|  | Question | Points | Correct | Incorrect | Time (hh:mm:ss) | Results |
| --- | --- | --- | --- | --- | --- | --- |
| Category: **CDA - Deployment** | | | | | | |
| 1 | A developer has just finished writing a serverless application using AWS SAM (Serverless Application Model) on his local machine. He has a SAM template ready and the corresponding Lambda function code in a directory. The developer now wants to deploy this application to AWS.  Which combination of steps should the developer follow to successfully deploy the SAM application? (Select THREE)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:27 |  |
| * Deploy the SAM template from AWS CodeCommit. * **Build the SAM template in the local environment** * Build the SAM template using the AWS SDK for AWS CodeDeploy. * **Deploy the SAM template from an Amazon S3 bucket.** * **Package the SAM application for deployment.** * Build the SAM template in an Amazon EC2 instance.   **AWS SAM** uses AWS CloudFormation as the underlying deployment mechanism. You can deploy your application by using AWS SAM command line interface (CLI) commands. You can also use other AWS services that integrate with AWS SAM to automate your deployments.    The typical AWS SAM deployment workflow starts with the sam build command, which compiles source code and readies deployment artifacts. Once built for deployment, the SAM template and the associated artifacts need to be stored in an S3 bucket. The sam deploy command takes care of this by first uploading the CloudFormation template to the S3 bucket. Though historically, the sam package command was used for this purpose, it's become somewhat legacy, as sam deploy , now implicitly handles the packaging. Once the template is in the S3 bucket, AWS CloudFormation references it to create or update the defined resources.  Hence, the correct answers are:  **- Build the SAM template in the local environment**  **- Package the SAM application for deployment.**  **- Deploy the SAM template from an Amazon S3 bucket.**  The option that says: **Deploy the SAM template from AWS CodeCommit**is incorrect. AWS CodeCommit is simply a version control service. While you can store SAM templates and code in CodeCommit, the deployment process with SAM requires the template and artifacts to be uploaded to an S3 bucket.  The option that says: **Build the SAM template using the AWS SDK for AWS CodeDeploy**is incorrect. The AWS SDK for CodeDeploy is used for management operations of the CodeDeploy service, not for building SAM templates. Building the SAM application is a separate process, typically done using the SAM CLI.  The option that says: **Build the SAM template in an Amazon EC2 instance**is incorrect. This option is unnecessary. While you can technically build on an EC2 instance, it's not a requirement for SAM deployment. In the scenario, there's no condition that warrants the use of an EC2 instance.    **References:**  <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-deploying.html>  <https://docs.aws.amazon.com/serverlessrepo/latest/devguide/what-is-serverlessrepo.html>    **Check out this AWS SAM Cheat Sheet:**  <https://tutorialsdojo.com/aws-serverless-application-model-sam/> | | | | | | |
| 2 | An Elastic Beanstalk application becomes inaccessible for several minutes whenever a failed deployment is rolled back. A developer should recommend a strategy that will have the least impact on the application's availability if the deployment fails. Teams must be able to revert changes quickly as well.  Which deployment method should the developer suggest?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:28 |  |
| * Rolling with Additional Batches * **Blue/Green** * Rolling * All at Once   Because **AWS Elastic Beanstalk** performs an in-place update when you update your application versions, your application can become unavailable to users for a short period of time. You can avoid this downtime by performing a blue/green deployment, where you deploy the new version to a separate environment and then swap CNAMEs of the two environments to redirect traffic to the new version instantly.  Blue/green deployments require that your environment runs independently of your production database if your application uses one. If your environment has an Amazon RDS DB instance attached to it, the data will not transfer over to your second environment and will be lost if you terminate the original environment.    In Elastic Beanstalk, you can choose from a variety of deployment methods:  **All at once** – Deploy the new version to all instances simultaneously. All instances in your environment are out of service for a short time while the deployment occurs.  **Rolling** – Deploy the new version in batches. Each batch is taken out of service during the deployment phase, reducing your environment's capacity by the number of instances in a batch.  **Rolling with additional batch** – Deploy the new version in batches, but first launch a new batch of instances to ensure full capacity during the deployment process.  **Immutable** – Deploy the new version to a fresh group of instances by performing an immutable update.  **Traffic splitting**- Percentage of client traffic routed to new version temporarily impacted  **Blue/Green** - Deploy the new version to a separate environment, and then swap CNAMEs of the two environments to redirect traffic to the new version instantly.  The scenario is asking for the least impact on the application’s availability if a deployment fails. Naturally, we’d want to roll back to the last working version if a deployment does not succeed. The rollback process for a Blue/green deployment is the fastest since all you have to do is switch back to the working environment’s URL.  Hence, the correct answer is **Blue/Green**.  **All at once** is incorrect because this method deploys the new version to all instances simultaneously, which causes your instances to be out of service for a short time while the deployment occurs. This is also the case when you revert a failed deployment. In short, All at once has the MOST impact on your application's availability in case the deployment fails.  **Rolling with additional batch** is incorrect. Although this method ensures full capacity during deployment, its rollback process is quite slow because the deployment is done on fresh instances alongside the existing ones.  **Rolling**is incorrect. With Rolling, your environment's capacity to serve traffic is reduced by the number of instances the new version is being rolled out to, which may impact the availability of your application.    **References:**  <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rolling-version-deploy.html>  <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environmentmgmt-updates-immutable.html>    **Check out this AWS Elastic Beanstalk Cheat Sheet:**  <https://tutorialsdojo.com/aws-elastic-beanstalk/> | | | | | | |
| 3 | A developer is writing a CloudFormation template which will be used to deploy a simple Lambda function to AWS. The function to be deployed is made in Python with just 3 lines of codes which can be written inline in the template.  Which parameter of the AWS::Lambda::Function resource should the developer use to place the Python code in the template?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:18 |  |
| * Handler * Code * **ZipFile** * CodeUri   To create a Lambda function you first create a Lambda function deployment package, a .zip or .jar file consisting of your code and any dependencies. When creating the zip, include only the code and its dependencies, not the containing folder. You will then need to set the appropriate security permissions for the zip package.  If you are using a CloudFormation template, you can configure the AWS::Lambda::Function resource which creates a Lambda function. To create a 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.    Under the AWS::Lambda::Function resource, you can use the Code property which contains the deployment package for a Lambda function. For all runtimes, you can specify the location of an object in Amazon S3.  For Node.js and Python functions, you can specify the function code inline in the template. Changes to a deployment package in Amazon S3 are not detected automatically during stack updates. To update the function code, change the object key or version in the template.  Hence, the **ZipFile** parameter to is the correct one to be used in this scenario, which will allow the developer to place the python code inline in the template. If you include your function source inline with this parameter, AWS CloudFormation places it in a file named index and *zips* it to create a deployment package. This is the reason why it is called the "ZipFile" parameter, and not because it accepts zip files.  The **Handler** parameter is incorrect because this is not a valid property of AWS::Lambda::Function resource but of the AWS::Serverless::Function resource in AWS SAM. In addition, this parameter is primarily used to specify the name of the handler, which is just a function in your code that AWS Lambda can invoke when the service executes your code.  The **Code** parameter is incorrect because you should use the ZipFile parameter instead. Take note that the Code property is the parent property of the ZipFile parameter.  The **CodeUri** parameter is incorrect because this is not a valid property of AWS::Lambda::Function resource but of the AWS::Serverless::Function resource in AWS SAM. This parameter accepts the S3 URL of your code and not the actual code itself.    **References:**  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-lambda-function-code.html>  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html>    **Check out this AWS CloudFormation Cheat Sheet:**  <https://tutorialsdojo.com/aws-cloudformation/> | | | | | | |
| 4 | A development team has recently completed building their serverless application. They must zip their code artifacts, upload them to Amazon S3, produce the package template file for deployment, and deploy it to AWS.  Which command is the MOST suitable to use to automate the deployment steps?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:47 |  |
| * sam package * sam publish * aws cloudformation deploy * **sam deploy**   **AWS SAM** uses AWS CloudFormation as the underlying deployment mechanism. You can deploy your application by using AWS SAM command line interface (CLI) commands. You can also use other AWS services that integrate with AWS SAM to automate your deployments.  After you develop and test your serverless application locally, you can package and deploy your application by using the sam deploy command.    The sam deploy command zips your code artifacts, uploads them to Amazon S3, and produces a packaged AWS SAM template file that it uses to deploy your application.  To deploy an application that contains one or more nested applications, you must include the CAPABILITY\_AUTO\_EXPAND capability in the sam deploy command.  Hence, the correct answer is:**sam deploy**  **aws cloudformation deploy** is incorrect. While this command can be used to deploy a CloudFormation stack, it expects that your artifacts are already packaged and uploaded to S3. It doesn't handle the packaging process implicitly.  **sam package**is incorrect. This command simply prepares the serverless application for deployment by zipping artifacts, uploading them to S3, and generating a CloudFormation template with references to the uploaded artifacts in S3. It doesn't deploy the application.  **sam publish** is incorrect because this command publishes an AWS SAM application to the AWS Serverless Application Repository and does not generate the template file. It takes a packaged AWS SAM template and publishes the application to the specified region.    **References:**  <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-deploying.html>  <https://docs.aws.amazon.com/serverlessrepo/latest/devguide/what-is-serverlessrepo.html>    **Check out this AWS SAM Cheat Sheet:**  <https://tutorialsdojo.com/aws-serverless-application-model-sam/> | | | | | | |
| 5 | An aerospace engineering company has recently migrated to AWS for their cloud architecture. They are using CloudFormation and AWS SAM as deployment services for both of their monolithic and serverless applications. There is a new requirement where you have to dynamically install packages, create files, and start services on your EC2 instances upon the deployment of the application stack using CloudFormation.  Which of the following helper scripts should you use in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:08 |  |
| * cfn-get-metadata * cfn-signal * **cfn-init** * cfn-hup   **AWS CloudFormation** provides the following Python helper scripts that you can use to install software and start services on an Amazon EC2 instance that you create as part of your stack:  [cfn-init](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-init.html): Use to retrieve and interpret resource metadata, install packages, create files, and start services.  [cfn-signal](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-signal.html): Use to signal with a CreationPolicy or WaitCondition, so you can synchronize other resources in the stack when the prerequisite resource or application is ready.  [cfn-get-metadata](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-get-metadata.html): Use to retrieve metadata for a resource or path to a specific key.  [cfn-hup](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-hup.html): Use to check for updates to metadata and execute custom hooks when changes are detected.  You call the scripts directly from your template. The scripts work in conjunction with resource metadata that's defined in the same template. The scripts run on the Amazon EC2 instance during the stack creation process. The scripts are not executed by default. You must include calls in your template to execute specific helper scripts.    Hence, **cfn-init** helper script is the correct answer since it interprets the metadata that contains the sources, packages, files, and services. You run the script on the EC2 instance when it is launched. The script is installed by default on Amazon Linux and Windows AMIs.  The **cfn-get-metadata** helper script is incorrect since it is only a wrapper script that retrieves either all metadata that is defined for a resource or path to a specific key or a subtree of the resource metadata, but does not interpret the resource metadata, install packages, create files, and start services.  The **cfn-signal**helper script is incorrect since it does not perform any retrieval and interpretation of resource metadata, installation of packages, creation of files, and starting of services. Instead, it is a wrapper thats signals an AWS CloudFormation WaitCondition for synchronizing other resources in the stack when the application is ready.  The **cfn-hup**helper script is incorrect because this is just a daemon that checks for updates to metadata and executes custom hooks when changes are detected. It does not retrieve and interpret the resource metadata, install packages, create files, and start services unlike cfn-init helper script.    **References:**  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-init.html>  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html>  <https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf>    **Check out this AWS CloudFormation Cheat Sheet:**  <https://tutorialsdojo.com/aws-cloudformation/>    **AWS CloudFormation - Templates, Stacks, Change Sets:** | | | | | | |
| Sub-Total: | | 5 | 5 (100%) | 0 (0%) | 00:04:08 | **100%** |
|  | | | | | | |
| Category: **CDA - Development with AWS Services** | | | | | | |
| 1 | A media company seeks to protect its copyrighted images from unauthorized distribution. They want images uploaded to their Amazon S3 bucket to be automatically watermarked. A developer has already prepared the Lambda function for this image-processing job.  Which option must the developer configure to automatically invoke the function at each upload?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:46 |  |
| * **Set up an Amazon S3 Event Notification to trigger the Lambda function when an ObjectCreated:Put event is detected in the bucket.** * Enable S3 Storage Lens to monitor the bucket and configure the Lambda function to be invoked whenever the metrics indicate a new object creation. * Configure an S3 Lifecycle policy to transition images to the INTELLIGENT\_TIERING storage class. Use S3 Inventory to generate a report of images that weren't watermarked and set up the Lambda function to process the report. * **Use S3 Select to identify unwatermarked images based on metadata and create an EventBridge rule that triggers the Lambda function upon such findings.**   You can use the Amazon S3 Event Notifications feature to receive notifications when certain events happen in your S3 bucket. To enable notifications, add a notification configuration that identifies the events that you want Amazon S3 to publish. Make sure that it also identifies the destinations where you want Amazon S3 to send the notifications.    Amazon S3 can send event notification messages to the following destinations:  - Amazon SQS queue  - AWS Lambda function  - Amazon SNS topic  - Amazon EventBridge  In the given scenario, you can set up a notification for the ObjectCreated:Put event to immediately trigger a Lambda function when an object is uploaded to the S3 bucket.  Hence, the correct answer is:**Set up an Amazon S3 Event Notification to trigger the Lambda function when an ObjectCreated:Put event is detected in the bucket.**  The option that says: **Enable S3 Storage Lens to monitor the bucket and configure the Lambda function to be invoked whenever the metrics indicate a new object creation**is incorrect. S3 Storage Lens just provide visibility into storage usage and activity trends. It does not trigger actions or Lambda functions based on object operations.  The option that says: **Configure an S3 Lifecycle policy to transition images to the INTELLIGENT\_TIERING storage class. Use S3 Inventory to generate a report of images that weren't watermarked and set up the Lambda function to process the report**is incorrect. S3 Lifecycle policies simply manage storage transitions and object expirations, not event-driven actions like invoking Lambda functions upon uploads. Moreover, S3 Inventory just provides object lists and their metadata, but it doesn't automatically invoke Lambda functions upon image uploads.  The option that says: **Use S3 Select to identify unwatermarked images based on metadata and create an EventBridge rule that triggers the Lambda function upon such findings**is incorrect. S3 Select is primarily designed to retrieve specific subsets of data from objects in S3 without retrieving the entire object. It's typically used with text data formats like CSV and JSON, not images. Furthermore, Amazon EventBridge doesn't capture events emitted as a result of S3 Select operations.    **References:**  <https://docs.aws.amazon.com/AmazonS3/latest/userguide/EventNotifications.html>  <https://docs.aws.amazon.com/AmazonS3/latest/userguide/enable-event-notifications.html>    **Check out this Amazon S3 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-s3/>    **Check out this blog about S3 Event Notification:**  <https://tutorialsdojo.com/amazon-s3-event-notifications/> | | | | | | |
| 2 | A developer is launching a Lambda function that requires access to a MySQL RDS instance that is in a private subnet. Which of the following is the MOST secure way to achieve this?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:34 |  |
| * Ensure that the Lambda function has proper IAM permission to access RDS. * Expose an endpoint of your RDS to the Internet using an Elastic IP. * Move your RDS instance to a public subnet. * **Configure the Lambda function to connect to your VPC.**   You can configure a Lambda function to connect to a virtual private cloud (VPC) in your account. Use Amazon Virtual Private Cloud (Amazon VPC) to create a private network for resources such as databases, cache instances, or internal services. Connect your function to the VPC to access private resources during execution.  AWS Lambda runs your function code securely within a VPC by default. However, to enable your Lambda function to access resources inside your private VPC, you must provide additional VPC-specific configuration information that includes VPC subnet IDs and security group IDs. AWS Lambda uses this information to set up elastic network interfaces [(ENIs)](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_ElasticNetworkInterfaces.html) that enable your function to connect securely to other resources within your private VPC.  The following diagram guides you through a decision tree as to whether you should use a VPC (Virtual Private Cloud):    Don't put your Lambda function in a VPC unless you have to. There is no benefit outside of using this to access resources you cannot expose publicly, like a private Amazon Relational Database instance. Services like Amazon OpenSearch Service can be secured over IAM with access policies, so exposing the endpoint publicly is safe and wouldn't require you to run your function in the VPC to secure it.  Hence, **configuring the Lambda function to connect to your VPC** is the correct answer for this scenario.  **Ensuring that the Lambda function has proper IAM permission to access RDS** is incorrect. Even though you grant the necessary IAM permissions to the Lambda function to access RDS, the function would still not be able to connect to RDS since there is no established connection between Lambda and the private subnet of your VPC.  **Exposing an endpoint of your RDS to the Internet using an Elastic IP** is incorrect because this is not the most secure way of granting access to your Lambda function. It will be able to connect to RDS but so will the billions of people on the public Internet.  **Moving your RDS instance to a public subnet** is incorrect because this is an unnecessary change and not a best practice from a security perspective. You only need to configure your Lambda function to your VPC so it can connect to the RDS in the private subnet. If you move your RDS instance to a public subnet, it will introduce a critical security flaw to your entire architecture since your database will become accessible publicly.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html#lambda-vpc>  <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 3 | You are planning to launch a Lambda function integrated with API Gateway. It is required 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.  Which of the following options is the MOST appropriate method use to meet this requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:33 |  |
| * **Lambda custom integration** * **Lambda proxy integration** * HTTP custom integration * HTTP Proxy integration   You choose an API integration type according to the types of integration endpoint you work with and how you want data to pass to and from the integration endpoint. For a Lambda function, you can have two types of integration:   - Lambda proxy integration   - Lambda custom integration  In Lambda proxy integration, the setup is simple. If your API does not require content encoding or caching, you only need to set the integration's HTTP method to POST, the integration endpoint URI to the ARN of the Lambda function invocation action of a specific Lambda function, and the credential to an IAM role with permissions to allow API Gateway to call the Lambda function on your behalf.  In Lambda non-proxy (or custom) integration, in addition to the proxy integration setup steps, you also 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.  For an AWS service action, you have the AWS integration of the non-proxy type only. API Gateway also supports the mock integration, where API Gateway serves as an integration endpoint to respond to a method request.  The Lambda custom integration is a special case of the AWS integration, where the integration endpoint corresponds to the [function-invoking action](https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html)of the Lambda service.    The Lambda custom integration is a type of integration that lets an API expose AWS service actions. In AWS integration, you must configure both the integration request and integration response and set up necessary data mappings from the method request to the integration request, and from the integration response to the method response. To configure your API Gateway with this type of configuration, you have to set the resource with an AWS integration type.  Hence, **Lambda custom integration** is correct as it matches the description depicted in the scenario.  **Lambda proxy integration** is incorrect as this type of integration is the one where you do **not** have to configure both the integration request and integration response.  **HTTP custom integration** is incorrect because this type is only used where 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. Take note that the scenario uses an application hosted in Lambda which is why you have to use Lambda integration instead.  **HTTP proxy integration**is incorrect because the scenario uses an application hosted in Lambda which is why you have to use Lambda integration instead.    **References:**  <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-integration-types.html>  <https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format>    **Check out this Amazon API Gateway Cheat Sheet:**  <https://tutorialsdojo.com/amazon-api-gateway/> | | | | | | |
| 4 | A developer must set up a caching layer in front of the tutorialsdojo database. The developer should come up with a function that ensures cached data is always up-to-date. Stale records in the cache must be automatically deleted as well to prevent the build-up of extra data.  Which pseudocode best represents this caching strategy?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:46 |  |
| * **save\_item(item\_id, item\_value):** * **ttl = 500** * **tutorialsdojo.query("UPDATE Customers SET value = %s WHERE id = %s", item\_value, item\_id)** * **cache.set(item\_id, item\_value, ttl)** * **return 'ok'** * save\_item(item\_id, item\_value): * ttl = 500 * tutorialsdojo.query("SELECT Customers WHERE id = %s", item\_id) * cache.set(item\_id, item\_value, 500) * return 'ok' * save\_item(item\_id, item\_value): * ttl = 500 * cache.set(item\_id, item\_value, 500) * return 'ok' * save\_item(item\_id, item\_value): * tutorialsdojo.query("UPDATE Customers SET value = %s WHERE id = %s", item\_value, item\_id) * cache.delete(item\_id) * return 'ok'   The write-through strategy adds or updates data in the cache whenever data is written to the database. With this strategy, the data in the cache is never stale since the data in the cache is updated every time it is written to the database. This is why the data in the cache is always current.  One of its disadvantages is that, since most data are never read in the cluster, you are actually wasting resources and disk space. This issue can be rectified by simply adding TTL to minimize wasted space.    In the scenario, to implement write-through caching, you must first update the item in the tutorialsdojo database by running an "UPDATE" query, then adds the updated item to the cache with a time-to-live (TTL) value of 500 seconds. This ensures that the item is always up to date in both the cache and the database, and it will expire from the cache after 500 seconds. This way, the cache is not cluttered with stale data, and the performance will not degrade over time.  Hence, the correct answer is the option that says:  **save\_item(item\_id, item\_value):**  **ttl = 500**  **tutorialsdojo.query("UPDATE Customers SET value = %s WHERE id = %s", item\_value, item\_id)**  **cache.set(item\_id, item\_value, ttl)**  **return 'ok**'  The option that says:  **save\_item(item\_id, item\_value):**  **ttl = 500**  **tutorialsdojo.query("SELECT Customers WHERE id = %s", item\_id)**  **cache.set(item\_id, item\_value, 500)**  **return 'ok'**  is an incorrect implementation of write-through caching because the code is not updating the database when a new item is added, or an existing item is updated. Instead, it is making a SELECT query to the database. This means that the primary storage is not being updated with the latest data, and the cache will hold stale data.  The option that says:  **save\_item(item\_id, item\_value):**  **ttl = 500**  **cache.set(item\_id, item\_value, 500)**  **return 'ok'**  does not implement write-through caching because the function only stores the item in the cache and doesn't update the database.  The option that says:  **save\_item(item\_id, item\_value):**  **tutorialsdojo.query("UPDATE Customers SET value = %s WHERE id = %s", item\_value, item\_id)**  **cache.delete(item\_id)**  **return 'ok'**  is an incorrect implementation of write-through caching because it updates the item in the database, but it also deletes the item from the cache. This means that the cache will not have the updated item, and subsequent requests for the same item will have to be retrieved from the primary database, which is slower than retrieving it from the cache.    **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:** | | | | | | |
| 5 | Your application is hosted on an Auto Scaling group of EC2 instances with a DynamoDB database. There were a lot of data discrepancy issues where the changes made by one user were always overwritten by another user. You noticed that this usually happens whenever there are a lot of people updating the same data.  What should you do to solve this problem?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:00 |  |
| * Use DynamoDB global tables and implement a pessimistic locking strategy. * Implement a pessimistic locking strategy in your application source code by designating one property to store the version number in the mapping class for your table. * Use DynamoDB global tables and implement an optimistic locking strategy. * **Implement an optimistic locking strategy in your application source code by designating one property to store the version number in the mapping class for your table.**   **Optimistic locking** is a strategy to ensure that the client-side item that you are updating (or deleting) is the same as the item in DynamoDB. If you use this strategy, then your database writes are protected from being overwritten by the writes of others — and vice-versa. Take note that:   - DynamoDB global tables use a “last writer wins” reconciliation between concurrent updates. If you use Global Tables, last writer policy wins. So in this case, the locking strategy does not work as expected.   - DynamoDBMapper transactional operations do not support optimistic locking.  With optimistic locking, each item has an attribute that acts as a version number. If you retrieve an item from a table, the application records the version number of that item. You can update the item, but only if the version number on the server side has not changed. If there is a version mismatch, it means that someone else has modified the item before you did; the update attempt fails, because you have a stale version of the item. If this happens, you simply try again by retrieving the item and then attempting to update it. Optimistic locking prevents you from accidentally overwriting changes that were made by others; it also prevents others from accidentally overwriting your changes.  Hence, **implementing an optimistic locking strategy in your application source code by designating one property to store the version number in the mapping class for your table** is the correct answer in this scenario.  **Using DynamoDB global tables and implementing a pessimistic locking strategy** is incorrect because you have to use optimistic locking here just as what was explained above.  **Implementing a pessimistic locking strategy in your application source code by designating one property to store the version number in the mapping class for your table** is incorrect because an optimistic locking strategy is a more suitable solution for this scenario. Although the provided steps here are correct, the name of the strategy is wrong.  **Using DynamoDB global tables and implementing an optimistic locking strategy** is incorrect. Although it is correct to use the optimistic locking strategy, the use of DynamoDB global tables is wrong. This uses a *"last writer wins"* reconciliation between concurrent updates. If you use Global Tables, the last writer policy is in effect so in this case, the locking strategy will not work as expected.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithItems.html>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.OptimisticLocking.html>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/>    **Amazon DynamoDB Overview:** | | | | | | |
| 6 | An e-commerce application, which is hosted in an ECS Cluster, contains the connection string of an external database and other sensitive configuration files. Since the application accepts credit card payments, the company has to meet strict security compliance which requires that the database credentials are encrypted and periodically rotated.  Which of the following should you do to comply to the requirements?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:26 |  |
| * Store the database credentials as a secure string parameter in Systems Manager Parameter Store. * Store the database credentials in an encrypted ecs.config configuration file. * **Store the database credentials in AWS Secrets Manager and enable rotation.** * Store the database credentials in an encrypted dockerrun.aws.json configuration file.   **Amazon ECS** enables you to inject sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. This feature is supported by tasks using both the EC2 and Fargate launch types.  Secrets can be exposed to a container in the following ways:  - To inject sensitive data into your containers as environment variables, use the secrets container definition parameter.  - To reference sensitive information in the log configuration of a container, use the secretOptions container definition parameter.    You can configure AWS Secrets Manager to automatically rotate the secret for a secured service or database. Secrets Manager already natively knows how to rotate secrets for supported Amazon RDS databases. However, Secrets Manager also can enable you to rotate secrets for other databases or third-party services. Because each service or database can have a unique way of configuring its secrets, Secrets Manager uses a Lambda function that you can customize to work with whatever database or service that you choose. You customize the Lambda function to implement the service-specific details of how to rotate a secret.  **AWS Secrets Manager**is the correct answer for this scenario because it can provide both the required encryption as well as the ability to periodically rotate the secrets.  **Storing the database credentials as a secure string parameter in Systems Manager Parameter Store** is incorrect. Although this service can encrypt your sensitive database credentials, it doesn't have the capability to periodically rotate your secrets, unlike AWS Secrets Manager.  **Storing the database credentials in an encrypted ecs.config configuration file** is incorrect because this file is primarily used to store the environment variables of the Amazon ECS container agent.  **Storing the database credentials in an encrypted dockerrun.aws.json configuration file** is incorrect because this file is just an Elastic Beanstalk–specific JSON file that describes how to deploy a set of Docker containers as an Elastic Beanstalk application. This is primarily used in a multicontainer Docker environment and it is not suitable for storing sensitive database credentials, which requires periodic rotation.    **References:**  <https://docs.aws.amazon.com/secretsmanager/latest/userguide/rotating-secrets.html>  <https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-paramstore-securestring.html>  <https://aws.amazon.com/systems-manager/faq/>    **Check out these AWS Systems Manager and Secrets Manager Cheat Sheets:**  <https://tutorialsdojo.com/aws-systems-manager/>  <https://tutorialsdojo.com/aws-secrets-manager/> | | | | | | |
| 7 | An application running in an EC2 instance is regularly fetching and processing a lot of data from an S3 bucket which resulted in a significant increase in your operating costs. You want to lower the latency of retrieving data from S3 and bring the operating costs down. To improve the system, you need 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.  Which is the MOST suitable service that will help you accomplish this requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:27 |  |
| * **Athena** * Redshift Spectrum * **S3 Select** * AWS Step Functions   With **Amazon S3 Select**, you can 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. By using Amazon S3 Select to filter this data, you can reduce the amount of data that Amazon S3 transfers, which reduces the cost and latency of retrieving these data.    Amazon S3 Select works on objects stored in CSV, JSON, or Apache Parquet format. It also works with objects that are compressed with GZIP or BZIP2 (for CSV and JSON objects only), and server-side encrypted objects. You can specify the format of the results as either CSV or JSON, and you can determine how the records in the result are delimited.  Hence, the most suitable service which will meet the specified requirements is **S3 Select**.  **Athena** is incorrect. Although it can handle far more complex ad-hoc queries on data in Amazon S3, this entails an increase in operating cost as compared to S3 Select. Moreover, the requirement just requires you to run simple structured query language (SQL) statements to filter the contents of Amazon S3 objects and retrieve just the subset of data that you need. This can be done using S3 Select alone.  **Amazon Redshift Spectrum** is incorrect. Although it provides similar in-query functionality like S3 Select, this service is more suitable for querying your Redshift clusters and not your S3 buckets. The Redshift queries are run on your cluster resources against the local disk. Redshift Spectrum queries run using per-query scale-out resources against data in S3 which can entail additional costs compared to S3 Select.  **Step Functions** is incorrect because this service only lets you coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. It doesn't provide a function to do an in-place query to an S3 bucket.    **References:**  <https://docs.aws.amazon.com/AmazonS3/latest/dev/selecting-content-from-objects.html>  <https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-glacier-select-sql-reference-select.html>    **Check out this Amazon S3 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-s3/> | | | | | | |
| 8 | A developer is planning to add a global secondary index in a DynamoDB table. This will allow the application to query a specific index that can span all of the data in the base table, across all partitions.  Which of the following should the developer consider when using this type of index? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:55 |  |
| * **Queries or scans on this index consume capacity units from the index, not from the base table.** * When you query this index, you can choose either eventual consistency or strong consistency. * Queries or scans on this index consume read capacity units from the base table. * For each partition key value, the total size of all indexed items must be 10 GB or less. * **Queries on this index support eventual consistency only.**   A global secondary index is 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.  To create a table with one or more global secondary indexes, use the CreateTable operation with the GlobalSecondaryIndexes parameter. For maximum query flexibility, you can create up to 20 global secondary indexes (default limit) per table. You must specify one attribute to act as the index partition key; you can optionally specify another attribute for the index sort key. It is not necessary for either of these key attributes to be the same as a key attribute in the table. Global secondary indexes inherit the read/write capacity mode from the base table.    As shown in the above table, the following are the things that the developer should consider when using a global secondary index:   - **Queries or scans on this index consume capacity units from the index, not from the base table.**   - **Queries on this index support eventual consistency only.**  The following options are incorrect because these are about local secondary indexes:   - **When you query this index, you can choose either eventual consistency or strong consistency.**   - **Queries or scans on this index consume read capacity units from the base table**   - **For each partition key value, the total size of all indexed items must be 10 GB or less.**    **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/>    **Comparison of AWS Services Cheat Sheets:**  <https://tutorialsdojo.com/comparison-of-aws-services/> | | | | | | |
| 9 | A developer is creating a new global secondary index on a provisioned mode DynamoDB table. Since the application will store large quantities of data, the write capacity units must be specified for the expected workload on both the base table and its secondary index.  Which of the following should the developer do to avoid any potential request throttling?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:30 |  |
| * **Ensure that the global secondary index's provisioned WCU is equal or greater than the WCU of the base table.** * Ensure that the global secondary index's provisioned RCU is equal or greater than the RCU of the base table. * **Ensure that the global secondary index's provisioned WCU is equal or less than the WCU of the base table.** * Ensure that the global secondary index's provisioned RCU is equal or less than the RCU of the base table.   A **global secondary index (GSI)** is an index with a partition key and a sort key that can be different from those on the base table. It is considered "global" because queries on the index can span all of the data in the base table, across all partitions.  Every global secondary index has its own provisioned throughput settings for read and write activity. Queries or scans on a global secondary index consume capacity units from the index, not from the base table. The same holds true for global secondary index updates due to table writes.  When you create a global secondary index on a provisioned mode table, you must specify read and write capacity units for the expected workload on that index. The provisioned throughput settings of a global secondary index are separate from those of its base table. A Query operation on a global secondary index consumes read capacity units from the index, not the base table. When you put, update, or delete items in a table, the global secondary indexes on that table are also updated; these index updates consume write capacity units from the index, not from the base table.    For example, if you Query a global secondary index and exceed its provisioned read capacity, your request will be throttled. If you perform heavy write activity on the table but a global secondary index on that table has insufficient write capacity, then the write activity on the table will be throttled.  To avoid potential throttling, the provisioned write capacity for a global secondary index should be equal or greater than the write capacity of the base table since new updates will write to both the base table and global secondary index.  Hence, the correct answer in this scenario is to **ensure that the global secondary index's provisioned WCU is equal to or greater than the WCU of the base table**.  **Ensuring that the global secondary index's provisioned WCU is equal or less than the WCU of the base table** is incorrect because it should be the other way around, just as what is mentioned above. The provisioned write capacity for a global secondary index should be equal to or greater than the write capacity of the base table.  **Ensuring that the global secondary index's provisioned RCU is equal to or greater than the RCU of the base table** is incorrect because you have to set the WCU and not the RCU.  **Ensuring that the global secondary index's provisioned RCU is equal or less than the RCU of the base table** is incorrect because this should be WCU and in addition, the global secondary index's provisioned WCU should be set to a value that is equal or greater than the WCU of the base table to prevent request throttling.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html#GSI.ThroughputConsiderations>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.Errors.html#Programming.Errors.MessagesAndCodes>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/> | | | | | | |
| 10 | You are designing the DynamoDB table that will be used by your Node.js application. It will have to handle 10 writes per second and then 20 eventually consistent reads per second where all the items have a size of 2 KB for both operations.  Which of the following are the most optimal WCU and RCU that you should provision to the table?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:39 |  |
| * **10 RCU and 20 WCU** * 20 RCU and 20 WCU * 40 RCU and 20 WCU * 40 RCU and 40 WCU   When you create a new provisioned table in DynamoDB, you must specify its *provisioned throughput capacity*—the amount of read and write activity that the table will be able to support. DynamoDB uses this information to reserve sufficient system resources to meet your throughput requirements.  You can optionally allow DynamoDB auto-scaling to manage your table's throughput capacity. However, you still must provide initial settings for read and write capacity when you create the table. DynamoDB auto scaling uses these initial settings as a starting point and then adjusts them dynamically in response to your application's requirements. You specify throughput requirements in terms of capacity units—the amount of data your application needs to read or write per second. You can modify these settings later, if needed, or enable DynamoDB auto-scaling to modify them automatically.  **1 WCU** can do **1 write per second** for an item up to **1KB.**To get the required WCU, simply multiply the given average item size by the required writes per second. In the scenario, the DynamoDB table is expected to perform 10 writes per second of a 2KB item. Multiplying 10 by 2 gives **20 WCU**.  **1 RCU** can do **1 strongly consistent read** or **2 eventually consistent reads** for an item up to **4KB**.  **To get the RCU with eventually consistent reads, do the following steps:**  **Step #1 Divide the average item size by 4 KB. Round up the result**  Average Item Size = 2 KB  = **2KB/4KB**  **= 0.5 ≈ 1**  **Step #2 Multiply the number of reads per second by the resulting value from Step 1. Divide the product by 2 for eventually consistent reads.**  =**20** reads per second **x** **1**  = **20**RCU  Since the type of read being asked is eventually consistent, we get half of 20, which is 10.  = 20/2 = **10 RCU**  Hence, the correct answer is to provision **10 RCU and 20 WCU** to your DynamoDB table.  The **20 RCU and 20 WCU** setting is incorrect because this would be the result if you use strong consistency reads. Remember that the scenario explicitly said that eventual consistency reads would be used.  The **40 RCU and 20 WCU**is incorrect because 40 RCU is overkill for the required eventual consistency reads. If the scenario was asking for transactional read requests, then this option could have been correct.  The **40 RCU and 40 WCU** setting is incorrect because this would be the result if you chose transactional requests both on your reads and writes. Take note that the scenario didn't say that the database is using DynamoDB Transactions.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Writes>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/>    **Calculating the Required Read and Write Capacity Unit for your DynamoDB Table:**  <https://tutorialsdojo.com/calculating-the-required-read-and-write-capacity-unit-for-your-dynamodb-table/> | | | | | | |
| 11 | The operating cost of a serverless application is quite high and you are instructed to look for ways to lower the costs. As part of its processing, a Lambda function sends 320 strongly consistent read requests per second to a DynamoDB table which has a provisioned RCU of 5440. The average size of items stored in the database is 17 KB.  Which of the following is the MOST suitable action that should you do to make the application more cost-effective while maintaining its performance?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:03 |  |
| * Implement exponential backoff. * Decrease the provisioned RCU down to 800. * Switch the table from using provisioned mode to on-demand mode. * **Set the provisioned RCU to 1600.**   In this scenario, a Lambda function makes 320 strongly consistent read requests per second against a DynamoDB table which has a provisioned RCU of 5440. The average size of items stored in the database is 17 KB.  It seems that the RCU is quite high and the calculations to come up with this value are incorrect. If you multiply 320 x 17, then you'll get 5,440 which is a correct calculation for WCU but **not** for the RCU.    **1 RCU** can do **1 strongly consistent read** or **2 eventually consistent reads** for an item up to **4KB**.  **To get the RCU with strongly consistent reads, do the following steps:**  **Step #1 Divide the average item size by 4 KB. Round up the result**  Average Item Size = 17 KB  = **17KB/4KB**  **= 4.25 ≈ 5**  **Step #2 Multiply the number of reads per second by the resulting value from Step 1. (Divide the product by 2 for eventually consistent reads)**  =**320** reads per second **x** **5**  = **1,600** **strongly consistent read requests**  Hence, the correct answer is to **set the provisioned RCU to 1600** as this will lower the cost and still maintain the performance of your application.  **Implementing exponential backoff**is incorrect because this is only applicable for error retries and error handling of the serverless application.  **Decreasing the provisioned RCU down to 800**is incorrect. Although this will lower the cost, it will not meet the strong consistency requirements of the application. Take note that the Lambda function makes read requests with a strong consistency type and not eventual consistency.  **Switching the table from using provisioned mode to on-demand mode**is incorrect. Although this will lower the cost, the on-demand mode is more suitable for unpredictable application traffic. The scenario explicitly mentioned the exact application traffic, which is why the provisioned mode is more suitable to use.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Reads>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/>    **Calculating the Required Read and Write Capacity Unit for your DynamoDB Table:**  <https://tutorialsdojo.com/calculating-the-required-read-and-write-capacity-unit-for-your-dynamodb-table/> | | | | | | |
| 12 | A company is developing a serverless website that consists of images, videos, HTML pages and JavaScript files. There is also a requirement to serve the files with lowest possible latency to its global users.  Which combination of services should be used in this scenario? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:13 |  |
| * Amazon EC2 * **Amazon S3** * Amazon Elastic File System * Amazon Glacier * **Amazon CloudFront**   You can configure your application to deliver static content and decrease the end-user latency using Amazon S3 and Amazon CloudFront. High-resolution images, videos, and other static files can be stored in Amazon S3. CloudFront speeds up content delivery by leveraging its global network of data centers, known as edge locations, to reduce delivery time by caching your content close to your end-users.    CloudFront fetches your content from an origin, such as an Amazon S3 bucket, an Amazon EC2 instance, an Amazon Elastic Load Balancing load balancer, or your own web server, when it's not already in an edge location. CloudFront can be used to deliver your entire website or application, including dynamic, static, streaming, and interactive content. You can set your Amazon S3 bucket as the origin of your CloudFront web distribution.  Hence, the correct answers are:  - **Amazon S3**  - **Amazon CloudFront**  **Amazon EC2** is incorrect because EC2 instances are more suited for dynamic, server-side processing and they are region-specific, which means the latency for users would vary depending on their geographical distance from the EC2 instance. They are not designed for low-latency global content delivery of static web assets.  **Amazon Elastic File System**is incorrect because this is not a suitable service for storing static content, unlike S3. It is a regional service storing data within and across multiple Availability Zones (AZs) for high availability and durability. In addition, you can't directly connect it to CloudFront, unlike S3.  **Amazon Glacier**is incorrect because this is primarily used for data archival with usually a long data retrieval time. Like EFS, you can't directly connect it to CloudFront, unlike Amazon S3.    **References:**  <https://aws.amazon.com/getting-started/tutorials/deliver-content-faster/>  <https://aws.amazon.com/cloudfront/>  <https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/>    **Check out these Amazon S3 and CloudFront Cheat Sheets:**  <https://tutorialsdojo.com/amazon-s3/>  <https://tutorialsdojo.com/amazon-cloudfront/> | | | | | | |
| 13 | A developer is building an AI-based traffic monitoring application using Lambda in AWS. Due to the complexity of the application, the developer must do certain modifications such as the way Lambda runs the function's setup code and how the invocation events are read from the Lambda runtime API.  In this scenario, which feature of Lambda should you take advantage of to meet the above requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:40 |  |
| * **Custom Runtime** * Layers * Lambda@Edge * DLQ   You can implement an AWS Lambda runtime in any programming language. A runtime is a program that runs a Lambda function's handler method when the function is invoked. You can include a runtime in your function's deployment package in the form of an executable file named bootstrap.  A runtime is responsible for running the function's setup code, reading the handler name from an environment variable, and reading invocation events from the Lambda runtime API. The runtime passes the event data to the function handler and posts the response from the handler back to Lambda.  Your custom runtime runs in the standard Lambda execution environment. It can be a shell script, a script in a language that's included in Amazon Linux, or a binary executable file that's compiled in Amazon Linux.  Take note that Lambda has a deployment package size limit of 50 MB for direct upload (zipped file) and 250 MB for layers (unzipped).  Hence, the correct answer in this scenario is: **Custom Runtime.**  **Layers** is incorrect because this just enables you to use libraries and other dependencies in your function without having to include them in your deployment package.  **Lambda@Edge**is incorrect because this is actually a feature of Amazon CloudFront and not Lambda, which lets you run code closer to users of your application to improve performance and reduce latency.  **DLQ** is incorrect because the Dead Letter Queue (DLQ) only directs unprocessed events to an Amazon SQS queue or an Amazon SNS topic to analyze the failure if the retries fail. This feature does not meet the required capabilities mentioned in the scenario.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html>  <https://docs.aws.amazon.com/lambda/latest/dg/runtimes-walkthrough.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 14 | The company that you are working for recently decided to migrate and transform their monolithic application on-premises to a Lambda application. It is your responsibility to ensure that application works effectively in AWS.  Which of the following are the best practices in developing Lambda functions? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:19 |  |
| * **Take advantage of Execution Context reuse to improve the performance of your function.** * **Use AWS Lambda Environment Variables to pass operational parameters to your function.** * Use recursive code. * Include the core logic in the Lambda handler. * Use Amazon Inspector for troubleshooting.   Below are some of the best practices in working with AWS Lambda Functions:   - Separate the Lambda handler (entry point) from your core logic.   - Take advantage of Execution Context reuse to improve the performance of your function   - Use AWS Lambda Environment Variables to pass operational parameters to your function.   - Control the dependencies in your function's deployment package.   - Minimize your deployment package size to its runtime necessities.   - Reduce the time it takes Lambda to unpack deployment packages   - Minimize the complexity of your dependencies   - Avoid using recursive code  Hence, the correct answers in this scenario are:  **- Take advantage of Execution Context reuse to improve the performance of your function**  **- Use AWS Lambda Environment Variables to pass operational parameters to your function**  **Using recursive code** is incorrect because this is a situation wherein the function automatically calls itself until some arbitrary criteria is met. This could lead to an unintended volume of function invocations and escalated costs.  **Including the core logic in the Lambda handler**is incorrect because you have to separate the Lambda handler (entry point) from your core logic instead.  **Using Amazon Inspector for troubleshooting**is incorrect because this service is primarily used for EC2 and not for Lambda. You have to use X-Ray instead of troubleshooting your functions.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html>  <https://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 15 | You are configuring the task definitions of your ECS Cluster in AWS to make sure that the tasks are scheduled on instances with enough resources to run them. It should also follow the constraints that you specified both implicitly or explicitly.  Which of the following options should you implement to satisfy the requirement which requires the LEAST amount of configuration?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:33 |  |
| * Use a binpack task placement strategy. * **Use a random task placement strategy.** * Use a spread task placement strategy which uses the instanceId and host attributes. * **Use a spread task placement strategy with custom placement constraints.**   By default, tasks are randomly placed with RunTask or spread across Availability Zones with CreateService. Spread is typically used to achieve high availability by making sure that multiple copies of a task are scheduled across multiple instances based on attributes such as Availability Zones.  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 *Random* task placement strategy is fairly straightforward as it doesn’t require further parameters. The two other strategies, such as binpack and spread, take opposite actions. Binpack places tasks on as few instances as possible, helping to optimize resource utilization, while spread places tasks evenly across your cluster to help maximize availability. By default, ECS uses spread with the *ecs.availability-zone* attribute to place tasks.  *Random* places tasks on instances at random yet still honors the other constraints that you specified, implicitly or explicitly. Specifically, it still makes sure that tasks are scheduled on instances with enough resources to run them.  Hence, the correct answer is to **use a random task placement strategy**for this scenario.  **Using a binpack task placement strategy**is incorrect because this configuration will place the tasks based on the least available amount of CPU or memory. There are also additional configuration steps where you need to specify the type of field that ECS would be using such as CPU or memory.  **Using a spread task placement strategy which uses the instanceId and host attributes** is incorrect because this entails a lot of configuration as compared to using the *Random* task placement strategy type.  **Using a spread task placement strategy with custom placement constraints**is incorrect because a *task placement constraint* is just a rule that is considered during task placement.    **References:**  <https://aws.amazon.com/blogs/compute/amazon-ecs-task-placement/>  <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-placement.html>  <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-placement-strategies.html>    **Check out this Amazon ECS Cheat Sheet:**  <https://tutorialsdojo.com/amazon-elastic-container-service-amazon-ecs/> | | | | | | |
| 16 | You are developing an online game where the app preferences and game state of the player must be synchronized across devices. It should also allow multiple users to synchronize and collaborate shared data in real time.  Which of the following is the MOST appropriate solution that you should implement in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:19 |  |
| * **Integrate AWS AppSync to your mobile app.** * Integrate AWS Amplify to your mobile app. * **Integrate Amazon Cognito Sync to your mobile app.** * Integrate Amazon Pinpoint to your mobile app.   **AWS AppSync** simplifies application development by letting you create a flexible API to securely access, manipulate, and combine data from one or more data sources. AppSync is a managed service that uses GraphQL to make it easy for applications to get exactly the data they need.  With AppSync, you can build scalable applications, including those requiring real-time updates, on a range of data sources such as NoSQL data stores, relational databases, HTTP APIs, and your custom data sources with AWS Lambda. For mobile and web apps, AppSync additionally provides local data access when devices go offline, and data synchronization with customizable conflict resolution, when they are back online.    AWS AppSync is quite similar with Amazon Cognito Sync which is also a service for synchronizing application data across devices. It enables user data like app preferences or game state to be synchronized as well however, the key difference is that, it also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.  Hence, the correct answer is to **integrate AWS AppSync to your mobile app**.  **Integrating AWS Amplify to your mobile app**is incorrect because this service just makes it easy to create, configure, and implement scalable mobile and web apps powered by AWS. This is primarily used to automate the application release process of both your frontend and backend allowing you to deliver features faster, and not for synchronizing application data across devices.  **Integrating Amazon Cognito Sync to your mobile app**is incorrect. Although this service can also be used in synchronizing application data across devices, it does not allow multiple users to synchronize and collaborate in real-time on shared data, unlike AWS AppSync.  **Integrating Amazon Pinpoint to your mobile app**is incorrect because this service simply allows you to engage with your customers across multiple messaging channels. This is primarily used to send push notifications, emails, SMS text messages, and voice messages.    **References:**  <https://aws.amazon.com/appsync/>  <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-sync.html> | | | | | | |
| 17 | You currently have an IAM user for working in the development environment using shell scripts that call the AWS CLI. The EC2 instance that you are using already contains the access key credential set and an IAM role, which are used to run the CLI and access the development environment. You were given a new set of access key credentials with another IAM role that allows you to access and manage the production environment.  Which of the following is the EASIEST way to switch from one role to another?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:02:43 |  |
| * **Create a new profile for the role in the AWS CLI configuration file then append the --profile parameter, along with the new profile name, whenever you run the CLI command.** * Store the production access key credentials set in the instance metadata and call this whenever you need to access the production environment. * Create a new instance profile in the AWS CLI configuration file then append the --profile parameter, along with the new profile name, whenever you run the CLI command. * Store the production access key credentials set in the user data of the instance and call this whenever you need to access the production environment.   Using roles to grant permissions to applications that run on EC2 instances requires a bit of extra configuration. An application running on an EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, an additional step is needed to assign an AWS role and its associated permissions to an EC2 instance and make them available to its applications.  This extra step is the creation of an [*instance profile*](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html) that is attached to the instance. The instance profile contains the role and can provide the role's temporary credentials to an application that runs on the instance. Those temporary credentials can then be used in the application's API calls to access resources and to limit access to only those resources that the role specifies. Note that only one role can be assigned to an EC2 instance at a time, and all applications on the instance share the same role and permissions.    Using roles in this way has several benefits. Because role credentials are temporary and rotated automatically, you don't have to manage credentials, and you don't have to worry about long-term security risks. In addition, if you use a single role for multiple instances, you can make a change to that one role and the change is propagated automatically to all the instances.  Imagine that you have an IAM user for working in the development environment and you occasionally need to work with the production environment at the command line with the [AWS CLI](http://aws.amazon.com/cli/). You already have an access key credential set available to you. This can be the access key pair that is assigned to your standard IAM user. Or, if you signed in as a federated user, it can be the access key pair for the role that was initially assigned to you. If your current permissions grant you the ability to assume a specific IAM role, then you can identify that role in a "profile" in the AWS CLI configuration files. That command is then run with the permissions of the specified IAM role, not the original identity.  Note that when you specify that profile in an AWS CLI command, you are using the new role. In this situation, you cannot make use of your original permissions in the development account at the same time. The reason is that only one set of permissions can be in effect at a time.  Hence, the correct answer is to **create a new profile for the role in the AWS CLI configuration file then append the --profile parameter, along with the new profile name, whenever you run the CLI command**.  **Storing the production access key credentials set in the instance metadata and calling this whenever you need to access the production environment**is incorrect because instance metadata is primarily used to fetch the data about your instance that you can use to configure or manage the running instance. This is not suitable for use in storing the access keys of your AWS CLI.  **Creating a new instance profile in the AWS CLI configuration file then appending the --profile parameter along with the new profile name whenever you run the CLI command**is incorrect because an instance profile is just a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts. This is different from an AWS CLI **profile**, which you can use for switching to various profiles. In addition, an instance profile is associated with the instance and not configured in the AWS CLI.  **Storing the production access key credentials set in the user data of the instance and calling this whenever you need to access the production environment**is incorrect because user data is primarily used to configure an instance during launch, or to run a configuration script. Just like instance metadata, this is not suitable for use in storing the access keys of your AWS CLI.    **References:**  <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html>  <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html>    **Check out this AWS Identity & Access Management (IAM) Cheat Sheet:**  <https://tutorialsdojo.com/aws-identity-and-access-management-iam/> | | | | | | |
| 18 | You have two users concurrently accessing a DynamoDB table and submitting updates. If a user will modify a specific item in the table, she needs to make sure that the operation will not affect another user's attempt to modify the same item. You have to ensure that your update operations will only succeed if the item attributes meet one or more expected conditions.  Which of the following DynamoDB features should you use in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:25 |  |
| * Projection Expressions * Update Expressions * **Conditional writes** * Batch Operations   By default, the **DynamoDB** write operations (PutItem, UpdateItem, DeleteItem) are *unconditional*: each of these operations will overwrite an existing item that has the specified primary key.  DynamoDB optionally supports **conditional writes** for these operations. A conditional write will succeed only if the item attributes meet one or more expected conditions. Otherwise, it returns an error. Conditional writes are helpful in cases where multiple users attempt to modify the same item.  For example, by adding a conditional expression that checks if the current value of the item is still the same, you can be sure that your update will not affect the operations of other users:  aws dynamodb update-item \  --table-name ProductCatalog \  --key '{"Id":{"N":"1"}}' \  --update-expression "SET Price = :newval" \  --condition-expression "Price = :currval" \  --expression-attribute-values [file://expression-attribute-values.json](file:///expression-attribute-values.json/)    Hence, the correct answer is **conditional writes**.  **Using projection expressions** is incorrect because this is just a string that identifies the attributes you want to retrieve during a GetItem, Query, or Scan operation. Take note that the scenario calls for a feature that can be used during a write operation hence, this option is irrelevant.  **Using update expressions**is incorrect because this simply specifies how UpdateItem will modify the attributes of an item such as for setting a scalar value or removing elements from a list or a map. This feature doesn't use any conditions which is what the scenario is looking for. Therefore, this option is incorrect.  **Using batch operations**is incorrect because these are essentially wrappers for multiple read or write requests. Batch operations are primarily used when you want to retrieve or submit multiple items in DynamoDB through a single API call, which reduces the number of network round trips from your application to DynamoDB.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithItems.html#WorkingWithItems.ReadingData>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.html>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/> | | | | | | |
| 19 | A serverless application, which uses a DynamoDB database, is experiencing throttling issues during peak times. To troubleshoot the problem, you were instructed to get the total number of write capacity units consumed for the table and any secondary indexes whenever the UpdateItem operation is sent.  In this scenario, what is the MOST appropriate value for the ReturnConsumedCapacity parameter that you should set in the update request?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:42 |  |
| * TRUE * TOTAL * NONE * **INDEXES**   To create, update, or delete an item in a DynamoDB table, use one of the following operations:  - PutItem  - UpdateItem  - DeleteItem  For each of these operations, you need to specify the entire primary key, not just part of it. For example, if a table has a composite primary key (partition key and sort key), you must supply a value for the partition key and a value for the sort key.  To return the number of write capacity units consumed by any of these operations, set the ReturnConsumedCapacity parameter to one of the following:  **TOTAL** — returns the total number of write capacity units consumed.  **INDEXES** — returns the total number of write capacity units consumed, with subtotals for the table and any secondary indexes that were affected by the operation.  **NONE** — no write capacity details are returned. (This is the default.)  Hence, the correct answer is to add the ReturnConsumedCapacity parameter with a value of **INDEXES** in every update request.  Setting the parameter to **TRUE** is incorrect because the ReturnConsumedCapacity parameter is not a boolean type. The valid values that you can use are **TOTAL**, **INDEXES** and **NONE** only.  Setting the parameter to **TOTAL** is incorrect because this just returns the total number of write capacity units consumed but not the subtotals for the table and any secondary indexes that were affected by the operation.  Setting the parameter to **NONE** is incorrect because this is the default value where no write capacity details are returned.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithItems.html#WorkingWithItems.WritingData>  <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html#API_PutItem_RequestParameters>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/> | | | | | | |
| 20 | A developer wants to perform additional processing on newly inserted items in Amazon DynamoDB using AWS Lambda. In order to implement this requirement, the developer will have to use DynamoDB Streams to automatically send the new items in the table to a Lambda function for processing.  Given the scenario, what steps should be performed by the developer to integrate his/her DynamoDB to his/her Lambda functions? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:47 |  |
| * Create an SNS topic to capture new records from DynamoDB. * **Create an event source mapping in Lambda to send records from your stream to a Lambda function.** * **Select AWSLambdaDynamoDBExecutionRole managed policy as the function's execution role.** * Select AWSLambdaBasicExecutionRole managed policy as the function's execution role. * Create a trigger for a Kinesis Data Firehose delivery stream that uses a Lambda function for data processing.   You can use an **AWS Lambda function** to process records in an Amazon DynamoDB Streams stream. With **DynamoDB Streams**, you can trigger a Lambda function to perform additional work each time a DynamoDB table is updated.    You need to create an event source mapping to tell Lambda to send records from your stream to a Lambda function. You can create multiple event source mappings to process the same data with multiple Lambda functions, or process items from multiple streams with a single function. To configure your function to read from DynamoDB Streams in the Lambda console, create a DynamoDB trigger.  You also need to assign the following permissions to Lambda:  [dynamodb:DescribeStream](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_streams_DescribeStream.html)  [dynamodb:GetRecords](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_streams_GetRecords.html)  [dynamodb:GetShardIterator](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_streams_GetShardIterator.html)  [dynamodb:ListStreams](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_streams_ListStreams.html)  The AWSLambdaDynamoDBExecutionRole managed policy already includes these permissions.  Hence, the correct answers are:  **- Create an event source mapping in Lambda to send records from your stream to a Lambda function.**  **- Select AWSLambdaDynamoDBExecutionRole managed policy as the function's execution role.**  The option that says: **Create an SNS topic to capture new records from DynamoDB**is incorrect as there is no need to do this. Your Lambda triggers should be able to catch the events from DynamoDB Streams.  The option that says: **Select AWSLambdaBasicExecutionRole managed policy with full access to DynamoDB as the function's execution role** is incorrect because it lacks the necessary permissions. This role only provides Lambda permissions to upload logs to CloudWatch.  The option that says: **Create a trigger for a Kinesis Data Firehose delivery stream that uses a Lambda function for data processing**is incorrect. You can only use Lambda functions as triggers for DynamoDB Streams.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html>  <https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/>    **AWS Lambda Integration with Amazon DynamoDB Streams:**  <https://tutorialsdojo.com/aws-lambda-integration-with-amazon-dynamodb-streams/> | | | | | | |
| 21 | A company is transitioning their systems to AWS due to the limitations of their on-premises data center. As part of this project, a developer was assigned to build a brand new serverless architecture in AWS, which will be composed of AWS Lambda, API Gateway, and DynamoDB in a single stack. She needs a simple and reliable framework that will allow her to share configuration such as memory and timeouts between resources and deploy all related resources together as a single, versioned entity.  Which of the following is the MOST appropriate service that the developer should use in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:44 |  |
| * AWS CloudFormation * **AWS SAM** * AWS Systems Manager * Serverless Application Framework   The AWS Serverless Application Model (AWS SAM) is an open source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings. You define the application you want with just a few lines per resource and model it using YAML.  AWS SAM is natively supported by AWS CloudFormation and provides a simplified way of defining the Amazon API Gateway APIs, AWS Lambda functions, and Amazon DynamoDB tables needed by your serverless application. During deployment, SAM transforms and expands the SAM syntax into AWS CloudFormation syntax. Then, CloudFormation provisions your resources with reliable deployment capabilities.  Hence, the correct answer is **AWS SAM**.  **AWS CloudFormation**is incorrect. Although this service can deploy the serverless application to AWS, it is still more appropriate to use AWS SAM instead. AWS SAM can simplify the deployment of the serverless application by deploying all related resources together as a single, versioned entity.  **AWS Systems Manager**is incorrect because it is more focused on management and operations of AWS resources, such as automation, patching, and configuration, but it is not a deployment or application modeling tool.  **Serverless Application Framework**is incorrect. Although it is a well-known framework for building and deploying serverless applications into the AWS cloud, this is not an AWS native solution. It also does not allow configuration of DynamoDB databases or API Gateway APIs, unlike AWS SAM.    **References:** <https://aws.amazon.com/serverless/sam/>  <https://aws.amazon.com/serverless/developer-tools/>  <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>    **Check out this AWS Serverless Application Model Cheat Sheet:**  <https://tutorialsdojo.com/aws-serverless-application-model-sam/> | | | | | | |
| 22 | You were recently hired by a media company that is planning to build a news portal using Elastic Beanstalk and DynamoDB database, which already contains a few data. There is already an existing DynamoDB Table that has an attribute of ArticleName which acts as the partition key and a Category attribute as its sort key. You are instructed to develop a feature that will query the ArticleName attribute but will use a different sort key other than the existing one. The feature also requires strong read consistency to fetch the most up-to-date data.  Which of the following solutions should you implement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:48 |  |
| * Create a Global Secondary Index which uses the ArticleName attribute and your alternative sort key as projected attributes. * **Create a new DynamoDB table with a Local Secondary Index that uses the ArticleName attribute with a different sort key. Migrate the data from the existing table to the new table.** * Create a Global Secondary Index that uses the ArticleName attribute and a different sort key. * **Create a Local Secondary Index that uses the ArticleName attribute and a different sort key.**   A **local secondary index** maintains an alternate sort key for a given partition key value. A local secondary index also contains a copy of some or all of the attributes from its base table; you specify which attributes are projected into the local secondary index when you create the table. The data in a local secondary index is organized by the same partition key as the base table, but with a different sort key. This lets you access data items efficiently across this different dimension. For greater query or scan flexibility, you can create up to five local secondary indexes per table.  Suppose that an application needs to find all of the threads that have been posted within the last three months. Without a local secondary index, the application would have to Scan the entire *Thread*table and discard any posts that were not within the specified time frame. With a local secondary index, a Query operation could use *LastPostDateTime* as a sort key and find the data quickly.  To create a Local Secondary Index, make sure that the primary key of the index is the same as the primary key/partition key of the table, just as shown below. Then you must select an alternative sort key which is different from the sort key of the table.    When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful. A strongly consistent read might not be available if there is a network delay or outage. Strongly consistent reads are not supported on global secondary indexes.  The primary key of a local secondary index must be composite (partition key and sort key). A local secondary index lets you query over a single partition, as specified by the partition key value in the query.  Local secondary indexes are created at the same time that you create a table. You cannot add a local secondary index to an existing table, nor can you delete any local secondary indexes that currently exist.  Hence, the correct answer in this scenario is to **create a new DynamoDB table with a Local Secondary Index that uses the ArticleName attribute with a different sort key then migrate the data from the existing table to the new table**.  **Creating a Global Secondary Index that uses the ArticleName attribute and a different sort key**is incorrect because it is stated in the scenario that you are still using the same partition key, but with an alternate sort key that warrants the use of a local secondary index instead of a global secondary index.  **Creating a Global Secondary Index which uses the ArticleName attribute and your alternative sort key as projected attributes** is incorrect because using a local secondary index is a more appropriate solution to be used in this scenario just as explained above. Moreover, projected attributes are just attributes stored in the index that can be returned by queries and scans performed on the index hence, these are not useful in satisfying the provided requirement.  **Creating a Local Secondary Index that uses the ArticleName attribute and a different sort key** is incorrect. Although it uses the correct type of index, you cannot add a local secondary index to an already existing table.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/LSI.html>    **Global Secondary Index vs. Local Secondary Index:**  <https://tutorialsdojo.com/global-secondary-index-vs-local-secondary-index/>    **Tutorials Dojo's AWS Certified Developer Associate Exam Study Guide:**  <https://tutorialsdojo.com/aws-certified-developer-associate/> | | | | | | |
| 23 | You are a newly hired developer in a leading investment bank which uses AWS as its cloud infrastructure. One of your tasks is to develop an application that will store financial data to an already existing S3 bucket, which has the following bucket policy:  {  "Version": "2012-10-17",  "Id": "PutObjPolicy",  "Statement": [  {  "Sid": "AllowUploadCheck",  "Effect": "Deny",  "Principal": "\*",  "Action": "s3:PutObject",  "Resource": "arn:aws:s3:::tutorialsdojo\/\*",  "Condition": {  "StringNotEquals": {  "s3:x-amz-server-side-encryption": "AES256"  }  }  },  {  "Sid": "AllowNullCheck",  "Effect": "Deny",  "Principal": "\*",  "Action": "s3:PutObject",  "Resource": "arn:aws:s3:::tutorialsdojo\/\*",  "Condition": {  "Null": {  "s3:x-amz-server-side-encryption": "true"  }  }  }  ]  }  Which of the following statements is true about uploading data to this S3 bucket?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:04:29 |  |
| * **The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of AES256.** * The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of Null. * The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of true. * The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of aws:kms.   Server-side encryption protects data at rest. Amazon S3 encrypts each object with a unique key. 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 to the *tutorialsdojo* bucket unless the request includes the x-amz-server-side-encryption header to request server-side encryption:    {  "Version": "2012-10-17",  "Id": "PutObjPolicy",  "Statement": [  {  "Sid": "DenyIncorrectEncryptionHeader",  "Effect": "Deny",  "Principal": "\*",  "Action": "s3:PutObject",  "Resource": "arn:aws:s3:::tutorialsdojo/\*",  "Condition": {  "StringNotEquals": {  "s3:x-amz-server-side-encryption": "AES256"  }  }  },  {  "Sid": "DenyUnEncryptedObjectUploads",  "Effect": "Deny",  "Principal": "\*",  "Action": "s3:PutObject",  "Resource": "arn:aws:s3:::tutorialsdojo/\*",  "Condition": {  "Null": {  "s3:x-amz-server-side-encryption": "true"  }  }  }  ]  }    Take note that the *Sid* (statement ID) is just an optional identifier that you provide for the policy statement. The *Effect* element is required and specifies whether the statement results in an allow or an explicit deny. The valid values for the Effect element are Allow and Deny. The value of the *Sid* element does not affect nor override the *Effect* element.  Hence, the correct answer is: **The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of AES256.**  The option that says: **The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of Null** is incorrect because the value of the header should be*AES256*and not*Null*. Take note that the *Null* in the policy actually means that the request will be denied if there is no *x-amz-server-side-encryption* header included.  The option that says: **The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of true** is incorrect because this header is not a boolean type. It can only accept two values: *AES256* and *aws:kms*.  The option that says: **The bucket will deny object uploads unless the request includes the x-amz-server-side-encryption header with a value of aws:kms**is incorrect. Based on the given bucket policy, the value of the header should be *AES256,* which means that the bucket is using Amazon S3-Managed Keys (SSE-S3). Conversely, if this header has a value of *aws:kms*, then it uses AWS KMS Keys (SSE-KMS).    **References:**  <http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html>  <https://docs.aws.amazon.com/AmazonS3/latest/dev/DataDurability.html>  <https://aws.amazon.com/blogs/security/how-to-prevent-uploads-of-unencrypted-objects-to-amazon-s3/>    **Check out this Amazon S3 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-s3/> | | | | | | |
| 24 | A company has an application hosted in an On-Demand EC2 instance in your VPC. The developer has been instructed to create a shell script that fetches the instance's associated public and private IP addresses.  What should the developer do to complete this task?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:59 |  |
| * Get the public and private IP addresses from Amazon CloudWatch. * Get the public and private IP addresses from AWS CloudTrail. * **Get the public and private IP addresses from the instance metadata service using the http://169.254.169.254/latest/meta-data/ endpoint.** * Get the public and private IP addresses from the instance user data service using the http://169.254.169.254/latest/userdata/ endpoint.   **Instance metadata** is data about your EC2 instance that you can use to configure or manage the running instance. Because your instance metadata is available from your running instance, you do not need to use the Amazon EC2 console or the AWS CLI. This can be helpful when you're writing scripts to run from your instance. For example, you can access the local IP address of your instance from instance metadata to manage a connection to an external application.    To view the private IPv4 address, public IPv4 address, and all other categories of instance metadata from within a running instance, use the following URL: http://169.254.169.254/latest/meta-data/.  Hence, the correct answer is: **Get the public and private IP addresses from the instance metadata service using the http://169.254.169.254/latest/meta-data/ endpoint.**  The option that says:**Get the public and private IP addresses from Amazon CloudWatch** is incorrect because there is no direct way to fetch the public and private IP addresses of the EC2 instance using CloudWatch.  The option that says: **Get the public and private IP addresses from AWS CloudTrail** is incorrect because CloudTrail is primarily used to track the API activity of each AWS service. Just like CloudWatch, there is no easy way to get the associated IP addresses of the EC2 instance using CloudTrail.  The option that says: **Get the public and private IP addresses from the instance user data service using the http://169.254.169.254/latest/userdata/ endpoint** is incorrect because a user data is mainly used to perform common automated configuration tasks and run scripts after the instance starts. You will not find the associated IP addresses of the EC2 instance from its user data. You have to use the metadata service instead.    **References:**  <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>  <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html>    **Check out this Amazon EC2 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-elastic-compute-cloud-amazon-ec2/> | | | | | | |
| 25 | Your request to increase your account's concurrent execution limit to 2000 has been recently approved by AWS. There are 10 Lambda functions running in your account and you already specified a concurrency execution limit on one function at 400 and on another function at 200.  Which of the following statements are TRUE in this scenario? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:02:44 |  |
| * **The remaining 1400 concurrent executions will be shared among the other 8 functions.** * The unreserved concurrency pool is 600. * You can still set a concurrency execution limit of 1400 to a third Lambda function. * **The combined allocated 600 concurrent execution will be shared among the 2 functions.** * **You can still set a concurrency execution limit of 1300 to a third Lambda function.**   The unit of scale for AWS Lambda is a concurrent execution. However, scaling indefinitely is not desirable in all scenarios. For example, you may want to control your concurrency for cost reasons or to regulate how long it takes you to process a batch of events, or to simply match it with a downstream resource. To assist with this, Lambda provides a concurrent execution limit control at both the account level and the function level.  The *concurrent executions* refer 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.  If you set the concurrent execution limit for a function, the value is deducted from the unreserved concurrency pool. For example, if your account's concurrent execution limit is 1000 and you have 10 functions, you can specify a limit on one function at 200 and another function at 100. The remaining 700 will be shared among the other 8 functions.  AWS Lambda will keep the unreserved concurrency pool at a minimum of 100 concurrent executions so that functions that do not have specific limits set can still process requests. So, in practice, if your total account limit is 1000, you are limited to allocating 900 to individual functions.  In this scenario, you still have 1400 concurrent executions remaining which will be shared by the other 8 Lambda functions in your AWS account. Take note that the unreserved account concurrency can't go below **100**, which means that you only set a concurrency execution limit of 1300 to a single function or spread out to the remaining 8 functions.  Hence, the correct answers in this scenario are:  **- The remaining 1400 concurrent executions will be shared among the other 8 functions.**  **- You can still set a concurrency execution limit of 1300 to a third Lambda function.**  The option that says: **the unreserved concurrency pool is 600** is incorrect because this is the value of the total reserved concurrency that you have allocated to the 2 Lambda functions.  The option that says: **you can still set a concurrency execution limit of 1400 to a third Lambda function** is incorrect because the unreserved account concurrency cannot go below 100, which means that you only set a concurrency execution limit of 1300 to the third function or spread out to the remaining 8 functions.  The option that says: **the combined allocated 600 concurrent execution will be shared among the 2 functions** is incorrect because the execution limit is per function only and will not be shared with other functions, which also have reserved concurrent executions.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html>  <https://docs.aws.amazon.com/lambda/latest/dg/scaling.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/>    **Tutorials Dojo's AWS Certified Developer Associate Exam Study Guide:**  <https://tutorialsdojo.com/aws-certified-developer-associate/> | | | | | | |
| 26 | A company has assigned a developer to automate the patch management, data synchronization, and other recurring tasks in their department. The developer needs to have a service which can coordinate multiple AWS services into serverless workflows.  Which of the following is the MOST cost-effective service that the developer should implement in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:17 |  |
| * SWF * AWS Lambda * **AWS Step Functions** * AWS Batch   **AWS Step Functions** provides serverless orchestration for modern applications. Orchestration centrally manages a workflow by breaking it into multiple steps, adding flow logic, and tracking the inputs and outputs between the steps. As your applications execute, Step Functions maintains application state, tracking exactly which workflow step your application is in, and stores an event log of data that is passed between application components. That means that if networks fail or components hang, your application can pick up right where it left off.    Application development is faster and more intuitive with Step Functions, because you can define and manage the workflow of your application independently from its business logic. Making changes to one does not affect the other. You can easily update and modify workflows in one place, without having to struggle with managing, monitoring and maintaining multiple point-to-point integrations. Step Functions frees your functions and containers from excess code, so your applications are faster to write, more resilient, and easier to maintain.  **SWF** is incorrect because this is just a fully-managed state tracker and task coordinator service. It does not provide serverless orchestration to multiple AWS resources.  **Lambda** is incorrect. Although it is used for serverless computing, it does not provide a direct way to coordinate multiple AWS services into serverless workflows.  **AWS Batch** is incorrect because it is primarily used to efficiently run hundreds of thousands of batch computing jobs in AWS.    **Reference:**  <https://aws.amazon.com/step-functions/features/>    **Check out this AWS Step Functions Cheat Sheet:**  <https://tutorialsdojo.com/aws-step-functions/>    **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/> | | | | | | |
| 27 | A developer has recently launched a new API Gateway service which is integrated with AWS Lambda. He enabled API caching and per-key cache invalidation features in the API Gateway to comply with the requirement of the front-end development team which will use the API. The front-end team will have to invalidate an existing cache entry in some scenarios and fetch the latest data from the integration endpoint.  Which of the following should the consumers of the API do to invalidate the cache in API Gateway?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:23 |  |
| * Send a request with the Cache-Control: INVALIDATE\_CACHE header. * **Send a request with the Cache-Control: max-age=0 header.** * Send a request with the Cache-Control: no-cache header. * Configure the front-end application to clear the browser cache before fetching data from API Gateway.   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 **send a request with the Cache-Control: max-age=0 header**.  **Sending a request with the Cache-Control: no-cache header**is incorrect because you have to use value of the max-age directive in API Gateway instead of the no-cache directive. This just forces the cache to submit the request to the origin server for validation before releasing a cached copy.  **Configuring the frontend application to clear the browser cache before fetching data from API Gateway**is incorrect because the browser cache and the API Gateway cache are not connected with each other. The correct method of invalidating the cache is to add the Cache-Control: max-age=0 header*.*  **Sending a request with the Cache-Control: INVALIDATE\_CACHE header** is incorrect because there is no directive called INVALIDATE\_CACHE in the Cache-Control header.    **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/> | | | | | | |
| 28 | You are working as an IT Consultant for a top investment bank in Europe which uses several serverless applications in their AWS account. They just launched a new API Gateway service with a Lambda proxy integration and you were instructed to test out the new API. However, you are getting a Connection refused error whenever you use this Invoke URL http://779protaw8.execute-api.us-east-1.amazonaws.com/tutorialsdojo/ of the API Gateway.  Which of the following is the MOST likely cause of this issue?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:42 |  |
| * You are not using HTTP/2 in invoking the API. * **You are not using HTTPS in invoking the API.** * You are not using FTP in invoking the API. * You are not using WebSocket in invoking the API.   All of the APIs created with Amazon API Gateway expose **HTTPS** endpoints only. Amazon API Gateway does not support unencrypted (HTTP) endpoints. By default, Amazon API Gateway assigns an internal domain to the API that automatically uses the Amazon API Gateway certificate. When configuring your APIs to run under a custom domain name, you can provide your own certificate for the domain.    Calling a deployed API involves submitting requests to the URL for the API Gateway component service for API execution, known as execute-api. The base URL for REST APIs is in the following format:  **https**://*{restapi\_id}*.execute-api.*{region}*.amazonaws.com/*{stage\_name}*/  where *{restapi\_id}* is the API identifier, *{region}* is the region, and *{stage\_name}* is the stage name of the API deployment.  Hence, the most likely cause of the issue in the scenario is that **you are not using HTTPS in invoking the API**.  The option that says: **you are not using HTTP/2 in invoking the API** is incorrect because API Gateway only supports HTTPS.  The option that says: **you are not using FTP in invoking the API**is incorrect because API Gateway is using HTTPS to expose the APIs. FTP is primarily used for accessing file servers and not Web APIs.  The option that says: **you are not using WebSocket in invoking the API** is incorrect because all of the APIs created with Amazon API Gateway expose HTTPS endpoints only.    **References:**  <https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-call-api.html>  <https://aws.amazon.com/api-gateway/faqs/>    **Check out this Amazon API Gateway Cheat Sheet:**  <https://tutorialsdojo.com/amazon-api-gateway/> | | | | | | |
| 29 | You are a developer for a global technology company, which heavily uses AWS with regional offices in San Francisco, Manila, and Bangalore. Most of the clients of your company are using serverless computing in which you are responsible for ensuring that their applications are working efficiently.  Which of the following options are valid considerations in improving the performance of your Lambda function? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:51 |  |
| * You have to install the X-Ray daemon in Lambda to enable active tracing. * **An increase in memory size triggers an equivalent increase in CPU available to your function.** * **The concurrent execution limit is enforced against the sum of the concurrent executions of all function.** * Lambda automatically creates Elastic IP's that enable your function to connect securely to other resources within your private VPC. * You can throttle all incoming executions and stop processing any invocations to your function by setting concurrency to false.   You can use the AWS Lambda API or console to configure settings on your Lambda functions. Basic function settings include the description, role, and runtime that you specify when you create a function in the Lambda console. You can configure more settings after you create a function, or use the API to set things like the handler name, memory allocation, and security groups during creation.  Lambda counts a request each time it starts executing in response to an event notification or invocation call, including test invokes from the console. You are charged based on the total number of requests processed across all of your Lambda functions.    Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 1ms. The price depends on the amount of memory you allocate to your function.  In the AWS Lambda resource model, you choose the amount of memory you want for your function and are allocated proportional CPU power and other resources. An increase in memory size triggers an equivalent increase in CPU available to your function.  The unit of scale for AWS Lambda is a concurrent execution. However, scaling indefinitely is not desirable in all scenarios. For example, you may want to control your concurrency for cost reasons or to regulate how long it takes you to process a batch of events, or to simply match it with a downstream resource. To assist with this, Lambda provides a concurrent execution limit control at both the account level and the function level.    Hence, the valid considerations in improving the performance of Lambda functions are:  ***-*An increase in memory size triggers an equivalent increase in CPU available to your function.**  ***-*The concurrent execution limit is enforced against the sum of the concurrent executions of all functions.**  The option that says:**You have to install the X-Ray daemon in Lambda to enable active tracing** is incorrect because you only have to install the X-Ray daemon if you are using Elastic Beanstalk, ECS, or EC2 instances. You simply need to tick the *Enable AWS X-Ray* checkbox in the Lambda function to enable active tracing.  The option that says: **Lambda automatically creates Elastic IPs that enable your function to connect securely to other resources within your private VPC** is incorrect because Lambda actually creates ENI (Elastic Network Interface) and not Elastic IPs if the function is connected to your VPC.  The option that says: **You can throttle all incoming executions and stop processing any invocations to your function by setting concurrency to false**is incorrect because the concurrency setting is not a boolean type which is why setting it as *false* is invalid. To throttle all incoming executions, you can manually set the concurrency to 0 or just click the 'Throttle' button in the Lambda console.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html>  [https://aws.amazon.com/lambda/pricing/](https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html)  [https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html](https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html)    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 30 | A developer has a [Node.js](http://node.js/) function running in AWS Lambda. Currently, the code initializes a database connection to an Amazon RDS database every time the Lambda function is executed, and closes the connection before the function ends.  What feature in AWS Lambda will allow the developer to reuse the already existing database connection instead of initializing it each time the function is run?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:25 |  |
| * **Execution context** * **Environment variables** * Event source mapping * AWS Lambda is not capable of maintaining existing database connections due to its transient data store.   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. 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.  Hence, the correct answer is: **Execution context.**  **Environment variables** is incorrect because these are just variables that 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.  **Event source mapping**is incorrect because an event source is just an entity that publishes events and is integrated with your Lambda function. Supported event sources are the AWS services that can be preconfigured to work with AWS Lambda. The configuration is referred to as *event source mapping*, which maps an event source to a Lambda function. It enables automatic invocation of your Lambda function when events occur.  The option that says: **AWS Lambda is not capable of maintaining existing database connections due to its transient data store** is incorrect because Lambda actually is capable of doing this using the execution context just as discussed above.    **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-code>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 31 | A developer is planning to build a serverless Rust application in AWS using AWS Lambda and Amazon DynamoDB. Much to his disappointment, AWS Lambda does not natively support the Rust programming language.  Can the developer still proceed with creating serverless Rust applications in AWS given the situation above?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:20 |  |
| * Yes. The developer can submit a request ticket to AWS so that they can provide him a Lambda runtime environment that supports Rust. * Yes. The developer will just have to use AWS Fargate instead of AWS Lambda. * **Yes. The developer can create a custom runtime for hist Rust applications and bootstrap it to an AWS Lambda function.** * No. The developer will have to wait for a new support release in AWS Lambda.   AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code. It also provides a **Runtime API** which allows you to use any additional programming languages to author your functions. A runtime is a program that runs a Lambda function's handler method when the function is invoked. You can include a runtime in your function's deployment package in the form of an executable file named bootstrap.  Your custom runtime runs in the standard Lambda execution environment. It can be a shell script, a script in a language that's included in Amazon Linux, or a binary executable file that's compiled in Amazon Linux. Therefore, if the developer publishes a custom runtime for Rust, he can continue building his serverless application in AWS Lambda.  Hence, the correct answer in this scenario is: **Yes. The developer can create a custom runtime for hist Rust applications and bootstrap it to an AWS Lambda function**.  The option that says: **Yes. The developer can submit a request ticket to AWS so that they can provide him a Lambda runtime environment that supports Rust** is incorrect because you cannot request specific runtime environments for AWS Lambda from AWS. You would need to create this yourself using the **Runtime API**.  The option that says: **Yes. The developer will just have to use AWS Fargate instead of AWS Lambda** is incorrect since this service is a serverless compute engine for containers. Unless the Rust application is running in Docker, which is not explicitly stated in the scenario, it'll be best to use AWS Lambda for serverless computing.  The option that says: **No. The developer will have to wait for a new support release in AWS Lambda** is incorrect because there is no need to wait for a new feature release or for code translation since AWS Lambda allows you to create a runtime that appropriately handles your function code when invoked.    **References:**  <https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html>  <https://aws.amazon.com/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components/>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 32 | Your Lambda function initializes a lot of external dependencies such as database connections and HTTP endpoints, which are required for data processing. It also fetches static data with a size of 20 MB from a third-party provider over the Internet every time the function is invoked. This adds significant time in the total processing, which greatly affects the performance of their serverless application.  Which of the following should you do to improve the performance of your function?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:36 |  |
| * Use unreserved concurrency for your function. * Allocate more memory to your function. * Increase the CPU allocation of the function by submitting a service limit increase ticket to AWS. * **Place the database and HTTP initialization logic outside the Lambda function handler and store the external files in the /tmp directory.**   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: **Place the database and HTTP initialization logic outside the Lambda function handler and store the external files in the /tmp directory.**  The option that says: **Increase the CPU allocation of the function by submitting a service limit increase ticket to AWS**is incorrect because, in the first place, you cannot do that in AWS. In the AWS Lambda resource model, you choose the amount of memory you want for your function, which will then automatically allocate proportional CPU power to your function. An increase in memory size triggers an equivalent increase in CPU available to your function. This is the proper way to increase the CPU allocation and not by submitting a support ticket. In addition, the root cause of this issue is not the CPU nor the memory, but the 20 MB file that is always downloaded by your function.  The option that says: **Allocate more memory to your function** is incorrect because this will just increase the amount of memory available to the function during execution and not solve the underlying issue. The actual processing time may be reduced by having more memory but there is still a lot of time wasted in downloading the 20 MB file every time the function is invoked.  The option that says: **Use unreserved concurrency for your function**is incorrect because the issue does not relate to concurrency. Just as mentioned above, the root cause is that the function downloads a large file every time it is invoked, which causes significant delays and time-outs.    **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](https://docs.aws.amazon.com/lambda/latest/dg/running-lambda-code.html)    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 33 | A company has recently developed a containerized application that uses a multicontainer Docker platform which supports multiple containers per instance. They need a service that automatically handles tasks such as provisioning of the resources, load balancing, auto-scaling, monitoring, and placing the containers across the cluster.  Which of the following services provides the EASIEST way to accomplish the above requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:49 |  |
| * **ECS** * Lambda * EKS * **Elastic Beanstalk**   You can create docker environments that support multiple containers per Amazon EC2 instance with multicontainer Docker platform for Elastic Beanstalk.  **Elastic Beanstalk** uses Amazon Elastic Container Service (Amazon ECS) to coordinate container deployments to multicontainer Docker environments. Amazon ECS provides tools to manage a cluster of instances running Docker containers. Elastic Beanstalk takes care of Amazon ECS tasks including cluster creation, task definition and execution.    **AWS Elastic Beanstalk** is an application management platform that helps customers easily deploy and scale web applications and services. It keeps the provisioning of building blocks (e.g., EC2, RDS, Elastic Load Balancing, Auto Scaling, CloudWatch), deployment of applications, and health monitoring abstracted from the user so they can just focus on writing code. You simply specify which container images are to be deployed, the CPU and memory requirements, the port mappings, and the container links.  Elastic Beanstalk will automatically handle all the details such as provisioning an Amazon ECS cluster, balancing load, auto-scaling, monitoring, and placing your containers across your cluster. Elastic Beanstalk is ideal if you want to leverage the benefits of containers but just want the simplicity of deploying applications from development to production by uploading a container image. You can work with Amazon ECS directly if you want more fine-grained control for custom application architectures.  Hence, the correct answer in this scenario is **Elastic Beanstalk.**  **ECS** is incorrect. Although it can host Docker applications, it doesn't automatically handle all the details such as resource provisioning, balancing load, auto-scaling, monitoring, and placing your containers across your cluster, unlike Elastic Beanstalk. Take note that even though you can use Service Auto Scaling in ECS, you still have to enable and configure it. Elastic Beanstalk still provides the easiest way to accomplish the requirements.  **Lambda** is incorrect because this is primarily used for serverless applications and not for Docker or any other containerized applications.  **EKS** is incorrect because Amazon EKS just provides you an easy way to run Kubernetes on AWS without needing to install and operate your own Kubernetes clusters.    **References:**  <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs.html>  <https://aws.amazon.com/ecs/faqs/>  **Check out this AWS Elastic Beanstalk Cheat Sheet:**  <https://tutorialsdojo.com/aws-elastic-beanstalk/> | | | | | | |
| 34 | A startup has recently launched their new mobile game and is gaining a lot of new users everyday. The founders plan to add a new feature which will enable cross-device syncing of user profile data across mobile devices to improve the user experience.  Which of the following services should they use to meet this requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:15 |  |
| * AWS Amplify * **Cognito Sync** * Cognito User Pools * Cognito Identity Pools   **Amazon Cognito Sync** is an AWS service and client library that enables cross-device syncing of application-related user data. You can use it to synchronize user profile data across mobile devices and the web without requiring your own backend. The client libraries cache data locally so your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data, and if you set up push sync, notify other devices immediately that an update is available.  Amazon Cognito lets you save end user data in datasets containing key-value pairs. This data is associated with an Amazon Cognito identity, so that it can be accessed across logins and devices. To sync this data between the Amazon Cognito service and an end user’s devices, invoke the synchronize method. Each dataset can have a maximum size of 1 MB. You can associate up to 20 datasets with an identity.    The Amazon Cognito Sync client creates a local cache for the identity data. Your app talks to this local cache when it reads and writes keys. This guarantees that all of your changes made on the device are immediately available on the device, even when you are offline. When the synchronize method is called, changes from the service are pulled to the device, and any local changes are pushed to the service. At this point the changes are available to other devices to synchronize.  Amazon Cognito automatically tracks the association between identity and devices. Using the push synchronization, or push sync, feature, you can ensure that every instance of a given identity is notified when identity data changes. Push sync ensures that, whenever the sync store data changes for a particular identity, all devices associated with that identity receive a silent push notification informing them of the change.  Hence, the correct answer is to use **Cognito Sync***.*  **Cognito User Pools**is incorrect because this is just a user directory that allows your users to sign in to your web or mobile app through Amazon Cognito.  **Cognito Identity Pools**is incorrect because this simply enables you to create unique identities for your users and federate them with identity providers where you can obtain temporary, limited-privilege AWS credentials to access other AWS services.  **AWS Amplify**is incorrect because this just makes it easy for you to create, configure, and implement scalable mobile and web apps powered by AWS. It does not have the ability to synchronize user profile data across mobile devices.    **References:**  <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-sync.html>  <https://docs.aws.amazon.com/cognito/latest/developerguide/push-sync.html> | | | | | | |
| 35 | You are working as a software developer for an online training company, which is currently developing a learning portal that will use a DynamoDB table. One of the acceptance criteria requires you to ensure that there will be no hot partitions in the table which will result in throttling and inefficient use of your provisioned I/O capacity. The portal contains hundreds of thousands of online courses including the ones from their 3rd-party educational partners, which may or may not have the same Course ID. The table is structured as shown below:    Which of the following is the MOST suitable partition key to use in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:27 |  |
| * Course ID * **Item ID** * Course Name * Course Price   The primary key that uniquely identifies each item in a DynamoDB table can be simple (a partition key only) or composite (a partition key combined with a sort key).  Generally speaking, you should design your application for uniform activity across all logical partition keys in the Table and its secondary indexes. You can determine the access patterns that your application requires, and estimate the total read capacity units and write capacity units that each table and secondary Index requires.  The partition key portion of a table's primary key determines the logical partitions in which a table's data is stored. This, in turn, affects the underlying physical partitions. Provisioned I/O capacity for the table is divided evenly among these physical partitions. Therefore a partition key design that doesn't distribute I/O requests evenly can create "hot" partitions that result in throttling and use your provisioned I/O capacity inefficiently.        The optimal usage of a table's provisioned throughput depends not only on the workload patterns of individual items but also on the partition-key design. This doesn't mean that you must access all partition key values to achieve an efficient throughput level, or even that the percentage of accessed partition key values must be high. It does mean that the more distinct partition key values that your workload accesses, the more those requests will be spread across the partitioned space. In general, you will use your provisioned throughput more efficiently as the ratio of partition key values accessed to the total number of partition key values increases.  Hence, the correct answer in this scenario is ***Item ID***since it uses automatically generated GUID of the online course items that provide more uniformity than the other attributes.  ***Course ID***is incorrect because although this can be used as a partition key, it is not as unique since the scenario says that the there are online courses from the company's 3rd-party educational partners which may or may not have the same Course ID. Hence, the most suitable attribute to be used as a partition key is *Item ID.*  ***Course Name***is incorrect because just as mentioned above, it is possible that there will be two or more online courses with the same name which means that this is not an appropriate attribute to be used.  ***Course Price***is incorrect because this is a non-unique value where two or more items can have the exact same price. Hence, it is not suitable to be used as a partition key.    **References:**  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html>  <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-uniform-load.html>    **Check out this Amazon DynamoDB Cheat Sheet:**  <https://tutorialsdojo.com/amazon-dynamodb/> | | | | | | |
| 36 | A software development company uses AWS CodePipeline as its CI/CD platform to build, test, and push deployments to its production environment. Recently, a developer created a Lambda function that will push the build details to a separate DynamoDB table. The Lambda function should be triggered after a successful build on the Pipeline.  Which of the following services will meet the specified requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:02:00 |  |
| * AWS CloudTrail Events * **Amazon EventBridge (Amazon CloudWatch Events)** * AWS Systems Manager * AWS CodeBuild   Monitoring is an important part of maintaining the reliability, availability, and performance of AWS CodePipeline. You should collect monitoring data from all parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs.  You can monitor CodePipeline events in EventBridge, which delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services. EventBridge routes that data to targets such as Lambda functions and Simple Notification Service (SNS) topics.    Depending on the type of state change, you might want to send notifications, capture state information, take corrective action, initiate events, or take other actions.  Hence, the correct answer is **Amazon EventBridge (Amazon CloudWatch Events).**  **AWS CloudTrail Events**is incorrect because this is just an event in CloudTrail that records activity in your AWS account, which can be an action taken by a user, role, or service that is monitorable by CloudTrail. A more suitable solution is to use Amazon EventBridge (Amazon CloudWatch Events) instead.  **AWS Systems Manager** is incorrect because it is primarily used for managing and configuring AWS resources, including EC2 instances, databases, and more. It's not designed for event-driven automation based on state changes in AWS resources.  **AWS CodeBuild** is incorrect because it cannot trigger a Lambda function directly. AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. You should use Amazon EventBridge (Amazon CloudWatch Events) for this scenario instead.    **References:**  <https://docs.aws.amazon.com/codepipeline/latest/userguide/detect-state-changes-cloudwatch-events.html>    **Check out this AWS CodePipeline Cheat Sheet:**  <https://tutorialsdojo.com/aws-codepipeline/> | | | | | | |
| Sub-Total: | | 36 | 26 (72.22%) | 10 (27.78%) | 00:37:09 | **72.22%** |
|  | | | | | | |
| Category: **CDA - Security** | | | | | | |
| 1 | A company is legally obligated to keep transaction records containing Personally Identifiable Information (PII) for a duration of five years. These records are stored in Amazon S3. To handle data redaction, the company has developed Lambda functions with naming conventions starting as RedactPII-[role], where <role>  represents different roles.  The company wants to provide varying levels of redaction based on each role, ensuring each user only sees the necessary data. Only a single copy of the records should be maintained.  Which combination of actions will achieve the given requirements? (Select THREE.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:04:10 |  |
| * **Create an S3 Access Point for each user role.** * Set up an S3 event notification to invoke the corresponding RedactPII-[role] function in response to GET requests. * Set up S3 Replication for the bucket. * **Configure an S3 Object Lambda Access Point for each S3 Access Point name. Associate the RedactPII-[role] Lambda functions with the corresponding S3 Object Lambda Access Point.** * Use the GetObjectLegalHold API to retrieve the redacted data. * **Use the GetObject API to retrieve the redacted data**   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, we have to create dedicated S3 Object Lambda Access Points, with each user having access to their own unique Access Point. Each of these Access Points is associated with the corresponding RedactPII-[role] Lambda function. On the client side, users issue standard GetObject requests to their specific S3 Object Lambda Access Point, allowing them to retrieve redacted data tailored to their role. This setup ensures that each user can only access the data associated with their role while maintaining the requirement of a single copy of the records in the S3 bucket.  Hence, the correct answers are:  **- Create an S3 Access Point for each user role.**  **- Configure an S3 Object Lambda Access Point for each S3 Access Point name. Associate the RedactPII-[role] Lambda functions with the corresponding S3 Object Lambda Access Point.**  **- Use the GetObject API to retrieve the redacted data**  The option that says: **Set up an S3 event notification to invoke the corresponding RedactPII-[role] function in response to GET requests** is incorrect because S3 event notifications are primarily used for notifying of new object creations and not for invoking a function in response to GET requests.  The option that says: **Set up S3 Replication for the bucket** is incorrect. S3 Replication is used to automatically replicate objects across different buckets or across AWS regions. This would create additional copies of the data, contradicting the requirement to maintain only one copy of the records.  The option that says: **Use the GetObjectLegalHold API to retrieve the redacted data** is incorrect. This API simply retrieves the current legal hold status for a specific S3 object. It doesn't play a role in the process of getting the redacted 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/>  <https://aws.amazon.com/blogs/machine-learning/protect-pii-using-amazon-s3-object-lambda-to-process-and-modify-data-during-retrieval/>    **Check out this Amazon S3 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-s3/> | | | | | | |
| 2 | A company uses AWS Systems Manager (SSM) Parameter Store to manage configuration details for multiple applications. The parameters are currently stored in the Standard tier. The company wants its operations team to be notified if there are sensitive parameters that haven’t been rotated within 90 days.  Which must be done to meet the requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:56 |  |
| * **Convert the sensitive parameters from Standard tier into Advanced tier. Set a NoChangeNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS.** * Configure a NoChangeNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS. * **Convert the sensitive parameters from Standard tier into Advanced tier. Set a ExpirationNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS.** * Set up an Amazon EventBridge (Amazon CloudWatch Events) event pattern that captures SSM Parameter-related events. Use Amazon SNS to send notifications.   Parameter policies help you manage a growing set of parameters by allowing you to assign specific criteria to a parameter, such as an expiration date or time to live. Parameter policies are especially helpful in forcing you to update or delete passwords and configuration data stored in Parameter Store, a capability of AWS Systems Manager. Take note that parameter policies are only available for parameters in the **Advanced tier**.  Parameter Store offers the following types of policies:  **Expiration -**deletes the parameter at a specific date  **ExpirationNotification -**sends an event to Amazon EventBridge (Amazon CloudWatch Events)when the specified expiration time is reached.  **NoChangeNotification -** sends an event to Amazon EventBridge (Amazon CloudWatch Events) when a parameter has not been modified for a specified period of time.  The **NoChangeNotification**policy sends a notification based on the LastModifiedTime attribute of the parameter. If you change or edit a parameter, the system resets the notification time period based on the new value of LastModifiedTime. In the scenario's case, we want to be notified if specific parameters were not rotated in the last 90 days.  In the scenario, the goal is to be notified if specific sensitive parameters have not been rotated within the past 90 days. Configuring the **NoChangeNotification** policy with a value of 90 days allows SSM to emit a notification to EventBridge whenever the LastModifiedTime of the sensitive parameters exceeds the specified time frame. However, setting the notification policy alone is not enough. You must configure Amazon EventBridge (Amazon CloudWatch Events) to capture the emitted events and route them to an Amazon SNS topic.  Hence, the correct answer is: **Convert the sensitive parameters from Standard tier into Advanced tier. Set a NoChangeNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS.**  The option that says: **Configure a NoChangeNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS**is incorrect because notification policies are not supported in the Standard tier. You must convert the parameters first into the Advanced tier.  The option that says: **Convert the sensitive parameters from Standard tier into Advanced tier. Set a ExpirationNotification policy with a value of 90 days. Use Amazon EventBridge (Amazon CloudWatch Events) to send a notification via Amazon SNS**is incorrect because the ExpirationNotification policy is for notifying when a parameter is about to expire, not when it hasn't been rotated. In this case, the NoChangeNotification policy should be used instead.  The option that says: **Set up an Amazon EventBridge (Amazon CloudWatch Events) event pattern that captures SSM Parameter-related events. Use Amazon SNS to send notifications**is incorrect. A notification policy must be enabled as well, otherwise, Amazon EventBridge (Amazon CloudWatch Events) won't be able to receive any notifications.    **References:**  <https://docs.aws.amazon.com/systems-manager/latest/userguide/parameter-store-policies.html>  <https://aws.amazon.com/about-aws/whats-new/2019/04/aws_systems_manager_parameter_store_introduces_advanced_parameters/>    **Check out this cheat sheet on AWS Secrets Manager vs Systems Manager Parameter Store:**  <https://tutorialsdojo.com/aws-secrets-manager-vs-systems-manager-parameter-store/> | | | | | | |
| 3 | A developer is managing an application hosted in EC2, which stores data in an S3 bucket. The application also uses HTTPS for secure communication. To comply with the new security policy, the developer must ensure that the data is encrypted at rest using an encryption key that is provided and managed by the company. The change should also provide AES-256 encryption to their data.  Which of the following actions could the developer take to achieve this? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:03:26 |  |
| * Use SSL to encrypt the data while in transit to Amazon S3. * Implement Amazon S3 server-side encryption with Amazon S3-Managed Encryption Keys. * Implement Amazon S3 server-side encryption with AWS KMS Keys (SSE-KMS). * **Implement Amazon S3 server-side encryption with customer-provided keys (SSE-C).** * **Encrypt the data on the client-side before sending to Amazon S3 using their own master key.**   Data protection refers to protecting data while in transit (as it travels to and from Amazon S3) and at rest (while it is stored on disks in Amazon S3 data centers). You can protect data in transit by using SSL or by using client-side encryption.  You have the following options for protecting data at rest in Amazon S3:  **Use Server-Side Encryption** – You request Amazon S3 to encrypt your object before saving it on disks in its data centers and decrypt it when you download the objects.  1. Use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)  2. Use Server-Side Encryption with AWS KMS Keys (SSE-KMS)  3. Use Server-Side Encryption with Customer-Provided Keys (SSE-C)  **Use Client-Side Encryption** – You can encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools.  1. Use Client-Side Encryption with AWS KMS Key  2. Use Client-Side Encryption Using a Client-Side Master Key    Hence, the valid actions that the developer can implement in this scenario are:  **- Implement Amazon S3 server-side encryption with customer-provided keys (SSE-C)**  **- Encrypt the data on the client-side before sending to Amazon S3 using their own master key.**  **Using SSL to encrypt the data while in transit to Amazon S3**is incorrect because the requirement is to only secure the data at rest and not data in transit. Hence, you have to use server-side encryption instead. Moreover, the scenario explicitly states that the application already uses HTTPS for secure communication.  **Implementing Amazon S3 server-side encryption with AWS KMS Keys (SSE-KMS)** is incorrect. Although you can upload the company's KMS keys (CMKs), the keys will be managed by KMS and not your company. This does not comply with the security policy mandated by the company.  **Implementing Amazon S3 server-side encryption with Amazon S3-Managed Encryption Keys**is incorrect because the Amazon S3-Managed encryption does not comply with the policy mentioned in the given scenario since the keys are managed by AWS (through Amazon S3) and not by the company. The suitable server-side encryption that you should use here is SSE-C.    **References:**  <http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html>  <https://docs.aws.amazon.com/AmazonS3/latest/dev/DataDurability.html>    **Check out this Amazon S3 Cheat Sheet:**  <https://tutorialsdojo.com/amazon-s3/> | | | | | | |
| 4 | You are a software developer for a multinational investment bank which has a hybrid cloud architecture with AWS. To improve the security of their applications, they decided to use AWS Key Management Service (KMS) to create and manage their encryption keys across a wide range of AWS services. You were given the responsibility to integrate AWS KMS with the financial applications of the company.  Which of the following are the recommended steps to locally encrypt data using AWS KMS that you should follow? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:41 |  |
| * **Use the GenerateDataKey operation to get a data encryption key then use the plaintext data key in the response to encrypt data locally.** * **Erase the plaintext data key from memory and store the encrypted data key alongside the locally encrypted data.** * Use the GenerateDataKeyWithoutPlaintext operation to get a data encryption key then use the plaintext data key in the response to encrypt data locally. * Erase the encrypted data key from memory and store the plaintext data key alongside the locally encrypted data. * Encrypt data locally using the Encrypt operation.   When you encrypt your data, your data is protected, but you have to protect your encryption key. One strategy is to encrypt it. *Envelope encryption* is the practice of encrypting plaintext data with a data key, and then encrypting the data key under another key.  You can even encrypt the data encryption key under another encryption key, and encrypt that encryption key under another encryption key. But, eventually, one key must remain in plaintext so you can decrypt the keys and your data. This top-level plaintext encryption key is known as the *master key*.    **AWS KMS** helps you to protect your master keys by storing and managing them securely. Master keys stored in AWS KMS, known as customer master keys (CMKs), never leave the AWS KMS FIPS validated hardware security modules unencrypted. To use an AWS KMS CMK, you must call AWS KMS.  It is recommended that you use the following pattern to encrypt data locally in your application:  1. Use the GenerateDataKey operation to get a data encryption key.  2. Use the plaintext data key (returned in the Plaintext field of the response) to encrypt data locally, then erase the plaintext data key from memory.  3. Store the encrypted data key (returned in the CiphertextBlob field of the response) alongside the locally encrypted data.  Hence, the valid steps in this scenario are the following:  **- Use the GenerateDataKey operation to get a data encryption key then use the plaintext data key in the response to encrypt data locally.**  **- Erase the plaintext data key from memory and store the encrypted data key alongside the locally encrypted data.**  The option that says: **Use the GenerateDataKeyWithoutPlaintext operation to get a data encryption key then using the plaintext data key in the response to encrypt data locally** is incorrect because you have to use the GenerateDataKey operation instead. This is because the GenerateDataKeyWithoutPlaintext operation will not return the plaintext data key just as its name implies.  The option that says: **Erase the encrypted data key from memory and storing the plaintext data key alongside the locally encrypted data** is incorrect because it should be the other way around. You have to erase the **plaintext** data key from memory and store the **encrypted** data key alongside the locally encrypted data.  The option that says: **Encrypt data locally using the Encrypt operation**is incorrect because the Encrypt operation is primarily used to encrypt RSA keys, database passwords, or other sensitive information. This operation can also be used to move encrypted data from one AWS region to another however, this is not recommended if you want to encrypt your data locally. You have to use the GenerateDataKey operation instead.    **References:**  <https://docs.aws.amazon.com/kms/latest/APIReference/API_GenerateDataKey.html>  [https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#enveloping](https://docs.aws.amazon.com/kms/latest/APIReference/API_GenerateDataKey.html)    **Check out these Amazon S3 and AWS KMS Cheat Sheets:**  <https://tutorialsdojo.com/amazon-s3/>  [https://tutorialsdojo.com/aws-key-management-service-aws-kms/](https://tutorialsdojo.com/amazon-s3/) | | | | | | |
| 5 | A software engineer is building a serverless application in AWS consisting of Lambda, API Gateway, and DynamoDB. She needs to implement a custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML to determine the caller's identity.  Which of the features of API Gateway is the MOST suitable one that she should use to build this feature?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:34 |  |
| * Resource Policy * Cross-Account Lambda Authorizer * Cross-Origin Resource Sharing (CORS) * **Lambda Authorizers**   A Lambda authorizer is an API Gateway feature that uses a Lambda function to control access to your API. When a client makes a request to one of your API's methods, API Gateway calls your Lambda authorizer, which takes the caller's identity as input and returns an IAM policy as output.  There are two types of Lambda authorizers:   - A **token-based** Lambda authorizer (also called a TOKEN authorizer) receives the caller's identity in a bearer token, such as a JSON Web Token (JWT) or an OAuth token.   - A **request parameter-based** Lambda authorizer (also called a REQUEST authorizer) receives the caller's identity in a combination of headers, query string parameters, stageVariables, and $context variables.    It is possible to use an AWS Lambda function from an AWS account that is different from the one in which you created your Lambda authorizer function by using a Cross-Account Lambda Authorizer.  Therefore, the correct answer in this scenario is to use **Lambda Authorizers** since this feature is useful if you want to implement a custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML.  **Resource Policy**is incorrect because this is simply a JSON policy document that you attach to an API to control whether a specified principal (typically an IAM user or role) can invoke the API. This can't be used to implement a custom authorization scheme.  **Cross-Origin Resource Sharing (CORS)**is incorrect because this just defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.  **Cross-Account Lambda Authorizer**is incorrect because this just enables you to use an AWS Lambda function from a different AWS account as your API authorizer function. Moreover, this is not a valid Lambda authorizer type.    **References:**  <https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html>  <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-lambda-authorizer-input.html>  <https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-lambda-authorizer-cross-account-lambda-authorizer.html>    **Check out this Amazon API Gateway Cheat Sheet:**  <https://tutorialsdojo.com/amazon-api-gateway/> | | | | | | |
| 6 | A company has a suite of web applications that is heavily using RDS database in Multi-AZ Deployments configuration with several Read Replicas. For improved security, you were instructed to ensure that all of their database credentials, API keys, and other secrets are encrypted and rotated on a regular basis. You should also configure your applications to use the latest version of the encrypted credentials when connecting to the RDS database.  Which of the following is the MOST appropriate solution to secure the credentials?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:26 |  |
| * Store the credentials to Systems Manager Parameter Store with a SecureString data type. * **Use AWS Secrets Manager to store and encrypt the credentials and enable automatic rotation.** * Store the credentials to AWS ACM. * Store the credentials in AWS KMS.   **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, **using AWS Secrets Manager to store and encrypt the credentials and enabling automatic rotation** is the most appropriate solution for this scenario.  **Storing the credentials to Systems Manager Parameter Store with a SecureString data type**is incorrect because, by default, Systems Manager Parameter Store doesn't rotate its parameters which is one of the requirements in the above scenario.  **Storing the credentials to AWS ACM**is incorrect because it is just a managed private CA service that helps you easily and securely manage the lifecycle of your private certificates to allow SSL communication to your application. This is not a suitable service to store database or any other confidential credentials.  **Storing the credentials in AWS KMS** is incorrect because this only makes it easy for you to create and manage encryption keys and control the use of encryption across a wide range of AWS services. This is primarily used for encryption and not for hosting your 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 these AWS Systems Manager and Secrets Manager Cheat Sheets:**  <https://tutorialsdojo.com/aws-systems-manager/>  <https://tutorialsdojo.com/aws-secrets-manager/> | | | | | | |
| 7 | A product design firm has adopted a remote work policy and wants to provide employees with access to a suite of CAD software through EC2 Spot instances. These instances will be deployed using a CloudFormation template. The development team must be able to securely obtain software license keys in the template each time it is needed.  Which solution meets this requirement while offering the most secure and cost-effective approach?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:47 |  |
| * Store the license key as a secret in AWS Secrets Manager. Use the secretsmanager dynamic reference to retrieve the secret in the CloudFormation template. * Pass the license key in the Parameter section of the CloudFormation template during stack creation. Enable the NoEcho attribute on the parameter. * **Store the license key as a SecureString in AWS Systems Manager (SSM) Parameter Store. Use the ssm-secure dynamic reference to retrieve the secret in the CloudFormation template.** * Embed the license keys in the Mapping section of the CloudFormation template. Let users choose the correct license key using the Parameter section. Enable the NoEcho attribute on the parameter.   Dynamic references provide a compact, powerful way for you to reference external values that are stored and managed in other services, such as the AWS Systems Manager Parameter Store or AWS Secrets Manager. When you use a dynamic reference, CloudFormation retrieves the value of the specified reference when necessary during stack and change set operations and passes the value to the appropriate resource. CloudFormation does not store the actual reference value.  The following snippet shows how you can use the ssm-secure dynamic reference to retrieve an IAM user's password from the Parameter Store for console login. IAMUserPassword pertains to the parameter name followed by the version number.  MyIAMUser:  Type: AWS::IAM::User  Properties:  UserName: 'MyUserName'  LoginProfile:  Password: '**{{resolve:ssm-secure:IAMUserPassword:10}}**'  In the scenario, storing the license key as SecureString means encrypting it using a KMS key, making it more secure than storing it in plaintext. It's also more cost-effective than Secrets Manager since you don't pay for the number of parameters you store in the Parameter Store (Standard tier).  Hence, the correct answer is:**Store the license key as a SecureString in AWS Systems Manager (SSM) Parameter Store. Use the ssm-secure dynamic reference to retrieve the secret in the CloudFormation template.**  The option that says: **Pass the license key in the Parameter section of the CloudFormation template during stack creation. Enable the NoEcho attribute on the parameter**is incorrect. Although using the NoEcho attribute can help prevent the license key from being displayed in plaintext in the CloudFormation logs and console outputs,  The option that says: **Store the license key as a secret in AWS Secrets Manager. Use the secretsmanager dynamic reference to retrieve the secret in the CloudFormation template**is incorrect. Although using Secrets Manager is a valid approach, it's less cost-effective compared to using SSM Parameter Store. With Secrets Manager, there is a monthly cost associated with storing secrets, whereas SSM Parameter Store (Standard tier) is free of charge.  The option that says: **Embed the license keys in the Mapping section of the CloudFormation template. Let users choose the correct license key using the Parameter section. Enable the NoEcho attribute on the parameter**is incorrect. Embedding sensitive data, such as license keys, within CloudFormation templates poses a security risk, as the data can be viewed by anyone with access to the template.    **References:**  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/dynamic-references.html>  <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/security-best-practices.html#creds>  <https://catalog.workshops.aws/cfn101/en-US/intermediate/templates/dynamic-references>    **Check out this AWS CloudFormation cheat sheet:**  <https://tutorialsdojo.com/aws-cloudformation/> | | | | | | |
| 8 | A company is currently in the process of integrating their on-premises data center to their cloud infrastructure in AWS. One of the requirements is to integrate the on-premises Lightweight Directory Access Protocol (LDAP) directory service to their AWS VPC using IAM.  Which of the following provides the MOST suitable solution to implement if the identity store that they are using is not compatible with SAML?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:00:41 |  |
| * Set up an IAM policy that references the LDAP identifiers and AWS credentials. * **Implement the AWS IAM Identity Center service to manage access between AWS and your LDAP.** * **Create a custom identity broker application in your on-premises data center and use STS to issue short-lived AWS credentials.** * Create IAM roles to rotate the IAM credentials whenever LDAP credentials are updated.   If your identity store is not compatible with SAML 2.0, then you can build a custom identity broker application to perform a similar function. The broker application authenticates users, requests temporary credentials for users from AWS, and then provides them to the user to access AWS resources.  The application verifies that employees are signed into the existing corporate network's identity and authentication system, which might use LDAP, Active Directory, or another system. The identity broker application then obtains temporary security credentials for the employees.  To get temporary security credentials, the identity broker application calls either **AssumeRole** or **GetFederationToken** to obtain temporary security credentials, depending on how you want to manage the policies for users and when the temporary credentials should expire. The call returns temporary security credentials consisting of an AWS access key ID, a secret access key, and a session token. The identity broker application makes these temporary security credentials available to the internal company application. The app can then use the temporary credentials to make calls to AWS directly. The app caches the credentials until they expire, and then requests a new set of temporary credentials.    Hence, the correct answer is: **Create a custom identity broker application in your on-premises data center and use STS to issue short-lived AWS credentials**.  The option that says: **Setting up an IAM policy that references the LDAP identifiers and AWS credentials**is incorrect because using an IAM policy is not enough to integrate your LDAP service into IAM. You need to use SAML, STS, or a custom identity broker instead.  The option that says: **Implementing the AWS IAM Identity Center service to manage access between AWS and your LDAP** is incorrect because the identity store that you are using is not SAML-compatible. AWS IAM Identity Center does not support non-SAML authentication methods.  The option that says: **Creating IAM roles to rotate the IAM credentials whenever LDAP credentials are updated**is incorrect because manually rotating the IAM credentials is not an optimal solution to integrate your on-premises and VPC network. You need to use SAML, STS, or a custom identity broker.    **References:**  <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_federated-users.html>  <https://aws.amazon.com/blogs/aws/aws-identity-and-access-management-now-with-identity-federation/>    **Check out this AWS Identity & Access Management (IAM) Cheat Sheet:**  <https://tutorialsdojo.com/aws-identity-and-access-management-iam/> | | | | | | |
| 9 | A financial mobile application has a serverless backend API which consists of DynamoDB, Lambda, and Cognito. Due to the confidential financial transactions handled by the mobile application, there is a new requirement provided by the company to add a second authentication method that doesn't rely solely on user name and password.  Which of the following is the MOST suitable solution that the developer should implement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:22 |  |
| * Create a custom application that integrates with Amazon Cognito which implements the second layer of authentication. * Use a new IAM policy to a user pool in Cognito. * Use Cognito with SNS to allow additional authentication via SMS. * **Integrate multi-factor authentication (MFA) to a user pool in Cognito to protect the identity of your users.**   You can add multi-factor authentication (MFA) to a user pool to protect the identity of your users. MFA adds a second authentication method that doesn't rely solely on usernames and passwords. You can choose to use SMS text messages, or time-based one-time (TOTP) passwords as second factors in signing in your users. You can also use adaptive authentication with its risk-based model to predict when you might need another authentication factor. It's part of the user pool's advanced security features, which also include protections against compromised credentials.    Multi-factor authentication (MFA) increases security for your app by adding another authentication method, and not relying solely on user name and password. You can choose to use SMS text messages, or time-based one-time (TOTP) passwords as second factors in signing in your users.  With adaptive authentication, you can configure your user pool to require second-factor authentication in response to an increased risk level.  Hence, the correct answer in this scenario is to **integrate multi-factor authentication (MFA) to a user pool in Cognito to protect the identity of your users.**  **Creating a custom application that integrates with Amazon Cognito which implements the second layer of authentication** is incorrect. Although this option is viable, it is not the most suitable solution in this scenario since you can simply use MFA as a second-factor authentication for the mobile app.  **Using a new IAM policy to a user pool in Cognito**is incorrect because an IAM Policy alone cannot implement a second-factor authentication. You have to configure Cognito to use MFA instead.  **Using Cognito with SNS to allow additional authentication via SMS**is incorrect. Although this is part of the MFA setup, using this solution alone is not enough if you didn't enable MFA in the first place.    **References:**  <https://docs.aws.amazon.com/cognito/latest/developerguide/managing-security.html>  <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html>    **Tutorials Dojo's AWS Certified Developer Associate Exam Study Guide:**  <https://tutorialsdojo.com/aws-certified-developer-associate/> | | | | | | |
| 10 | A web application is currently using an on-premises Microsoft SQL Server 2019 Enterprise Edition database. Your manager instructed you to migrate the application to Elastic Beanstalk and the database to RDS for SQL Server. For additional security, you must configure your database to automatically encrypt the actual data before it is written to storage, and automatically decrypt data when the data is read from storage.  Which of the following services will you use to achieve this?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:34 |  |
| * **Enable Transparent Data Encryption (TDE).** * Use IAM DB Authentication. * **Enable RDS Encryption.** * Use Microsoft SQL Server Windows Authentication.   Amazon RDS supports using Transparent Data Encryption (TDE) to encrypt stored data on your DB instances running Microsoft SQL Server. TDE automatically encrypts data before it is written to storage, and automatically decrypts data when the data is read from storage.  Amazon RDS supports TDE for the following SQL Server versions and editions:  - SQL Server 2019 Standard and Enterprise Editions  - SQL Server 2017 Enterprise Edition  - SQL Server 2016 Enterprise Edition  - SQL Server 2014 Enterprise Edition  - SQL Server 2012 Enterprise Edition  Transparent Data Encryption is used in scenarios where you need to encrypt sensitive data. For example, you might want to provide data files and backups to a third party, or address security-related regulatory compliance issues.  To enable transparent data encryption for an RDS SQL Server DB instance, specify the TDE option in an RDS option group that is associated with that DB instance.  Transparent data encryption for SQL Server provides encryption key management by using a two-tier key architecture. A certificate, which is generated from the database master key, is used to protect the data encryption keys. The database encryption key performs the actual encryption and decryption of data on the user database. Amazon RDS backs up and manages the database master key and the TDE certificate. To comply with several security standards, Amazon RDS is working to implement automatic periodic master key rotation.    TDE encrypts the actual data and log files at the database level. It automatically encrypts data before it is written to storage and automatically decrypts data when it is read from storage. Lastly, TDE provides encryption at the data level, protecting the data itself from unauthorized access.  Hence, the correct answer is to **Enable Transparent Data Encryption (TDE)***.*  The option that says: **Use IAM DB Authentication**is incorrect because this option just lets you authenticate to your DB instance using AWS Identity and Access Management (IAM) database authentication. The more appropriate security feature to use here is TDE.  The option that says: **Enable RDS Encryption** is incorrect. This option enables encryption at rest for the underlying storage volumes, but it does not automatically encrypt and decrypt the data itself. It relies on the database engine (in this case, SQL Server) to handle the encryption and decryption of the actual data.  The option that says: **Use Microsoft SQL Server Windows Authentication**is incorrect because this option is primarily used if you want to integrate RDS with your AWS Directory Service for Microsoft Active Directory (also called AWS Managed Microsoft AD) to enable Windows Authentication to authenticate users.    **References:**  <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.Options.TDE.html>  <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQLServer.html>    **Check out this Amazon RDS Cheat Sheet:**  <https://tutorialsdojo.com/amazon-relational-database-service-amazon-rds/> | | | | | | |
| 11 | A startup has recently launched a high-quality photo sharing portal using Amazon Lightsail and S3. They noticed that there are other external websites which are linking and using their photos without permission. This has caused an increase on their data transfer cost and potential revenue loss.  Which of the following is the MOST effective method to solve this issue?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:48 |  |
| * Enable cross-origin resource sharing (CORS) which allows cross-origin GET requests from all origins. * Configure the S3 bucket to remove public read access and use pre-signed URLs with expiry dates. * **Use a CloudFront web distribution with signed URLs or signed cookies.** * Block the IP addresses of the offending websites using Network Access Control List.   A signed URL contains extra information, such as an expiration date and time, providing greater control over access to your content. This additional information is presented in a policy statement, which is derived from either a predefined (canned) policy or a personalized (custom) policy.  CloudFront signed cookies allow you to control who can access your content when you don't want to change your current URLs or when you want to provide access to multiple restricted files, such as all of the files in a website's subscribers' area. This topic explains the considerations when using signed cookies and describes how to set signed cookies using canned and custom policies.  The most effective method to control unauthorized access to the photos and manage data transfer costs is to use a CloudFront web distribution with signed URLs or signed cookies. This approach allows the startup to enforce access controls at the CDN layer, ensuring that only authorized users can access the content.    Additionally, CloudFront Functions can be used to validate referrer headers, adding another layer of protection by ensuring that only requests originating from the startup’s own domain are allowed to access the content.    By using CloudFront with signed URLs or signed cookies, the startup can manage access effectively, providing a scalable solution that prevents unauthorized use while controlling data transfer costs. The use of CloudFront Functions to validate referrer headers adds an extra layer of security, ensuring that the photos are only accessible through authorized channels, thereby protecting the startup’s content and reducing unnecessary data transfer costs.  Hence, the correct answer is: **Use a CloudFront web distribution with signed URLs or signed cookies.**  The option that says:**Enable cross-origin resource sharing (CORS) which allows cross-origin GET requests from all origins**is incorrect as this will actually make the problem worse since you are allowing any website or origin to fetch the objects in the S3 bucket. Although using CORS is a valid solution, it should be properly configured to only enable access to your trusted domains.  The option that says:**Configure the S3 bucket to remove public read access and use pre-signed URLs with expiry dates**is incorrect because it is not scalable for large numbers of objects, as it would require generating pre-signed URLs for potentially thousands or millions of objects, making it impractical.  **Blocking the IP addresses of the offending websites using Network Access Control List**is incorrect because a quick change in IP address would easily bypass this configuration; hence, this is not an efficient method to implement.    **References:**  <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-signed-urls.html>  <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/SecurityAndPrivateContent.html>  <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/PrivateContent.html>    **Check out this Amazon CloudFront Cheat Sheet:**  <https://tutorialsdojo.com/amazon-cloudfront/>    **S3 Pre-signed URLs vs CloudFront Signed URLs vs Origin Access Identity (OAI) vs Origin Access Control (OAC)**  <https://tutorialsdojo.com/s3-pre-signed-urls-vs-cloudfront-signed-urls-vs-origin-access-identity-oai-origin-access-control-oac/> | | | | | | |
| Sub-Total: | | 11 | 8 (72.73%) | 3 (27.27%) | 00:14:25 | **72.73%** |
|  | | | | | | |
| Category: **CDA - Troubleshooting and Optimization** | | | | | | |
| 1 | In the next financial year, a company has decided to develop a completely new version of its legacy application that will utilize Node.js and GraphQL. The new architecture aims to offer an end-to-end view of requests as they traverse the application and display a map of the underlying components.  To achieve this, the application will be hosted in an Auto Scaling group (ASG) of Linux EC2 instances behind an Application Load Balancer (ALB) and must be instrumented to send trace data to the AWS X-Ray.  Which of the following options is the MOST suitable way to satisfy this requirement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:25 |  |
| * **Enable AWS X-Ray tracing on the ASG’s launch template.** * Enable AWS Web Application Firewall (WAF) on the ALB to monitor web requests. * Refactor your application to send segment documents directly to X-Ray by using the PutTraceSegments API. * **Use a user data script to install the X-Ray daemon.**   The AWS X-Ray SDK does not send trace data directly to AWS X-Ray. To avoid calling the service every time your application serves a request, the SDK sends the trace data to a daemon, which collects segments for multiple requests and uploads them in batches. Use a script to run the daemon alongside your application.  To properly instrument your application hosted in an EC2 instance, you have to install the X-Ray daemon by using a user data script. This will install and run the daemon automatically when you launch the instance. To use the daemon on Amazon EC2, create a new instance profile role or add the managed policy to an existing one. This will grant the daemon permission to upload trace data to X-Ray.    The AWS X-Ray daemon is a software application that listens for traffic on UDP port 2000, gathers raw segment data, and relays it to the AWS X-Ray API. The daemon works in conjunction with the AWS X-Ray SDKs and must be running so that data sent by the SDKs can reach the X-Ray service.  Hence, the correct answer is: **Use a user data script to install the X-Ray daemon.**  The option that says: **Enable AWS X-Ray tracing on the ASG’s launch template**is incorrect. There's no option to enable X-Ray tracing in a launch template of an ASG.  The option that says: **Enable AWS Web Application Firewall (WAF) on the ALB to monitor web requests**is incorrect. Although it can help monitor and protect the application from common web exploits, it's not capable of instrumenting the application.  The option that says: **Refactor your application to send segment documents directly to X-Ray by using the PutTraceSegments API**is incorrect. Although this solution will work, it entails a lot of manual effort to perform. You don't need to do this because you can just install the X-Ray daemon on the instance to automate this process.    **References:**  <https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-ec2.html>  <https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon.html#xray-daemon-permissions>    **Check out this AWS X-Ray Cheat Sheet:**  <https://tutorialsdojo.com/aws-x-ray/> | | | | | | |
| 2 | The infrastructure of an application is designed such that a producer sends data to a consumer via HTTPS. The consumer may sometimes take a while to process the messages, which can result in unexpected timeouts and cause newer messages not to be acknowledged immediately. To resolve this issue, a developer decided to introduce an Amazon SQS standard queue into the system. However, duplicate messages are still not being handled properly.  What should the developer do to ensure that messages are durably delivered and to prevent duplicate messages? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:45 |  |
| * Increase the timeout for the acknowledgement response. * **Configure the producer to set deduplication IDs for the messages.** * Increase the number of consumers polling from your standard queue. * **Create a FIFO queue as a replacement for the standard queue.** * Use a delay queue.   Amazon SQS offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components.  **Standard queues** support a nearly unlimited number of transactions per second (TPS) per action. Standard queues support at-least-once message delivery. However, occasionally (because of the highly distributed architecture that allows nearly unlimited throughput), more than one copy of a message might be delivered out of order.  **FIFO queues** have all the capabilities of the standard queue. The most important features of this queue type are FIFO (First-In-First-Out) delivery and *exactly-once processing.*Unlike standard queues, FIFO queues don't introduce duplicate messages. FIFO queues help you avoid sending duplicates to a queue. To configure deduplication, you must do one of the following:  - Enable content-based deduplication.  - Explicitly provide the message deduplication ID (or view the sequence number) for the message.    The **message deduplication ID** is the token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute (default) deduplication interval.  Hence, the correct answers in this scenario are:  **- Configure the producer to set deduplication IDs for the messages**  **- Create a FIFO queue as a replacement for the standard queue**  **Increasing the timeout period for acknowledgment responses**is incorrect. Although it can help reduce the chances of duplicate messages being sent, this option will still introduce duplicates since the system is using a Standard SQS queue. In addition, a longer timeout period may result in congestion in the system which can delay the delivery of messages that follows after an acknowledgment response.  **Increasing the number of consumers polling from your standard queue**is incorrect. Although it can help consume messages more quickly, the use of a standard queue will still introduce duplicates since this type of queue does not support exactly-once processing, unlike FIFO queues.  **Using a delay queue**is incorrect because this will just let you postpone the delivery of new messages to a queue for a number of seconds. If you create a delay queue, any messages that you send to the queue remain invisible to consumers for the duration of the delay period    **References:**  <https://docs.amazonaws.cn/en_us/AWSSimpleQueueService/latest/SQSDeveloperGuide/standard-queues.html>  <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html>  <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/using-messagededuplicationid-property.html>    **Check out this Amazon SQS Cheat Sheet:**  <https://tutorialsdojo.com/amazon-sqs/> | | | | | | |
| 3 | A developer uses AWS X-Ray to create a trace on an instrumented web application to identify any performance bottlenecks. The segment documents being sent by the application contain annotations that the developer wants to utilize in order to identify and filter out specific data from the trace.  Which of the following should the developer do in order to satisfy this requirement with minimal configuration? (Select TWO.)  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:02:33 |  |
| * **Use filter expressions via the X-Ray console.** * Configure Sampling Rules in the AWS X-Ray Console. * **Fetch the trace IDs and annotations using the GetTraceSummaries API.** * Fetch the data using the BatchGetTraces API. * Send trace results to an S3 bucket then query the trace output using Amazon Athena.   The compute resources running your application logic send data about their work as **segments**. A segment provides the resource's name, details about the request, and details about the work done.  A subset of segment fields are indexed by X-Ray for use with filter expressions. You can search for segments associated with specific information in the X-Ray console or by using the GetTraceSummaries API.    Even with sampling, a complex application generates a lot of data. When you choose a time period of traces to view in the X-Ray console, you might get more results than the console can display. You can narrow the results to just the traces that you want to find by using a **filter expression**. Running the GetTraceSummaries operation retrieves IDs and annotations for traces available for a specified time frame using an optional filter.  Hence, **using filter expressions via the X-Ray console** and **fetching the trace IDs and annotations using the GetTraceSummaries API** are the correct answers in this scenario.  **Fetching the data using the BatchGetTraces API** is incorrect because this API simply retrieves a list of traces specified by ID. It does not support filter expressions nor returns the annotations.  **Sending trace results to an S3 bucket then querying the trace output using Amazon Athena** is incorrect. Although this solution may work, this entails a lot of configuration which is contrary to what the scenario requires. There are other simpler methods of searching through traces in X-Ray such as using annotations and filter expressions.  **Configuring Sampling Rules in the AWS X-Ray Console** is incorrect because sampling rules just tell the X-Ray SDK how many requests to record for a set of criteria.    **References:**  <https://docs.aws.amazon.com/xray/latest/devguide/xray-console-filters.html>  <https://docs.aws.amazon.com/xray/latest/devguide/xray-api-segmentdocuments.html>  <https://docs.aws.amazon.com/xray/latest/api/API_GetTraceSummaries.html>    **Check out this AWS X-Ray Cheat Sheet:**  <https://tutorialsdojo.com/aws-x-ray/> | | | | | | |
| 4 | A developer is utilizing AWS X-Ray to generate a visual representation of the requests flowing through their enterprise web application. Since the application interacts with multiple services, all requests must be traced in X-Ray, including any downstream calls made to AWS resources.  Which of the following actions should the developer implement for this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:42 |  |
| * Use AWS X-Ray SDK to upload a trace segment by executing PutTraceSegments API. * Install AWS X-Ray on the different services that communicate with the application including the AWS resources that the application calls. * **Pass multiple trace segments as a parameter of PutTraceSegments API.** * **Use X-Ray SDK to generate segment documents with subsegments and send them to the X-Ray daemon, which will buffer them and upload to the X-Ray API in batches.**   You can send trace data to X-Ray in the form of segment documents. A **segment document** is a JSON formatted string that contains information about the work that your application does in service of a request. Your application can record data about the work that it does itself in segments or work that uses downstream services and resources in subsegments.  A segment document can be up to 64 kB and contain a whole segment with subsegments, a fragment of a segment that indicates that a request is in progress, or a single subsegment that is sent separately. You can send segment documents directly to X-Ray by using the PutTraceSegments API. An alternative is, instead of sending segment documents to the X-Ray API, you can send segments and subsegments to an X-Ray daemon, which will buffer them and upload to the X-Ray API in batches. The X-Ray SDK sends segment documents to the daemon to avoid making calls to AWS directly. This is the correct option among the choices.    Hence, **using X-Ray SDK to generate segment documents with subsegments and sending them to the X-Ray daemon, which will buffer them and upload to the X-Ray API in batches** is the correct answer in this scenario.  **Using AWS X-Ray SDK to upload a trace segment by executing PutTraceSegments API**is incorrect because you should upload the segment documents with subsegments instead. A trace segment is just a JSON representation of a request that your application serves.  **Installing AWS X-Ray on the different services that communicate with the application including the AWS resources that the application calls**is incorrect because you cannot run a trace on the application and the services at the same time as this will produce two different results. You simply have to send the segment documents with subsegments to get the information about downstream calls that your application makes to AWS resources.  **Passing multiple trace segments as a parameter of PutTraceSegments API** is incorrect because, contrary to the API's name, you have to upload **segment** documents and not trace segments. The API has a single parameter: TraceSegmentDocuments, that takes a list of JSON segment documents.    **References:**  <https://docs.aws.amazon.com/xray/latest/devguide/xray-api-segmentdocuments.html>  <https://docs.aws.amazon.com/xray/latest/devguide/xray-api-sendingdata.html>  <https://docs.aws.amazon.com/xray/latest/api/API_PutTraceSegments.html>    **Check out this AWS X-Ray Cheat Sheet:**  <https://tutorialsdojo.com/aws-x-ray/> | | | | | | |
| 5 | A developer is managing a real-time fraud detection system that ingests a stream of data using Amazon Kinesis. The system works well with millisecond end-to-end latency, but the allocated shards are way underutilized based on the performance data in CloudWatch.  Which of the following is the MOST suitable solution to reduce the cost and capacity of the stream?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:28 |  |
| * **Merge cold shards** * Split cold shards * Merge hot shards * Split hot shards   The purpose of resharding in Amazon Kinesis Data Streams is to enable your stream to adapt to changes in the rate of data flow. You split shards to increase the capacity (and cost) of your stream. You merge shards to reduce the cost (and capacity) of your stream.  One approach to resharding could be to split every shard in the stream—which would double the stream's capacity. However, this might provide more additional capacity than you actually need and therefore create unnecessary costs.    You can also use metrics to determine which are your "hot" or "cold" shards, that is, shards that are receiving much more data, or much less data, than expected. You could then selectively split the hot shards to increase capacity for the hash keys that target those shards. Similarly, you could ***merge cold shards***to make better use of their unused capacity.  You can obtain some performance data for your stream from the Amazon CloudWatch metrics that Kinesis Data Streams publishes. However, you can also collect some of your own metrics for your streams. One approach would be to log the hash key values generated by the partition keys for your data records. Recall that you specify the partition key at the time that you add the record to the stream.  Hence, the correct answer is to **merge cold shards** to reduce the capacity and the cost of running your Kinesis Data Stream.  **Splitting cold shards** is incorrect because a cold shard is the one that receives fewer data which means that you have to merge them to reduce the capacity rather than split them.  **Merging hot shards**is incorrect. Although merging shards is correct, the type of shard to be merged is wrong. A hot shard is the one that receives more data in the stream. Merging hot shards could potentially overload the newly merged shard with a high volume of data, causing a bottleneck in processing and degrading the overall performance of the stream.  **Splitting hot shards**is incorrect because this will actually further increase both the cost and capacity of the stream rather than reduce it. Moreover, there are no hot shards in the stream since the scenario specifically mentioned that the shards are way underutilized.    **References:**  <https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-resharding-strategies.html>  <https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-resharding.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/> | | | | | | |
| 6 | A developer recently deployed a serverless application, which consists of a Lambda function, API Gateway, and DynamoDB using the sam deploy CLI command. The Lambda function is invoked through the API Gateway and then processes and stores the data in a DynamoDB table with an average time of 20 minutes. However, the IT Support team noticed that there are several terminated Lambda invocations that happen every day, which is causing data discrepancies.  Which of the following options is the MOST likely root cause of this problem?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:01:48 |  |
| * The serverless application should be deployed using the sam publish CLI command instead. * The Lambda function contains a recursive code and has been running for over 15 minutes. * The concurrent execution limit has been reached. * **The failed Lambda invocations have been running for over 15 minutes and reached the maximum execution time.**   A Lambda function consists of code and any associated dependencies. In addition, a Lambda function also has configuration information associated with it. Initially, you specify the configuration information when you create a Lambda function. Lambda provides an API for you to update some of the configuration data.  You pay for the AWS resources that are used to run your Lambda function. To prevent your Lambda function from running indefinitely, you specify a **timeout**. When the specified timeout is reached, AWS Lambda terminates execution of your Lambda function. It is recommended that you set this value based on your expected execution time.    Take note that you can invoke a Lambda function synchronously either by calling the Invoke operation or by using an AWS SDK in your preferred runtime. If you anticipate a long-running Lambda function, your client may time out before function execution completes. To avoid this, update the client timeout or your SDK configuration.  The default timeout is 3 seconds and the maximum execution duration per request in AWS Lambda is 900 seconds, which is equivalent to 15 minutes. Hence, the most likely root cause in this scenario is that **the failed Lambda invocations have been running for over 15 minutes and reached the maximum execution time**.  The option that says: **The serverless application should be deployed using the sam publish CLI command instead** is incorrect as this CLI command just publishes an AWS SAM application to the AWS Serverless Application Repository. The fact that some invocations of the Lambda function work fine means that the deployment is successful. Hence, there is no issue on the deployment process of the serverless application but only on its maximum execution time.  The option that says: **The concurrent execution limit has been reached** is incorrect because, by default, the AWS Lambda limits the total concurrent executions across all functions within a given region to 1000. By setting a concurrency limit on a function, Lambda guarantees that allocation will be applied specifically to that function, regardless of the amount of traffic processing the remaining functions. If that limit is exceeded, the function will be throttled but not terminated, which is in contrast with what is happening in the scenario.  The option that says: **The Lambda function contains a recursive code and has been running for over 15 minutes** is incorrect because having a recursive code in your Lambda function does not directly result to an abrupt termination of the function execution. This is a scenario wherein the function automatically calls itself until some arbitrary criteria is met. This could lead to an unintended volume of function invocations and escalated costs, but not an abrupt termination because Lambda will throttle all invocations to the function.    **Reference:**  <https://docs.aws.amazon.com/lambda/latest/dg/limits.html>  <https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html>    **Check out this AWS Lambda Cheat Sheet:**  <https://tutorialsdojo.com/aws-lambda/> | | | | | | |
| 7 | A leading financial company has recently deployed its application to AWS using Lambda and API Gateway. However, they noticed that all metrics are being populated in their CloudWatch dashboard except for CacheHitCount and CacheMissCount.  What could be the MOST likely cause of this issue?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:13 |  |
| * They have not provided an IAM role to their API Gateway yet. * **API Caching is not enabled in API Gateway.** * The provided IAM role to their API Gateway only has read access but no write privileges to CloudWatch. * API Gateway Private Integrations has not been configured yet.   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. CacheMissCount tracks the number of requests served from the backend in a given period, when API caching is enabled. On the other hand, CacheHitCount track the number of requests served from the API cache in a given period.  Hence, the root cause of this issue is that the **API Caching is not enabled in API Gateway**which is why the ***CacheHitCount*** and ***CacheMissCount*** metrics are not populated.  The option that says: **they have not provided an IAM role to their API Gateway yet** is incorrect because, in the first place, the scenario already mentioned that all metrics are being populated in their CloudWatch dashboard except for two metrics. This implies that some of the metrics are populated which means that the API Gateway already has an IAM Role associated with it.  The option that says: **the provided IAM role to their API Gateway only has read access but no write privileges to CloudWatch** is incorrect because just as what is mentioned above, there is no issue with the IAM Role since all metrics are being populated except only for CacheHitCount and CacheMissCount. This means that the associated IAM Role already has *write* privileges to write logs to CloudWatch to begin with. The only reason why those two metrics are not being populated is that the API Caching is not enabled.  The option that says: **API Gateway Private Integrations has not been configured yet**is incorrect because this feature only makes it easier to expose your HTTP/HTTPS resources behind an Amazon VPC for access by clients outside of the VPC.    **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/) | | | | | | |
| 8 | A leading insurance firm is hosting its customer portal in Elastic Beanstalk, which has an RDS database in AWS. The support team in your company discovered a lot of SQL injection attempts and cross-site scripting attacks on the portal, which is starting to affect the production environment.  Which of the following services should you implement to mitigate this attack?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:27 |  |
| * Amazon Guard​Duty * AWS Firewall Manager * Network Access Control List * **AWS WAF**   **AWS WAF** is a web application firewall that lets you monitor the HTTP and HTTPS requests that are forwarded to an Amazon API Gateway API, Amazon CloudFront or an Application Load Balancer. AWS WAF also lets you control access to your content. Based on conditions that you specify, such as the IP addresses that requests originate from or the values of query strings, API Gateway, CloudFront or an Application Load Balancer responds to requests either with the requested content or with an HTTP 403 status code (Forbidden). You also can configure CloudFront to return a custom error page when a request is blocked.    At the simplest level, AWS WAF lets you choose one of the following behaviors:  **Allow all requests except the ones that you specify** – This is useful when you want CloudFront or an Application Load Balancer to serve content for a public website, but you also want to block requests from attackers.  **Block all requests except the ones that you specify** – This is useful when you want to serve content for a restricted website whose users are readily identifiable by properties in web requests, such as the IP addresses that they use to browse to the website.  **Count the requests that match the properties that you specify** – When you want to allow or block requests based on new properties in web requests, you first can configure AWS WAF to count the requests that match those properties without allowing or blocking those requests. This lets you confirm that you didn't accidentally configure AWS WAF to block all the traffic to your website. When you're confident that you specified the correct properties, you can change the behavior to allow or block requests.  Hence, the correct answer in this scenario is **AWS WAF.**  **Amazon Guard​Duty** is incorrect because this is just a threat detection service that continuously monitors malicious activity and unauthorized behavior to protect your AWS accounts and workloads.  **AWS Firewall Manager**is incorrect because this just simplifies your AWS WAF and AWS Shield Advanced administration and maintenance tasks across multiple accounts and resources.  **Network Access Control List** is incorrect because this is an optional layer of security for your VPC that acts as a firewall for controlling traffic in and out of one or more subnets.    **References:**  <https://aws.amazon.com/waf/>  <https://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html>  <https://aws.amazon.com/blogs/security/three-most-important-aws-waf-rate-based-rules/>    **Check out this AWS WAF Cheat Sheet:**  <https://tutorialsdojo.com/aws-waf/> | | | | | | |
| 9 | A developer has recently deployed an application, which is hosted in an Auto Scaling group of EC2 instances and processes data from an Amazon Kinesis Data Stream. Each of the EC2 instances has exactly one KCL worker processing one Kinesis data stream which has 10 shards. Due to performance issues, the systems operations team has resharded the data stream to increase the number of open shards to 20.  What is the maximum number of running EC2 instances that should ideally be kept to maintain application performance?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:02:15 |  |
| * 10 * **20** * 30 * 40   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 **20**, to match the number of open shards with a ratio of 1:1.  Although you can launch **10** 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.  Launching **30**or **40** 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 20.    **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/> | | | | | | |
| 10 | A company has an application hosted in an ECS Cluster that heavily uses an RDS database. A developer needs to closely monitor how the different processes on a DB instance use the CPU, such as the percentage of the CPU bandwidth or the total memory consumed by each process to ensure application performance.  Which of the following is the MOST suitable solution that the developer should implement?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:20 |  |
| * **Use Enhanced Monitoring in RDS.** * Develop a shell script that collects and publishes custom metrics to CloudWatch which tracks the real-time CPU Utilization of the RDS instance. * Use CloudWatch to track the CPU Utilization of your database. * **Track the CPU% and MEM% metrics which are readily available in the Amazon RDS console.**   **Amazon RDS** provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view the metrics for your DB instance using the console or consume the Enhanced Monitoring JSON output from CloudWatch Logs in a monitoring system of your choice. By default, Enhanced Monitoring metrics are stored in the CloudWatch Logs for 30 days. To modify the amount of time the metrics are stored in the CloudWatch Logs, change the retention for the RDSOSMetrics log group in the CloudWatch console.    Take note that there are certain differences between CloudWatch and Enhanced Monitoring Metrics. CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance, and Enhanced Monitoring gathers its metrics from an agent on the instance. As a result, you might find differences between the measurements, because the hypervisor layer performs a small amount of work.  The differences can be greater if your DB instances use smaller instance classes because then there are likely more virtual machines (VMs) that are managed by the hypervisor layer on a single physical instance. Enhanced Monitoring metrics are useful when you want to see how different processes or threads on a DB instance use the CPU.  Hence, the correct answer is to **use Enhanced Monitoring in RDS**.  **Developing a shell script that collects and publishes custom metrics to CloudWatch which tracks the real-time CPU Utilization of the RDS instance** is incorrect. Although you can use Amazon CloudWatch Logs and CloudWatch dashboard to monitor the CPU Utilization of the database instance, using CloudWatch alone is still not enough to get the specific percentage of the CPU bandwidth and total memory consumed by each database process. The data provided by CloudWatch is not as detailed as compared with the Enhanced Monitoring feature in RDS. Take note as well that you do not have direct access to the instances/servers of your RDS database instance, unlike with your EC2 instances where you can install a CloudWatch agent or a custom script to get CPU and memory utilization of your instance.  **Using CloudWatch to track the CPU Utilization of your database** is incorrect. Although you can use Amazon CloudWatch to monitor the CPU Utilization of your database instance, it does not provide the percentage of the CPU bandwidth and total memory consumed by each database process in your RDS instance. Take note that CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance, while RDS Enhanced Monitoring gathers its metrics from an agent on the instance.  **Tracking the CPU% and MEM% metrics which are readily available in the Amazon RDS console** is incorrect because these metrics are not readily available in the Amazon RDS console, which is contrary to what is being stated in this option.    **References:**  <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Monitoring.OS.html#USER_Monitoring.OS.CloudWatchLogs>  <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/MonitoringOverview.html#monitoring-cloudwatch>    **Check out these Amazon CloudWatch and RDS Cheat Sheets:**  <https://tutorialsdojo.com/amazon-cloudwatch/>  <https://tutorialsdojo.com/amazon-relational-database-service-amazon-rds/> | | | | | | |
| 11 | A company is using a combination of CodeCommit, CodeBuild, CodePipeline and CodeDeploy services for its continuous integration and continuous delivery (CI/CD) pipeline on AWS. They want someone to perform a code review before a revision is allowed into the next stage of a pipeline. If the action is approved, the pipeline execution resumes but if it is not then the pipeline execution will not proceed.  Which of the following is the MOST suitable solution to implement in this scenario?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:02:04 |  |
| * **Implement a manual approval actions configuration in CodePipeline. Send the approval request to an SNS Topic.** * Remodel the pipeline using AWS Serverless Application Model (AWS SAM) * Implement a manual approval actions configuration in CodePipeline. Send the approval request to an SQS Queue. * Split the processes into different Task states using Step Functions. Use a Wait state to set a timeout for approval.   In AWS CodePipeline, you can add an approval action to a stage in a pipeline at the point where you want the pipeline execution to stop so that someone with the required AWS Identity and Access Management permissions can approve or reject the action.  If the action is approved, the pipeline execution resumes. If the action is rejected - or if no one approves or rejects the action within seven days of the pipeline reaching the action and stopping - the result is the same as an action failing, and the pipeline execution does not continue.  You might use manual approvals for these reasons:  - You want someone to perform a code review or change management review before a revision is allowed into the next stage of a pipeline.  - You want someone to perform manual quality assurance testing on the latest version of an application, or to confirm the integrity of a build artifact, before it is released.  - You want someone to review new or updated text before it is published to a company website.    You can configure an approval action to publish a message to an Amazon Simple Notification Service topic when the pipeline stops at the action. Amazon SNS delivers the message to every endpoint subscribed to the topic. You must use a topic created in the same AWS region as the pipeline that will include the approval action. When you create a topic, it is recommended that you give it a name that will identify its purpose, in formats such as tutorialsdojoManualApprovalPHL-us-east-2-approval.  Hence, the correct answer is to **Implement a manual approval actions configuration in CodePipeline. Send the approval request to an SNS Topic.**  The option that says: **Remodel the pipeline using AWS Serverless Application Model (AWS SAM)** is incorrect because this service is just a framework for building serverless applications, not a replacement for a CI/CD pipeline.  The option that says: **Implement a manual approval actions configuration in CodePipeline. Send the approval request to an SQS Queue**is incorrect. Although setting up a manual approval is valid, the use of SQS is wrong because it doesn't have an integration with manual approval actions. Use SNS instead to send the approval action emails to the recipient who will either approve or deny the action.  The option that says: **Split the processes into different Task states using Step Functions. Use a Wait state to set a timeout for approval**is incorrect as this would add unnecessary complexity to the CI/CD pipeline. The requirement in the scenario can be achieved using the built-in manual approval actions in CodePipeline.    **References:**  <https://docs.aws.amazon.com/codepipeline/latest/userguide/approvals.html>  [https://docs.aws.amazon.com/codepipeline/latest/userguide/actions.html](https://docs.aws.amazon.com/codepipeline/latest/userguide/approvals.html)    **Check out this AWS CodePipeline Cheat Sheet:**  <https://tutorialsdojo.com/aws-codepipeline/>    **Comparison of AWS Services Cheat Sheets:**  <https://tutorialsdojo.com/comparison-of-aws-services/> | | | | | | |
| 12 | A web application hosted in Elastic Beanstalk has a configuration file named .ebextensions/debugging.config which has the following content:  option\_settings: aws:elasticbeanstalk:xray: XRayEnabled: true  For its database tier, it uses RDS with Multi-AZ deployments configuration and Read Replicas. There is a new requirement to record calls that your application makes to RDS and other internal or external HTTP web APIs. The tracing information should also include the actual SQL database queries sent by the application, which can be searched using the filter expressions in the X-Ray Console.  Which of the following should you do to satisfy the above task?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 1 | 0 | 00:00:50 |  |
| * Add metadata in the segment document. * Add annotations in the segment document. * Add metadata in the subsegment section of the segment document. * **Add annotations in the subsegment section of the segment document.**   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 is 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.  A trace segment is a JSON representation of a request that your application serves. A trace segment records information about the original request, information about the work that your application does locally, and subsegments with information about downstream calls that your application makes to AWS resources, HTTP APIs, and SQL databases.  Hence, **adding annotations in the subsegment section of the segment document** is the correct answer.  **Adding annotations in the segment document**is incorrect. Although the use of annotations is correct, you have to add this in the ***subsegment*** section of the *segment* document since you want to trace the downstream call to RDS and not the actual request to your application.  **Adding metadata in the 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 since this can't be picked up by filter expressions in the X-Ray Console. You have to use annotations instead. In addition, you have to add this in the *subsegment* section of the *segment* document since you want to trace the downstream call to RDS and not the actual request to your application.  **Adding metadata in the subsegment section of the segment document** is incorrect because, just as mentioned above, metadata is just 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/> | | | | | | |
| 13 | A developer is planning to use the AWS Elastic Beanstalk console to run the AWS X-Ray daemon on the EC2 instances in her application environment. She will use X-Ray to construct a service map to help identify issues with her application and to provide insight on which application component to optimize. The environment is using a default Elastic Beanstalk instance profile.  Which IAM managed policy does Elastic Beanstalk use for the X-Ray daemon to upload data to X-Ray?  [(view)](https://portal.tutorialsdojo.com/courses/aws-certified-developer-associate-practice-exams/) | 1 | 0 | 1 | 00:01:09 |  |
| * AWSXrayReadOnlyAccess * **AWSXRayElasticBeanstalkWriteAccess** * AWSXrayFullAccess * **AWSXRayDaemonWriteAccess**   You can use **AWS Identity and Access Management (IAM)** to grant X-Ray permissions to users and compute resources in your account. IAM controls access to the X-Ray service at the API level to enforce permissions uniformly, regardless of which client (console, AWS SDK, AWS CLI) your users employ. To use the X-Ray console to view service maps and segments, you only need read permissions. To enable console access, add the AWSXrayReadOnlyAccess managed policy to your IAM user. For local development and testing, create an IAM user with read and write permissions. Generate access keys for the user and store them in the standard AWS SDK location. You can use these credentials with the X-Ray daemon, the AWS CLI, and the AWS SDK.    To deploy your instrumented app to AWS, create an IAM role with write permissions and assign it to the resources running your application. AWSXRayDaemonWriteAccess includes permission to upload traces, and some read permissions as well to support the use of sampling rules.  The read and write policies do not include permission to configure encryption key settings and sampling rules. Use AWSXrayFullAccess to access these settings, or add configuration APIs in a custom policy. For encryption and decryption with a customer-managed key that you create, you also need permission to use the key.  On supported platforms, you can use a configuration option to run the X-Ray daemon on the instances in your environment. You can enable the daemon in the Elastic Beanstalk console or by using a configuration file. To upload data to X-Ray, the X-Ray daemon requires IAM permissions in the AWSXRayDaemonWriteAccess managed policy. These permissions are included in the Elastic Beanstalk instance profile.  Hence, the correct answer is the **AWSXRayDaemonWriteAccess** managed policy.  **AWSXrayReadOnlyAccess** is incorrect because this policy is primarily used if you just want a read-only access to X-Ray.  **AWSXrayFullAccess** is incorrect. Although this can provide the required access to the daemon, this is not being used in Elastic Beanstalk as it does not abide by the standard security advice of granting the least privilege.  **AWSXRayElasticBeanstalkWriteAccess** is incorrect because this is not an available managed policy.    **References:**  <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-configuration-debugging.html>  <https://docs.aws.amazon.com/xray/latest/devguide/xray-permissions.html>  <https://docs.aws.amazon.com/xray/latest/devguide/security.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/> | | | | | | |
| Sub-Total: | | 13 | 9 (69.23%) | 4 (30.77%) | 00:16:59 | **69.23%** |
|  | | | | | | |
|  | Total | 65 | 48 (73.85%) | 17 (26.15%) | 01:12:41 | **73.85%** |