invocations per second) x (average execution duration in seconds)**

For example, consider a Lambda function that processes Amazon S3 events. Suppose that the Lambda function takes on average three seconds and Amazon S3 publishes 10 events per second. Then, you will have 30 concurrent executions of your Lambda function. See the calculation shown below to visualize the process:

= **(10 events per second) x (3 seconds average execution duration)**  
=  **30 concurrent executions**

In this scenario, it is expected that the Lambda function takes an average of 100 seconds for every execution with 50 requests per second. Using the formula above, the function will have 5,000 concurrent executions.

**= (50 events per second) x (100 seconds average execution duration)  
=  5,000 concurrent executions**

AWS Lambda dynamically scales function execution in response to increased traffic, up to your [concurrency limit](https://docs.aws.amazon.com/lambda/latest/dg/limits.html). Under sustained load, your function’s concurrency bursts to an initial level between 500 and 3000 concurrent executions that varies per region. After the initial burst, the function’s capacity increases by an additional 500 concurrent executions each minute until either the load is accommodated, or the total concurrency of all functions in the region hits the limit.

By default, AWS Lambda limits the total concurrent executions across all functions within a given region to 1000. This limit can be raised by requesting for AWS to increase the limit of the concurrent executions of your account.

Since the expected concurrent executions of the Lambda function will exceed the default concurrency limit, the best thing to do here is to **request for AWS to increase the limit of your concurrent executions.**

Choosing to do **no additional action since Lambda will automatically scale based on the incoming requests** is incorrect because the dynamic scaling of AWS Lambda has its limits. Because the value of the expected concurrency executions has exceeded the default limit, it is best to contact AWS to increase the concurrent executions of your account to prevent any throttling issues when the function has been deployed and becomes operational.

**Implementing an exponential backoff in your application**is incorrect because this doesn’t address the concurrency issue of your Lambda function. This will just configure your application to have progressively longer waits between API call retries for consecutive error responses.

**Increasing the concurrency limit of the function** is incorrect because, by default, you can only set the limit as high as 900 per function, which is quite insufficient to handle the expected 5,000 concurrency executions. To properly provide the required capacity needed by the function, you have to request for AWS to increase the concurrency limit of your account.

**References:**

<https://docs.aws.amazon.com/lambda/latest/dg/running-lambda-code.html>

<https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html#function-configuration>

**Check out this AWS Lambda Cheat Sheet:**

<https://tutorialsdojo.com/aws-lambda/>

1. **58. Question**

A company has a microservices application that must be integrated with API Gateway. The developer must configure custom data mapping between the API Gateway and the microservices.

In addition, the developer must specify how the incoming request data is mapped to the integration request and how the resulting integration response data is mapped to the method response.

Which of the following integration types is the MOST suitable one to use in API Gateway to meet this requirement?

* + HTTP\_PROXY
  + **HTTP**
  + AWS\_PROXY
  + AWS

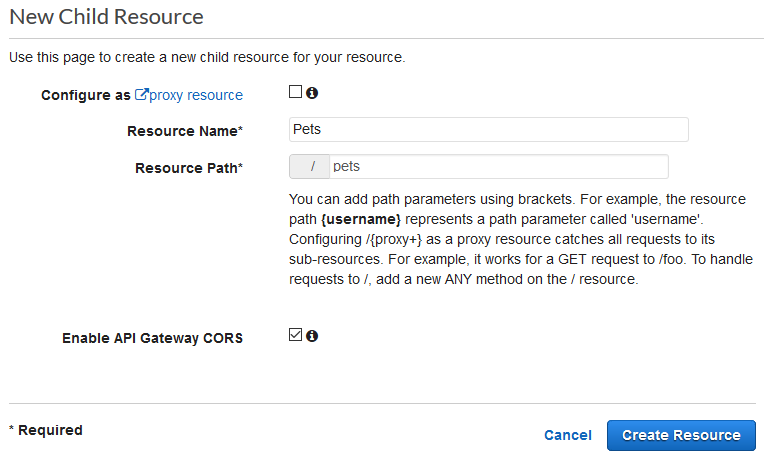
**Correct**

You can integrate an API method in your API Gateway with a custom HTTP endpoint of your application in two ways:

 – HTTP proxy integration

 – HTTP custom integration

In your API Gateway console, you can define the type of HTTP integration of your resource by toggling the “**Configure as proxy resource**” checkbox.



With proxy integration, the setup is simple. You only need to set the HTTP method and the HTTP endpoint URI, according to the backend requirements, if you are not concerned with content encoding or caching.

With custom integration, setup is more involved. In addition to the proxy integration setup steps, you need to specify how the incoming request data is mapped to the integration request and how the resulting integration response data is mapped to the method response. API Gateway supports the following endpoint ports: 80, 443 and 1024-65535.

Programmatically, you choose an integration type by setting the [type](https://docs.aws.amazon.com/apigateway/api-reference/resource/integration/#type) property on the [Integration](https://docs.aws.amazon.com/apigateway/api-reference/resource/integration/) resource. For the Lambda proxy integration, the value is **AWS\_PROXY**. For the Lambda custom integration and all other AWS integrations, it is **AWS**. For the HTTP proxy integration and HTTP integration, the value is **HTTP\_PROXY** and **HTTP**, respectively. For the mock integration, the type value is **MOCK**.

Since the integration type that is being described in the scenario fits the definition of an *HTTP custom integration*, the correct answer in this scenario is to use the **HTTP**integration type.

Hence, the correct answer is: **HTTP.**

**AWS**is incorrect because this type is primarily used for Lambda custom integration. Since the scenario does not specify that the microservices are Lambda functions, the HTTP integration type is the most flexible and suitable for such a scenario.

**AWS\_PROXY** is incorrect because this type is primarily used for Lambda proxy integration. The scenario didn’t mention that it uses a serverless application or Lambda.

**HTTP\_PROXY** is incorrect because this type is only used for HTTP proxy integration where you don’t need to do data mapping for your request and response data.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/setup-http-integrations.html>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-integration-types.html>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **59. Question**

A company selling smart security cameras uses an S3 bucket behind a CloudFront web distribution to store its static content, which it shares with customers worldwide. The company has recently released a new firmware update intended only for its premium customers, and unauthorized access should be denied with a user authentication process that has minimal latency.

How can a developer refactor the current setup to achieve this requirement with the MOST efficient solution?

* + Use Signed URLs and Signed Cookies in CloudFront to distribute the firmware update file.
  + Use the AWS Serverless Application Model (AWS SAM) and Amazon Cognito to authenticate the premium customers.
  + Restrict access to the S3 bucket only to premium customers using an Origin Access Control (OAC).
  + **Use Lambda@Edge and Amazon Cognito to authenticate and authorize premium customers to download the firmware update.**

**Correct**

Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) is a feature of Amazon CloudFront that lets you run code closer to users of your application, which improves performance and reduces latency. With Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/), you don’t have to provision or manage infrastructure in multiple locations around the world. You pay only for the compute time you consume – there is no charge when your code is not running.

With Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/), you can enrich your web applications by making them globally distributed and improving their performance — all with zero server administration. Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) runs your code in response to events generated by the Amazon CloudFront content delivery network (CDN). Just upload your code to AWS Lambda, which takes care of everything required to run and scale your code with high availability at an AWS location closest to your end user.



You can use Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) to help authenticate and authorize users for the premium pay-wall content on your website, filtering out unauthorized requests before they reach your origin infrastructure. For example, you can trigger a Lambda function to authorize each viewer request by calling authentication and user management service such as Amazon Cognito.

Hence, the correct answer is: **Use Lambda**[**@Edge**](https://portal.tutorialsdojo.com/members/edge/)**and Amazon Cognito to authenticate and authorize premium customers to download the firmware update.**

The option that says: **Use the AWS Serverless Application Model (AWS SAM) and Amazon Cognito to authenticate the premium customers** is incorrect because AWS SAM is just an open-source framework that you can use to build serverless applications on AWS. In this scenario, you have to integrate your CloudFront web distribution with Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/), and you can do this without using AWS SAM.

The option that says: **Restrict access to the S3 bucket only to premium customers by using an Origin Access Control (OAC)**is incorrect because OAC is primarily used to prevent your users from viewing your S3 files by simply using the direct S3 URL.

The option that says: **Use Signed URLs and Signed Cookies in CloudFront to distribute the firmware update file** is incorrect. Although this solution provides a way to authenticate the premium users for the private content, the process of authentication has a significant latency in comparison to the Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/) solution. In this option, you have to refactor your application (which is deployed to a specific AWS region) to either create and distribute signed URLs to authenticated users or to send Set-Cookie headers that set signed cookies on the viewers for authenticated users. This will cause the latency, which could have been improved if the authentication logic resides on CloudFront edge locations using Lambda[@Edge](https://portal.tutorialsdojo.com/members/edge/).

**References:**

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-at-the-edge.html>

<https://aws.amazon.com/lambda/edge/>

**Check out these Amazon CloudFront and AWS Lambda Cheat Sheets:**

<https://tutorialsdojo.com/amazon-cloudfront/>

<https://tutorialsdojo.com/aws-lambda/>

1. **60. Question**

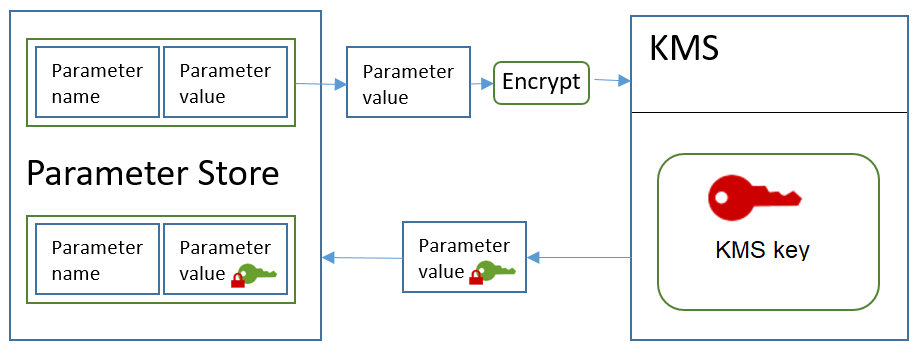
A serverless application is composed of several Lambda functions which reads data from RDS. These functions must share the same connection string that should be encrypted to improve data security.

Which of the following is the MOST secure way to meet the above requirement?

* + Use AWS Lambda environment variables encrypted with CloudHSM.
  + Create an IAM Execution Role that has access to RDS and attach it to the Lambda functions.
  + Use AWS Lambda environment variables encrypted with KMS which will be shared by the Lambda functions.
  + **Create a Secure String Parameter using the AWS Systems Manager Parameter Store.**

**Correct**

**AWS Systems Manager Parameter Store** provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, and license codes as parameter values. You can store values as plain text or encrypted data. You can then reference values by using the unique name that you specified when you created the parameter.



Parameter Store offers the following benefits and features:

– Use a secure, scalable, hosted secrets management service (No servers to manage).

– Improve your security posture by separating your data from your code.

– Store configuration data and secure strings in hierarchies and track versions.

– Control and audit access at granular levels.

– Configure change notifications and trigger automated actions.

– Tag parameters individually, and then secure access from different levels, including operational, parameter, Amazon EC2 tag, or path levels.

– Reference AWS Secrets Manager secrets by using Parameter Store parameters.

Hence, **creating a Secure String Parameter using the AWS Systems Manager Parameter Store** is the correct solution for this scenario.

The option that says: **Use AWS Lambda environment variables encrypted with KMS which will be shared by the Lambda functions** is incorrect. Even though the credentials will be encrypted, these environment variables will only be used by an individual Lambda function, and cannot be shared.

The option that says: **Create an IAM Execution Role that has access to RDS and attach it to the Lambda functions** is incorrect because this solution will not encrypt the database credentials for RDS.

The option that says:**Use AWS Lambda environment variables encrypted with CloudHSM** is incorrect because Lambda primarily uses KMS for encryption and not CloudHSM.

**References:**

<https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-paramstore.html>

<https://aws.amazon.com/blogs/compute/sharing-secrets-with-aws-lambda-using-aws-systems-manager-parameter-store/>

**Check out this AWS Systems Manager Cheat Sheet:**

<https://tutorialsdojo.com/aws-systems-manager/>

1. **61. Question**

You are building a distributed system using KMS where you need to encrypt data at a later time. An API must be called that returns only the encrypted copy of the data key which you will use for encryption. After an hour, you will decrypt the data key by calling the Decrypt API then using the returned plaintext data key to finally encrypt the data.

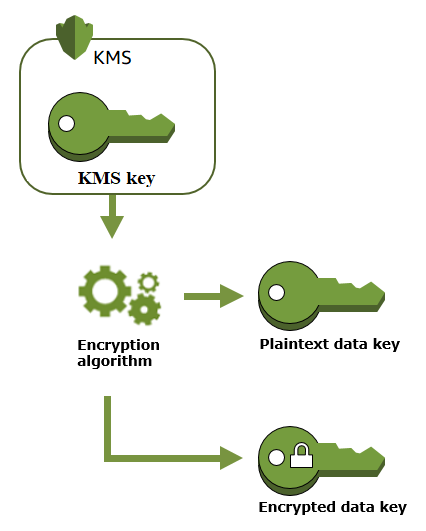
Which is the MOST suitable KMS API that the system should use to securely implement the requirements described above?

* + GenerateRandom
  + **GenerateDataKey**
  + **GenerateDataKeyWithoutPlaintext**
  + Encrypt

**Incorrect**

The GenerateDataKeyWithoutPlaintext API generates a unique data key. This operation returns a data key that is encrypted under a KMS Key that you specify. GenerateDataKeyWithoutPlaintext is identical to GenerateDataKey except that it returns only the encrypted copy of the data key.

Like GenerateDataKey, GenerateDataKeyWithoutPlaintext returns a unique data key for each request. The bytes in the key are not related to the caller or KMS key that is used to encrypt the data key.



This operation is useful for systems that need to encrypt data at some point, but not immediately. When you need to encrypt the data, you call the [Decrypt](https://docs.aws.amazon.com/kms/latest/APIReference/API_Decrypt.html) operation on the encrypted copy of the key.

It’s also useful in distributed systems with different levels of trust. For example, you might store encrypted data in containers. One component of your system creates new containers and stores an encrypted data key with each container. Then, a different component puts the data into the containers. That component first decrypts the data key, uses the plaintext data key to encrypt data, puts the encrypted data into the container, and then destroys the plaintext data key. In this system, the component that creates the containers never sees the plaintext data key.

Hence, the correct answer is: **GenerateDataKeyWithoutPlaintext**

**GenerateDataKey** is incorrect because this operation also returns a plaintext copy of the data key along with the copy of the encrypted data key under a KMS key that you specified. Take note that the scenario explicitly mentioned that the API must return **only** the encrypted copy of the data key which will be used later for encryption. Although this API can be used in this scenario, it is not recommended since the actual encryption process of the data happens at a later time and not in real-time.

**Encrypt** is incorrect because this just encrypts plaintext into ciphertext by using a KMS key. This is primarily used to move encrypted data from one AWS region to another.

**GenerateRandom** is incorrect because this just returns a random byte string that is cryptographically secure. This is not relevant in this scenario, as you have to use the *GenerateDataKeyWithoutPlaintext*API to properly implement the requirement.

**References:**

<https://docs.aws.amazon.com/kms/latest/APIReference/API_GenerateDataKeyWithoutPlaintext.html>

<https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#data-keys>

**Check out this AWS KMS Cheat Sheet:**

<https://tutorialsdojo.com/aws-key-management-service-aws-kms/>

1. **62. Question**

A developer is creating an analytics REST API service that is powered by API Gateway. Analysts from a separate AWS account must interact with the service through an IAM role. The IAM role already has a policy that grants permission to invoke the API.

What else should the developer do to meet the requirement without too much overhead?

* + Create a Cognito User Pool authorizer. Add the IAM role to the user pool. Authenticate the requester’s identity using Cognito. Ask the analysts to pass the token returned by Cognito in their request headers.
  + Create a Lambda function authorizer for the API. In the Lambda function, write a logic that verifies the requester’s identity by extracting the information from the context object.
  + Create an API Key for the API. Attach a resource policy to the API that grants permission to the specified IAM role to invoke the GetAPIKeys action.
  + **Set AWS\_IAM as the method authorization type for the API. Attach a resource policy to the API that grants permission to the specified IAM role to invoke the execute-api:Invoke action.**

**Correct**

By using AWS\_IAM as the method authorization type, it ensures that the API can only be accessed by IAM identities such as IAM users or IAM roles. Attaching a resource policy to the API that grants permission to the specified IAM role to invoke the execute-api:Invoke action allows the specified IAM role to make authorized requests to the API while denying access to any other unauthorized users or roles.

**{  
"Version": "2012-10-17",  
"Statement": [  
{  
"Effect": "Allow",  
"Principal": {  
"AWS": [  
"arn:aws:iam::account-id:role/Analyst"  
]  
},  
"Action": "execute-api:Invoke",  
"Resource": [  
"execute-api:/stage/GET/reports"  
]  
}  
]  
}**

This combination of method authorization and resource policy provides an additional layer of security for the API.

Hence, the correct answer in this scenario is to **Set AWS\_IAM as the method authorization type for the API. Attach a resource policy to the API that grants permission to the specified IAM role to invoke the execute-api:Invoke action.**

The option that says: **Create an API Key for the API. Attach a resource policy to the API that grants permission to the specified IAM role to invoke the GetAPIKeys action**is incorrect API Keys are just a way of identifying the calling parties that you trust, but they are not intended to be used to grant permissions to an IAM role.

The option that says: **Create a Lambda function authorizer for the API. In the Lambda function, write a logic that verifies the requester’s identity by extracting the information from the context object** is incorrect. While this may be possible, Lambda function authorizer is more suitable for custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML. Additionally, this approach requires you to write, test, and maintain custom authentication and authorization code, which can be complex and time-consuming.

The option that says: **Create a Cognito User Pool authorizer. Add the IAM role to the user pool. Authenticate the requester’s identity using Cognito. Ask the analysts to pass the token returned by Cognito in their request headers**is incorrect. Adding a Cognito User Pool authorizer is unnecessary since the API will be accessed through an IAM role.

**References:**

<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-resource-policies-examples.html#apigateway-resource-policies-cross-account-example>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-authorization-flow.html>

**Check out this AWS API Gateway Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

1. **63. Question**

A developer is building a web application which requires a multithreaded event-based key/value cache store that will cache result sets from database calls. You need to run large nodes with multiple cores for your cache layer and it should scale up or down as the demand on your system increases and decreases.

Which of the following is the MOST suitable service that you should use?

* + AWS Greengrass
  + **Amazon ElastiCache for Redis**
  + Amazon CloudFront
  + **Amazon ElastiCache for Memcached**

**Incorrect**

Redis and Memcached are popular, open-source, in-memory data stores. Although they are both easy to use and offer high performance, there are important differences to consider when choosing an engine. Memcached is designed for simplicity while Redis offers a rich set of features that make it effective for a wide range of use cases.

In this scenario, Redis can provide a much more durable and powerful cache layer to the prototype distributed system, however, you should take note of one keyword in the requirement: **multithreaded**. In terms of commands execution, Redis is mostly a single-threaded server. It is not designed to benefit from multiple CPU cores unlike Memcached, however, you can launch several Redis instances to scale out on several cores if needed.

Memcached is a more suitable choice since the scenario specifies that the system will run large nodes with multiple cores or threads which Memcached can adequately provide.

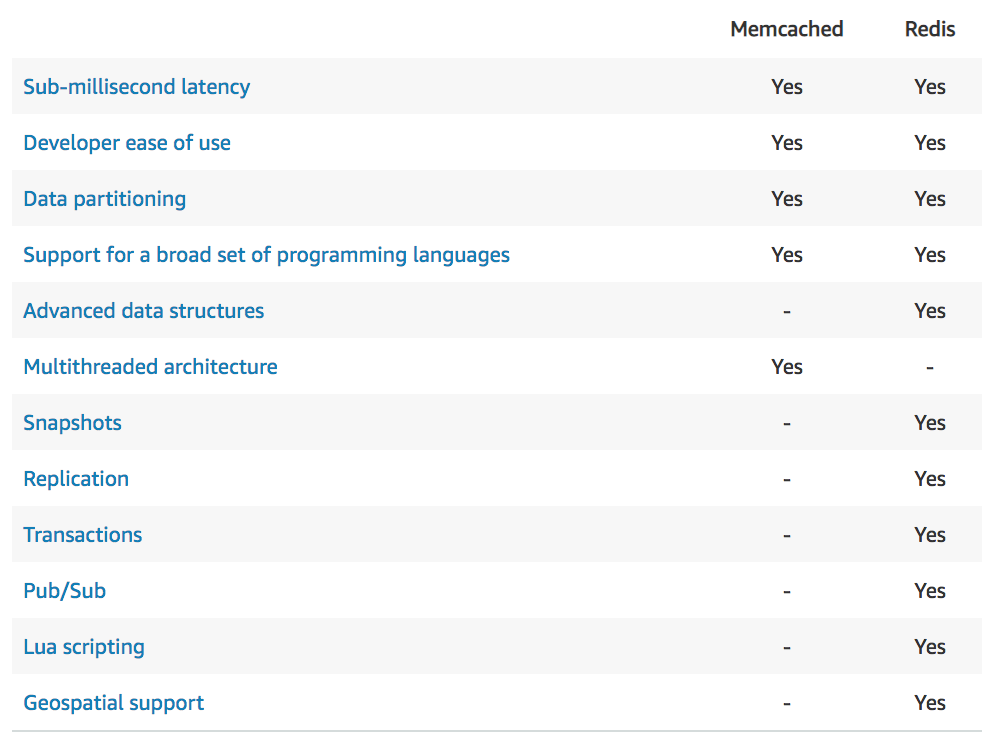
**You can choose Memcached over Redis if you have the following requirements:**

– You need the simplest model possible.

– You need to run large nodes with multiple cores or threads.

– You need the ability to scale out and in, adding and removing nodes as demand on your system increases and decreases.

– You need to cache objects, such as a database.



This is why the most suitable answer to this scenario is **Amazon ElastiCache for Memcached.**

**Amazon ElastiCache for Redis** is incorrect because it does not totally support a multithreaded architecture, unlike Memcached. Although Redis has more features compared with Memcached, the scenario requires that the cache layer is multithreaded. This is why Memcached is a more suitable cache engine to choose from instead of Redis.

**Amazon CloudFront** is incorrect because it is primarily used as a Content Delivery Network (CDN) service which delivers your entire website, including dynamic, static, streaming, and interactive content using a global network of edge locations.

**AWS IoT Greengrass** is incorrect because this service is primarily used to enable connected devices to run AWS Lambda functions, execute predictions based on machine learning models, keep device data in sync, and communicate with other devices securely even without an Internet connection. Hence, this is not a suitable option for this scenario.

**References:**

<https://aws.amazon.com/elasticache/redis-vs-memcached>

<https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/SelectEngine.html>

<https://aws.amazon.com/caching/aws-caching/>

**Redis (cluster mode enabled vs disabled) vs Memcached:**

<https://tutorialsdojo.com/redis-cluster-mode-enabled-vs-disabled-vs-memcached/>

**Comparison of AWS Services Cheat Sheets:**

<https://tutorialsdojo.com/comparison-of-aws-services-for-udemy-students/>

1. **64. Question**

A serverless application composed of Lambda, API Gateway, and DynamoDB has been running without issues for quite some time. As part of the IT compliance of the company, a developer was instructed to ensure that all of the new changes made to the items in DynamoDB are recorded and stored in another DynamoDB table in another region.

In this scenario, which of the following is the MOST ideal way to comply with the requirements?

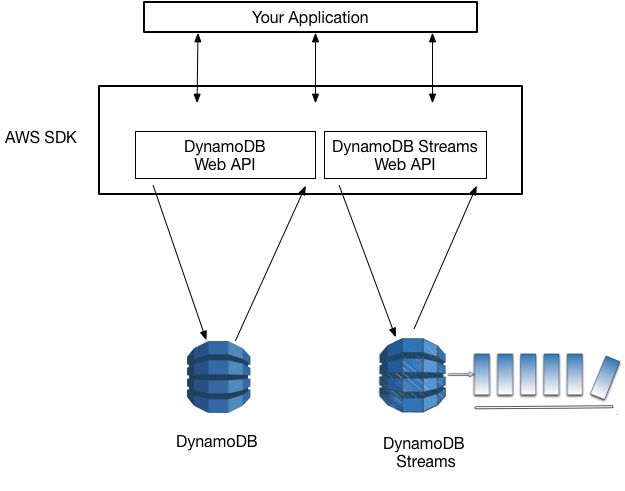
* + Set up DynamoDB Accelerator
  + Enable DynamoDB Point-in-Time Recovery to automatically sync the two tables.
  + **Enable DynamoDB Streams and configure a Lambda function to process and save new changes to the other table.**
  + Create an Amazon EventBridge (Amazon CloudWatch Events) rule that tracks table-level events in DynamoDB. Set a Lambda function as a rule target to process and save new changes to the other table.

**Correct**

**DynamoDB Streams** enables solutions such as these, and many others. DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table, and stores this information in a log for up to 24 hours. Applications can access this log and view the data items as they appeared before and after they were modified, in near real-time.

A *DynamoDB stream* is an ordered flow of information about changes to items in an Amazon DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.

Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attribute(s) of the items that were modified. A *stream record*contains information about a data modification to a single item in a DynamoDB table. You can configure the stream so that the stream records capture additional information, such as the “before” and “after” images of modified items.



Hence, the correct answer is: **Enable DynamoDB Streams and configure a Lambda function to process and save new changes to the other table.**

The option that says: **Set up DynamoDB Accelerator** is incorrect because the DynamoDB Accelerator (DAX) feature simply takes the performance of the DynamoDB table to the next level with response times in microseconds for millions of requests per second for read-heavy workloads. You have to use DynamoDB Streams instead.

The option that says: **Enable DynamoDB Point-in-Time Recovery to automatically sync the two tables** is incorrect because this feature just helps protect your DynamoDB tables from accidental write or delete operations.

The option that says: **Create an Amazon EventBridge (Amazon CloudWatch Events) rule that tracks table-level events in DynamoDB. Set a Lambda function as a rule target to process and save new changes to the other table** is incorrect. An Amazon EventBridge (Amazon CloudWatch Events) rule is not capable of detecting table-level events from Amazon DynamoDB.

**References:**

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.CrossRegionRepl.html>

**Check out this Amazon DynamoDB Cheat Sheet:**

<https://tutorialsdojo.com/amazon-dynamodb/>

**Check out this Amazon API Gateway Cheat Sheet:**

<https://tutorialsdojo.com/amazon-api-gateway/>

**AWS Lambda Integration with Amazon DynamoDB Streams:**

<https://tutorialsdojo.com/aws-lambda-integration-with-amazon-dynamodb-streams/>

1. **65. Question**

A website is hosted in an Auto Scaling group of EC2 instances behind an Application Load Balancer. It also uses CloudFront with a default domain name to distribute its static assets and dynamic contents. However, the website has a poor search ranking as it doesn’t use a secure HTTPS/SSL on its site.

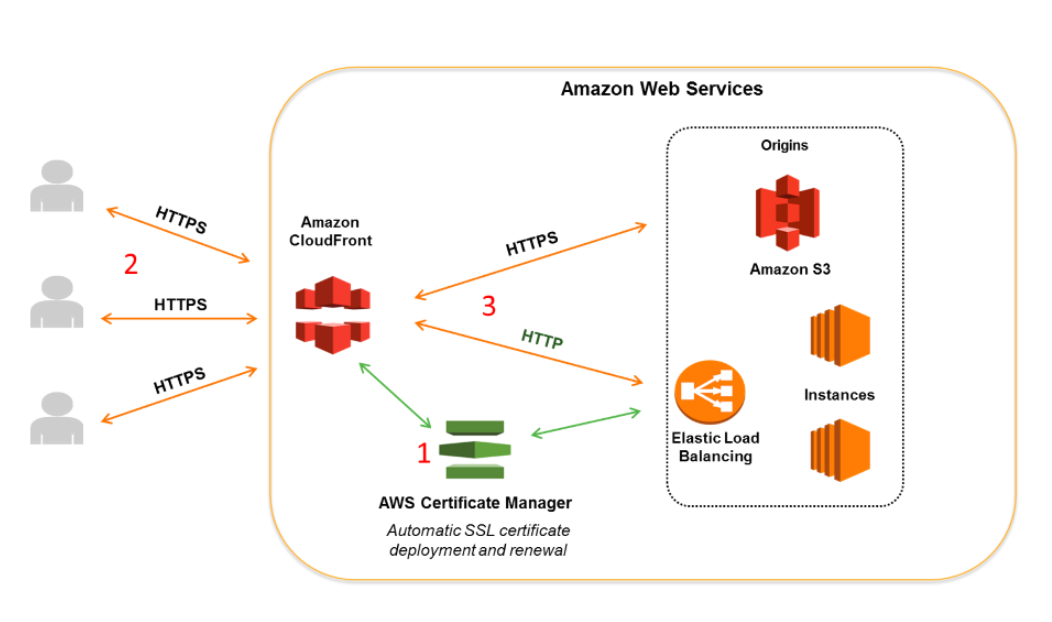
Which are the possible solutions that the developer can implement in order to set up HTTPS communication between the viewers and CloudFront? (Select TWO.)

* + Use a self-signed SSL/TLS certificate in the ALB which is stored in a private S3 bucket.
  + Use a self-signed certificate in the ALB.
  + **Set the Viewer Protocol Policy to use HTTPS Only.**
  + Configure the ALB to use its default SSL/TLS certificate.
  + **Set the Viewer Protocol Policy to use Redirect HTTP to HTTPS.**

**Correct**

You can configure one or more cache behaviors in your CloudFront distribution to require HTTPS for communication between viewers and CloudFront. You also can configure one or more cache behaviors to allow both HTTP and HTTPS so that CloudFront requires HTTPS for some objects but not for others.

If you’re using the domain name that CloudFront assigned to your distribution, such as dtut0ria1sd0jo.cloudfront.net, you can change the Viewer Protocol Policy setting for one or more cache behaviors to require HTTPS communication by setting it as either Redirect HTTP to HTTPS, or HTTPS Only.



If your origin is an Elastic Load Balancing load balancer, you can use a certificate provided by AWS Certificate Manager (ACM). You can also use a certificate that is signed by a trusted third-party certificate authority and imported into ACM. Note that you can’t use a self-signed certificate for HTTPS communication between CloudFront and your origin.

Hence, **setting the Viewer Protocol Policy to use Redirect HTTP to HTTPS** **and setting the Viewer Protocol Policy to use HTTPS Only** are the correct answers in this scenario.

**Using a self-signed SSL/TLS certificate in the ALB which is stored in a private S3 bucket** is incorrect because you don’t need to add an SSL certificate if you only require HTTPS for communication between the viewers and CloudFront. You should only do this if you require HTTPS between your origin and CloudFront. In addition, you can’t use a self-signed certificate in this scenario even though it is stored in a private S3 bucket. You need to use either a certificate from ACM or a third-party certificate.

**Configuring the ALB to use its default SSL/TLS certificate** is incorrect because there is no default SSL certificate in ELB, unlike what we have in CloudFront.

**Using a self-signed certificate in the ALB** is incorrect because adding an SSL certificate in the ELB is not required. Moreover, you can’t use a self-signed certificate in this scenario.

**References:**

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-cloudfront-to-custom-origin.html#using-https-cloudfront-to-origin-certificate>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-viewers-to-cloudfront.html>

**Check out this Amazon CloudFront Cheat Sheet:**

<https://tutorialsdojo.com/amazon-cloudfront/>