Question 2: Incorrect

You are a developer making programmatic API calls to AWS KMS. You have security compliance requests that require the use of encryption keys to encrypt large amounts of data using a practice known as envelope encryption. Which of the following API actions should you call?

* Decrypt
* Encrypt
* RetrieveCMK(Incorrect)
* GenerateDataKey(Correct)

#### Explanation

Correct answer - "GenerateDataKey" : **Envelope encryption is the practice of encrypting plaintext data with a data key, and then encrypting the data key under another key.** GenerateDataKey returns a data encryption key that you can use in your application to encrypt data locally.

Encrypt" - You don't need use this operation to encrypt a data key, GenerateDataKey returns the a data encryption key

"RetrieveCMK" - Action does not exist

"Decrypt" - Decrypts ciphertext

For more information visit

<https://docs.aws.amazon.com/kms/latest/APIReference/API_GenerateDataKey.html>

→ Check KMS Doc

AWS Key management service (AWS KMS) is an encryption and key management web service.

\*This is the guide that describes the operation that you can call programmatically\*

--High recommendation to use the AWS SDKs to make programmatic api calls to AWS KMS--

The commonly used api operations

* Encrypt
  + Encrypts plaintext into ciphertext by using a customer master key (CMK). The Encrypt operation has two primary use cases:
    - YOu can encrypt small amounts of arbitrary data, suca as a personal identifier or database password or other sensitive information
    - YOu can use the Encrypt operation to move encrypted data from one AWS region to another
* Decrypt
  + Decrypts ciphertext that was encrypted by a AWS KMS customer master key (CMK using any of the following operations:
    - Encrypt
    - GenerateDataKEy
    - GenerateDataKeyPair
    - GenerateDataKeyWithoutPlaintext
    - GenerateDataKeyPairWithoutPlaintext
  + You can use this operation to decrypt a ciphertext that was encrypted under a symmetric or asymmetric CMK. When the CMK is asymmetric, you must specify the CMK and the encryption algorithm that was used to encrypt the ciphertext.
* GenerateDataKey
  + Generates a unique symmetric data key for client-side encryption
  + This operation returns a plaintext copy of the data key and a copy that is encrypted under a customer master key (CMK) that you specify. You use the plaintext key to encrypt your data outside of aws kms and store the encrypted data key with the encrypted data.
* GenerateDataKeyWithoutPlaintext
  + Generates a unique asymmetric data key pair.
  + The GenerateDataKeyWithoutPlaintext operation returns a plaintext public key and a copy of the private key that is encrypted under the symmetric cmk you specify.
  + The difference between this operation and the generateDataKey operation is that this operation does not return a plaintext private key

Envelope Encryption

* 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 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 that you can decrypt the keys and your data.
* This TOP\_LEVEL PLAINTEXT KEY encryption key is known as the MASTER KEY

AWS KMS

* Helps you to protect the 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 the AWS KMS via:
  + The programmatic access
  + Or the AWS SDK for accessing it

Envelope encryption offers several benefits:

* Protecting data keys
  + When you encrypt a data key, you don’t have to worry about storing the encrypted data key, because the data key is inherently protected by encryption. You can safely encrypt data keys alongside encrypted data.
* Encrypting the same data under multiple master keys
  + Encryption operations can be time consuming, particularly when the data being encrypted are lore objects. Instead of re-encrypting raw data multiple times with different keys, you can re-encrypt only the data keys that protect the raw data
* Combining the strengths of multiple algorithms
  + In general, symmetric key algorithms are faster and produce smaller cipher texts than public key algorithms. But public key algorithms provide inherent separation of roles and easier key management. Envelope encryption lets you combine the strengths of each strategy.

Question 4: Incorrect

You have deployed a traditional 3-tier web application architecture with a Classic Load Balancer, an Auto Scaling group and an Amazon Relational Database Service (RDS) database. Users are reporting daily that they have to re-authenticate into the website often. The session information is stored in-memory for each application instance. What should you do to make the application tier stateless and outsource the session information?

* Use Elastic IP
* Enable RDS read replicas
* Enable Load Balancer stickiness(Incorrect)
* Add an ElastiCache Cluster(Correct)

#### Explanation

Correct answer - "Add an ElastiCache Cluster" : In order to provide a shared data storage for sessions that can be accessible from any individual web server, you can abstract the HTTP sessions from the web servers themselves. A common solution for this is to leverage an ElastiCache service offering which is an In-Memory Key/Value store such as Redis and Memcached.

"Enable RDS read replicas" - Read-replicas syncs from the master database so there will be inconsistencies in using this approach

"Enable Load Balancer stickiness" - With sticky sessions feature, it instructs the load balancer to route repeated requests to the same EC2 instance whenever possible. It can be used but ElastiCache is more scalable

"Use Elastic IP" - An Elastic IP is a way to give your server a static IP address but won't really solve your issue with sessions

For more information visit <https://aws.amazon.com/caching/session-management/>

Session Management

* Manage user sessions including storing those sessions locally to the node responding to the HTTP request
* Designating a layer in the architecture which can store those sessions in a scalable and robust manner.
* Two common approaches:
  + Sticky Sessions
  + Distributed Cache for your session management

Stick Sessions (Session Affinity) with Local Session Caching:

* Routing a site user to the particular web server that is managing that individual’s user seesion.
* The sessions can be validated by a number of methods EG: client-side cookies or configurable duration parameters that can be set at the load balancer which routes requests to the web servers
* ADVANTAGES
  + Cost effective because they are storing session the same web servers running applications
  + Fast, because it eliminates network latency.
* DRAWBACKS
  + In the event of failure, you are likely to lose the sessions that were resident on the failed node
  + In the event the number of the web servers change, for example a scale-up scenario, it’s possible that the traffic may be unequally spread across the web servers as active sessions may exist on particular servers. If not mitigated properly, this can hinder the scalability of your applications.

Distributed Session Management

* In order to address scalability and to provide a shared data storage for sessions that can be accessible from any individual web server you can abstract the HTTP sessions from the web servers themselves. A common solution for this is to leverage an In-Memory Key/Value store such as Redis and Memcached
* ADVANTAGES
  + Solves scalability
  + Pretty Fast
* DISADVANTAGE
  + Increased costs

Redis

Redis, which stands for Remote Dictionary Server, is a fast, pen-source, in-memory key value data store for use as a database, cache, message broker, and queue.

* How does Redis work
  + All redis data resides i-memory, in contrast to databasesthat store data on disk or SSDs. By elminating the need to access disks, in-memory data stores such as Redis avoid seek time delays and can access data in microseconds. Redis features versatile data structures, high availability, geospatial, Lua scripting, transactions, on-disk persistence, and cluster support making it simpler to build real-time internet scale apps.

Redis Vs Memcached

* Both redis and memcached are in-memory, open source data stores. Memecached, a high performance distributed memory cache service, is designed for simplicity while redis offers a rich set of features that make it effective for a wide range of use cases.
  + <https://aws.amazon.com/elasticache/redis-vs-memcached/>

Memcache

* Unlike databases that store data on disk or SSDs, Memcached keeps its data in memory. By eliminating the need to access disks, in-memory key-value stores such as Memcached avoid seek time delays and can access data in microseconds. Memcached is also distributed, meaning that it is easy to scale out by adding new nodes. And since Memcached is multithreaded, you can easily scale up compute capacity. As a result of its speed and scalability as well as its simple design, efficient memory management, and API support for most popular languages Memcached is a popular choice for high-performance, large-scale caching use cases.

Question 5: Incorrect

You have an Application Load Balancer listening on port 80 registered with a single EC2 instance also listening on port 80. You plan on changing the listener. Which of the following options is not a supported listener?

* Websocket(Incorrect)
* HTTP
* HTTPS
* TCP(Correct)

#### Explanation

Correct answer - "TCP" : The Application Load Balancer supports following protocols: WebSocket and HTTP/2.

"HTTP"

"HTTPS"

"Websocket"

For more information visit <https://aws.amazon.com/blogs/aws/new-aws-application-load-balancer/>

The reason why this is incorrect is that application load balancers do not deal with TCP. That being said websockets use TCP for therm. If you want to be working with TCP you need to use a network load balancer instead of an application load balancer.

Question 6: Incorrect

An Amazon Simple Storage Service (S3) bucket holds images uploaded from a website. When new objects are created in the bucket an AWS Lambda function is invoked based on the function ARN configured. **When new function versions are promoted changes should not have to be made in the configuration to point to the new function ARN. How can you accomplish this?**

* Enable X-Ray integration
* Disable AWS Lambda versioning(Incorrect)
* Use environment variables
* Use AWS Lambda aliases(Correct)

#### Explanation

Correct answer - "Use AWS Lambda aliases" : AWS Lambda also supports creating aliases for each of your Lambda function versions. Conceptually, an AWS Lambda alias is a pointer to a specific Lambda function version. It is also a resource similar to a Lambda function, and each alias has a unique ARN. Each alias maintains an ARN for the function version to which it points.

"Disable AWS Lambda versioning" - Although possible, aliases are a great way to shift traffic when new versions are available

"Use environment variables" - Environment variables enable you to dynamically pass settings to your function code

"Enable X-Ray integration" - You can use AWS X-Ray to trace your AWS Lambda functions

For more information visit <https://docs.aws.amazon.com/lambda/latest/dg/versioning-aliases.html>

AWS Lambda function versions <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-version.html>

* You can use versions to manage the deployment of your AWS Lambda functions. For example, you can publish a new version to manage the deployment of your AWS Lambda functions. For example, you can publish a new version of a function for beta testing without affecting the users of the stable production version
* The system creates a new version of your Lambda function each time that you publish the function. The new version is a copy of the unpublished version of the function. The function version includes the following information:
  + The function code and all associated dependencies
  + The Lambda runtime that executes the function
  + All of the function settings, including the environment variables.
  + A unique amazon resource name (ARN) to identify this version of the function.

CLOUDFORMATION

{

"Type" : "AWS::Lambda::Version",

"Properties" : {

"[CodeSha256](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-version.html#cfn-lambda-version-codesha256)" : *String*,

"[Description](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-version.html#cfn-lambda-version-description)" : *String*,

"[FunctionName](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-version.html#cfn-lambda-version-functionname)" : *String*,

"[ProvisionedConcurrencyConfig](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-version.html#cfn-lambda-version-provisionedconcurrencyconfig)" : [*ProvisionedConcurrencyConfiguration*](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-lambda-version-provisionedconcurrencyconfiguration.html)

}

}

Using AWS Lambda environment variables

* You can use environment variables to store secrets securely and adjust your functions behavior without having to update code. An environment variable is a pair of strings that are stored in a function’s version-specific configuration. The Lambda runtime makes environment variables available to your code and sets additional environment variables that contain information about the function and invocation requests
* Use environment variables to pass environment-specific settings to your code. For example, you can have two functions with the same code but different configuration. One function connects to a test database and the other connects to a production database. In this situation, you use environment variables to the ll the function the hostname and other connection details for the database. You might also set a environment variable to configure your test environment to use more verbose logging or more detailed tracing
* To store environment variables in lambda cloudformation use the **Environment** resource
  + API\_KEY: 12343456467567
* To call the environmental variables
  + process.env.ENV\_VAR\_NAME

AWS Lambda function aliases

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html>

* You can create one or more aliases for your AWS Lambda function. A Lambda alias is like a pointer to a specific Lambda function version. Users can access the function version using the alias ARN
* Unique ARN
* Point only to a function version not another alias
* Can update alias to point at a different version
* These event sources maintain a mapping that identifies the function to invoke when events occur. If you specify a Lambda function alias in the mapping configuration, you don't need to update the mapping when the function version changes.

CLOUDFORMATION

* Type: AWS::Lambda::Alias
* Properties:
* [Description](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-description): *String*
* [FunctionName](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-functionname): *String*
* [FunctionVersion](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-functionversion): *String*
* [Name](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-name): *String*
* [ProvisionedConcurrencyConfig](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-provisionedconcurrencyconfig):
* [*ProvisionedConcurrencyConfiguration*](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-lambda-alias-provisionedconcurrencyconfiguration.html)
* [RoutingConfig](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-alias.html#cfn-lambda-alias-routingconfig):
* [*AliasRoutingConfiguration*](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-lambda-alias-aliasroutingconfiguration.html)

NOTES

* Setting up a proper lambda from an API gateway:
  + Create three lambda aliases (test, dev, prod)
  + Each of these aliases will point to a different version of the same lambda
  + Each of the versions will dynamically look at a different stage object
  + Those objects will have the same property names, and those property name values will provide the correct information for the different states
  + This can be done by setting the environmental variables for the versions in CFN
* EX
  + Resources:
  + function:
  + Type: AWS::Lambda::Function
  + Properties:
  + Handler: index.handler
  + Role: arn:aws:iam::123456789012:role/lambda-role
  + Code:
  + ZipFile: |
  + exports.handler = function(event){
  + console.log(JSON.stringify(event, null, 2))
  + const response = {
  + statusCode: 200,
  + body: JSON.stringify('Hello again from Lambda!')
  + }
  + return response
  + }
  + Runtime: nodejs12.x
  + TracingConfig:
  + Mode: Active
  + version:
  + Type: AWS::Lambda::Version
  + Properties:
  + FunctionName: !Ref function
  + Description: v1
  + newVersion:
  + Type: AWS::Lambda::Version
  + Properties:
  + FunctionName: !Ref function
  + Description: v2
  + alias:
  + Type: AWS::Lambda::Alias
  + Properties:
  + FunctionName: !Ref function
  + FunctionVersion: !GetAtt newVersion.Version
  + Name: BLUE
  + RoutingConfig:
  + AdditionalVersionWeights:
  + - FunctionVersion: !GetAtt version.Version
  + FunctionWeight: 0.5