Question 1: Incorrect

Your manager assigned you a task of implementing server-side encryption with customer-provided encryption keys (SSE-C) to your S3 bucket, which will allow you to set your own encryption keys. Amazon S3 will manage both the encryption and decryption process using your key when you access your objects, which will remove the burden of maintaining any code to perform data encryption and decryption.

To properly upload data to this bucket, which of the following headers must be included in your request?

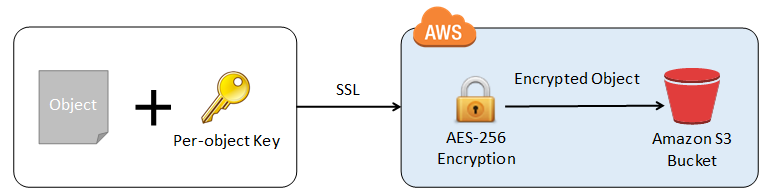
* x-amz-server-side-encryption, x-amz-server-side-encryption-customer-key and x-amz-server-side-encryption-customer-key-MD5 headers(Incorrect)
* x-amz-server-side-encryption-customer-key header only
* x-amz-server-side-encryption and x-amz-server-side-encryption-aws-kms-key-id headers
* x-amz-server-side​-encryption​-customer-algorithm, x-amz-server-side-encryption-customer-key and x-amz-server-side-encryption-customer-key-MD5 headers(Correct)

#### 

Server-side encryption is about protecting data at rest.

Using server-side encryption with customer-provided encryption keys (SSE-C) allows you to set your own encryption keys.

With the encryption key you provide as part of your request, Amazon S3 manages both the encryption, as it writes to disks, and decryption, when you access your objects. Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.



When you upload an object, Amazon S3 uses the encryption key you provide to apply AES-256 encryption to your data and removes the encryption key from memory. It is important to note that Amazon S3 does not store the encryption key you provide. Instead, it is stored in a randomly salted HMAC value of the encryption key in order to validate future requests.

The salted HMAC value cannot be used to derive the value of the encryption key or to decrypt the contents of the encrypted object. That means, if you lose the encryption key, you lose the object.

When you retrieve an object, you must provide the same encryption key as part of your request. Amazon S3 first verifies that the encryption key you provided matches, and then decrypts the object before returning the object data to you.

HMAC ( key-hashed message authentication code or hash-based message authentication code)

When using server-side encryption with customer-provided encryption keys (SSE-C), you must provide encryption key information using the following request headers:

**x-amz-server-side-encryption-customer-algorithm** - This header specifies the encryption algorithm. The header value must be "AES256".

**x-amz-server-side-encryption-customer-key** - This header provides the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.

**x-amz-server-side-encryption-customer-key-MD5** - This header provides the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321. Amazon S3 uses this header for a message integrity check to ensure the encryption key was transmitted without error.

Hence, the correct answer is to include the x-amz-server-side​-encryption​-customer-algorithm, x-amz-server-side-encryption-customer-key and x-amz-server-side-encryption-customer-key-MD5 headers on the upload request.

Including the x-amz-server-side-encryption and x-amz-server-side-encryption-aws-kms-key-id headers in the upload request is incorrect because these headers are primarily used in Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS) and not for Server-Side Encryption with Customer-Provided Keys (SSE-C).

Including the x-amz-server-side-encryption, x-amz-server-side-encryption-customer-key and x-amz-server-side-encryption-customer-key-MD5 headers is incorrect because the x-amz-server-side-encryption header is not used in SSE-C encryption. This should be replaced with the x-amz-server-side​-encryption​-customer-algorithm header.

Including just the x-amz-server-side-encryption-customer-key header only is incorrect because you have to include the x-amz-server-side​-encryption​-customer-algorithm and x-amz-server-side-encryption-customer-key-MD5 headers as well to upload the objects to the S3 bucket with SSE-C encryption.

References:

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

<https://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectPUT.html#RESTObjectPUT-responses-examples>

<https://aws.amazon.com/blogs/security/how-to-prevent-uploads-of-unencrypted-objects-to-amazon-s3/>

Check out these Amazon S3 and AWS KMS Cheat Sheets:

<https://tutorialsdojo.com/aws-cheat-sheet-amazon-s3/>

<https://tutorialsdojo.com/aws-cheat-sheet-aws-key-management-service-aws-kms/>

**Headers to use with Server-side encryption with customer provided keys (SSE-C)**

* **X-amz-server-side-encryption-customer-algorithm**
  + This header is to specify the encryption algorithm, the value of which MUST be “AES256”
* **X-amz-server-side-encryption-customer-key**
  + This header is to provide the 256-bit, base-64 encoded encryption key for Amazon S3 to used to encrypt or decrypt your data.
* **x-amz-server-side -encryption-customer-key-MD5**
  + This header is to provide the base64-encoded 128-bit MD5 digest of the encryption key. Amazon S3 uses this header for a message integrity check to ensure that the encryption key was transmitted without error/

**Headers to use with AWS Key Management Service (KMS) customer master keys (CMK) SSE-KMS**

* X-amz-server-side-encryption-aws-kms-key-id
* X-amz-server-side-encryption-context

**Amazon S3 REST APIs that Support SSE-KMS**

* X-amz-server-side-encryption
* X-amz-server-side-encryption-aws-key-id
* X-amz-server-side-encryption-context

**Encryption Context (x-amz-server--side-encryption-context)**

* Used with the the x-amz-server-side-encryption:aws:kms header
* IT is an optional set of key-value pairs that can contain additional contextual information about the data
* Can be any value you want, BUT it has to be in the base64 encoded JSON format
* HOWEVER, since it is not encrypted context data should not contain sensitive data
* IT IS logged with AWS CloudTrail if that service has been enabled

AWS KMS Key ID ( x-amz-server-side-encryption-aws-kms-key-id)

* Used to specify the ID of the customer managed CMK that is being used to protect the data
* If the header x-amz-server-side-encryption:aws:kms but DON”T specify a KEY ID in the header AWS uses the managed CMK in AWS SMK to protect the data
* Only works with symmetric encryption keys, will NOT work with asymmetric encryption keys

Question 3: Incorrect

An application in your development account is running in an AWS Elastic Beanstalk environment which has an attached Amazon RDS database. You noticed that if you terminate the environment, it also brings down the database which hinders you from performing seamless updates with blue-green deployments. This also poses a critical security risk if the company decides to deploy the application in production.

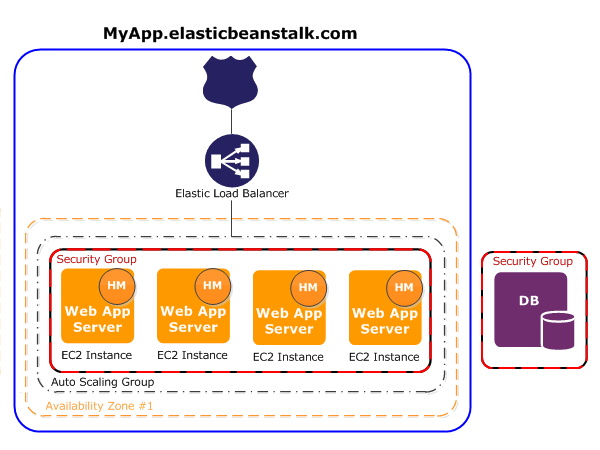
In this scenario, how can you decouple your database instance from your environment without having any data loss?

* ​
* Use the blue / green deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance. Before terminating the old Elastic Beanstalk environment, remove its security group rule first before proceeding.
* (Correct)
* ​
* Use the blue / green deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance and delete the old environment.
* (Incorrect)
* ​
* Use a Canary deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance and delete the old environment.
* ​
* Use a Canary deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and then create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance.

#### Explanation

AWS Elastic Beanstalk provides support for running Amazon Relational Database Service (Amazon RDS) instances in your Elastic Beanstalk environment. This works great for development and testing environments. However, it isn't ideal for a production environment because it ties the lifecycle of the database instance to the lifecycle of your application's environment.

If you haven't used a DB instance with your application before, try adding one to a test environment with the Elastic Beanstalk console first. This lets you verify that your application is able to read environment properties, construct a connection string, and connect to a DB instance before you add Amazon Virtual Private Cloud (Amazon VPC) and security group configuration to the mix.



To decouple your database instance from your environment, you can run a database instance in Amazon RDS and configure your application to connect to it on launch. This enables you to connect multiple environments to a database, terminate an environment without affecting the database, and perform seamless updates with blue-green deployments.

To allow the Amazon EC2 instances in your environment to connect to an outside database, you can configure the environment's Auto Scaling group with an additional security group. The security group that you attach to your environment can be the same one that is attached to your database instance, or a separate security group from which the database's security group allows ingress.

You can connect your environment to a database by adding a rule to your database's security group that allows ingress from the autogenerated security group that Elastic Beanstalk attaches to your environment's Auto Scaling group. However, doing so creates a dependency between the two security groups. Subsequently, when you attempt to terminate the environment, Elastic Beanstalk will be unable to delete the environment's security group because the database's security group is dependent on it.

Hence, the option that says: Use the blue/green deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance. Before terminating the old Elastic Beanstalk environment, remove its security group rule first before proceeding is the correct answer in this scenario.

The option that says: Use the blue/green deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance and delete the old environment is incorrect because although the deployment strategy being used here is valid, the existing security group rule is not yet removed which hinders the deletion of the old environment.

The option that says: Use a Canary deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and enable deletion protection. Create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance and delete the old environment is incorrect because there is no Canary deployment configuration in Elastic Beanstalk. This type of deployment strategy is usually used in Lambda.

The option that says: Use a Canary deployment strategy to decouple the Amazon RDS instance from your Elastic Beanstalk environment. Create an RDS DB snapshot of the database and then create a new Elastic Beanstalk environment with the necessary information to connect to the Amazon RDS instance is incorrect because you should use a blue/green deployment strategy instead. This will also cause a data loss since the deletion protection for the database is not enabled.

References:

<https://aws.amazon.com/premiumsupport/knowledge-center/decouple-rds-from-beanstalk/>

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

Check out this AWS Elastic Beanstalk Cheat Sheet:

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

AWS ELASTIC BEANSTALK

* Allows developers to quickly deploy and manage application sin the AWS Cloud without worrying about the infrastructure that runs those applications.
* Automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring for your applications
* WORKFLOW
  + Create an application in a supported language and possible container
  + Upload the application to Elastic Beanstalk
  + Launch the Environment
  + Manage the Environment ( if you upload a new version start at step two )
* App domain name is in this style
  + subdomain.region.elasticbeanstalk.com

ENVIRONMENT PAGE

* Configuration:
  + Shows the resources provisioned for this environment
  + Also lets you configure some of the provisioned resources
* Health
  + Shows the status and detailed health information about the EC2 instances running your application
* Monitoring
  + Shows the statistics for the environment, such as average latency and CPU utilization.
  + Use this page to create alarms for the metrics that you are monitoring
* Events
  + Shows any informational or error messages from services that this environment is using
* Tags
  + Shows any tags:
    - Key value pairs that are applied to resources in the environment
  + Manage the environment tags from this page

ELASTIC BEANSTALK CONCEPTS

* Application
  + A logical collection of elastic beanstalk components, including environments, version, and environment configurations. (conceptually like a folder)
* Application Versions
  + Refers to a specific labeled iteration of deployable code for a web application
  + A application version points to an amazon s3 object that contains the deployable code
  + Applications can have many versions and each application version is unique
* Environment
  + A version that is deployed on to AWS resources.
  + Each environment runs only a single application version at a time
* Environment tier
  + Determines whether Elastic Beanstalk provisions resources to support an application that handles HTTP requests or an application that pulls tasks from a queue.
  + An application that serves HTTP requests runs in a **web server environment**.
  + An environment that pulls tasks from an amazon SQS queue runs in a **worker environment**
* Environment Configuration
  + Identifies a collection of parameters and settings that define how an environment and is associated resources behave
  + **Configuration Template**
    - A starting point for creating unique environments configurations

ENVIRONMENT TYPES

* Load-balancing, Auto Scaling Environment
  + Automatically starts additional instances to accommodate increasing load on you application
* Single-Instance Environment
  + Contains one Amazon EC2 instance with an Elastic IP address

ENVIRONMENT CONFIGURATIONS

* EC2 virtual machines
  + Configured to run web apps on the platform you choose
* Auto Scaling Group
  + Which ensures that there is always one instance running in a single-instance environment,
  + Allows configuration of the group with a range of instances to run in a load-balanced environment
* Elastic Load Balancing Load Balancer
  + Only used if the Load-Balancing, Auto calling Environment environment type was chosen
  + Distributes traffic among your environment’s instances
* Amazon RDS
  + Provides integration with amazon RDS to help add a database instance to your Elastic Beanstalk Environment
  + When you add a DB instance to your environment, Elastic Beanstalk provides connection information to your application by setting environment properties for the database hostname, port, username password, and database name.
* Environment Properties
  + Used to pass secrets, endpoints, debug settings, and other information to your application
* Amazon SNS
  + Can be configured to notify you of important event that affect your application
* Subdomain of elasticbeanstalk.com
  + When you create an environment, you can shoes a unique subdomain that represents your application

How to decouple an RDS instance from an Elastic Beanstalk environment without downtime, database sync issues, or data loss

* Create a RDS DB snapshot
* Safeguard your amazon RDS DB instance from deletion
* Create a new Elastic beanstalk environment
* Perform a blue/green deployment to avoid down time
* Remove the security group rule for the old elastic beanstalk environment
* Terminate the old elastic beanstalk environment / Delete the stack

Question 4: Incorrect

An application is hosted in Elastic Beanstalk which is currently running in Java 7 runtime environment. A new version of the application is ready to be deployed and the developer was tasked to upgrade the platform to Java 8 to accommodate the changes.

Which of the following is the MOST appropriate action that the developer should do to upgrade the platform?

* ​Perform a Canary Deployment
* Perform a Blue/Green Deployment.(Correct)
* Manually upgrade the Java runtime environment of the EC2 instances in the Elastic Beanstalk environment.
* Update the environment's platform version to Java 8.(Incorrect)

Elastic Beanstalk regularly releases new platform versions to update all Linux-based and Windows Server-based [platforms](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html). New platform versions provide updates to existing software components and support for new features and configuration options.

You can use the Elastic Beanstalk console or the EB CLI to update your environment's platform version. Depending on the platform version you'd like to update to, Elastic Beanstalk recommends one of two methods for performing platform updates.

[Method 1 – Update your Environment's Platform Version](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.platform.upgrade.html#using-features.platform.upgrade.config) - This is the recommended method when you're **updating to the latest platform version**, without a change in runtime, web server, or application server versions, and without a change in the major platform version. This is the most common and routine platform update.

[Method 2 – Perform a Blue/Green Deployment](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.platform.upgrade.html#using-features.platform.upgrade.bluegreen) - This is the recommended method when you're updating to a **different runtime,** **web server**, or **application server versions**, or to a **different major platform version**. This is a good approach when you want to take advantage of new runtime capabilities or the latest Elastic Beanstalk functionality.

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.

Hence, the correct answer is to perform a Blue/Green deployment to safely upgrade the application's runtime environment from Java 7 to Java 8.

Updating the environment's platform version to Java 8 is incorrect because using this method is only recommended when you're updating to the latest platform version without a change in the runtime environment.

Manually upgrading the Java runtime environment of the EC2 instances in the Elastic Beanstalk environment is incorrect because although this method will work, this entails a lot of configuration to implement compared with just performing a blue/green deployment. In addition, this method may introduce operational risk because the environment may go down while the developer is doing the updates manually.

Performing a Canary deployment is incorrect because this type of deployment is primarily used in Lambda and not in Elastic Beanstalk.

References:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.platform.upgrade.html#using-features.platform.upgrade.bluegreen>

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

Check out this AWS Elastic Beanstalk Cheat Sheet:

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

Tutorials Dojo's AWS Certified Developer Associate Exam Study Guide:

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

Question 6: Incorrect

A write-heavy data analytics application is using DynamoDB database which has a global secondary index. Whenever the application is performing heavy write activities on the table, the DynamoDB requests return a ProvisionedThroughputExceededException.

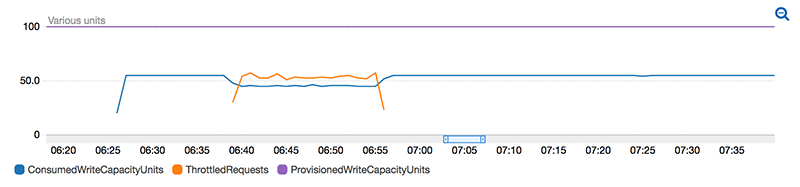
Which of the following is the MOST likely cause of this issue?

* The provisioned write capacity for the global secondary index is greater than the write capacity of the base table.(Incorrect)
* The provisioned throughput exceeds the current throughput limit for your account.
* The provisioned write capacity for the global secondary index is less than the write capacity of the base table.(Correct)
* ​The rate of requests exceeds the allowed throughput.

#### 

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.**

To view the provisioned throughput settings for a global secondary index, use the DescribeTable operation; detailed information about all of the table's global secondary indexes will be returned.

Hence, the most likely cause of this issue is that the provisioned write capacity for the global secondary index is less than the write capacity of the base table.

The option that says: The provisioned write capacity for the global secondary index is greater than the write capacity of the base table is incorrect because it should be the other way around, just as mentioned above. If the provisioned WCU of the global secondary index is greater than its base table then this issue is unlikely to happen.

The option that says: The provisioned throughput exceeds the current throughput limit for your account is incorrect because this will only happen if DynamoDB returns a RequestLimitExceeded exception.

The option that says: The rate of requests exceeds the allowed throughput is incorrect because this will only happen if DynamoDB returns a ThrottlingException.

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/aws-cheat-sheet-amazon-dynamodb/>

Tutorials Dojo's AWS Certified Developer Associate Exam Study Guide:

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

AMAZON DYNAMODB

* NoSQL database service that provides fast and predictable performance with seamless scalability.
* Offers encryption at rest
* You can create database tables that can store and retrieve any amount of data, and serve any level of request traffic.
* You can scale up or scaled down your tables’ throughput capacity without downtime or performance degradation, and use the AWS management console to monitor resource utilization and performance metrics
* Provides on-demand backup capability as well as enable point-in-time recovery for yourDynamoDB tables. With point in time recovery , you can restore that table to any point in time during the last 35 days
* All of your data is stored in partitions, backed by solid state disks and automatically replicated across multiple AZs in an AWS region, providing built-in high availability and data durability
* You can create tables that are automatically replicated across two or more AWS regions, with full support for multi master writes
* AWS now specifies the IP address ranges for Amazon DynamoDB endpoints. You can use these IP address ranges in your routing and firewall policies to control outbound application traffic. You can also use these ranges to control outbound traffic for your applications on your amazon VPC behind AWS VPN or AWS Direct Connect.

CORE COMPONENTS

* Tables
  + A collection of items
    - DynamoDB stores data in a table, which is a collection of data
    - Are schemaless
    - There is a initial limit of 256 tables per region
* Items
  + A colelciton of attributes
    - DynamoDb uses **primary keys** to uniquely identify each item in a table and **secondary indexes** to provide more querying flexibility
    - Each table contains zero or more items
* Attributes
  + A fundamental data element
    - DynamoDB supports nested attributes up to 32 levels deep
* Primary Key
  + Uniquely identifies each item in the table, so that no two items are the same key. Must be scalar
    - Partition key
      * A simple primary key, composed of one attribute
    - Partition key and Sort Key (composite primary key)
      * Composed of two attributes
  + DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition in which the item will be stored. All items with the same partition key are stored together, in sorted orderby sort key value. If no sort key is used, no two items can have the same partition key value
* Secondary Indexes
  + Let you query the table data using an alternate key, in addition to queries against the primary key
  + You can create one or two secondary indexes on a table
  + Two kinds of indexs
    - Global Secondary Index
      * An index with a partition key and sort key that can be different from those on the table
    - Local Secondary Index
      * An index that has the same partition key as the table, but a different sort key.
  + You can define up to 20 global secondary indexes and 5 local secondary indexes per table
* DynamoDB Streams
  + An optional feature that captures data modification events in DynamoDB tables
  + The naming convention for dynamoDB streams endpoints is steams.dynamodb.amazonaws.com
  + Each even is represented by a stream record, and captures the following events:
    - New Item is added to the table:
      * Captures an image of the entire item, including all of its attributes
    - Updated Item
      * Captures the “before” and “after” image of any attributes that were modified in the item.
    - Delete Item
      * Captures an image of the entire item before it was deleted.
  + Each stream record also contains the name of the tabel, the even timestamp, and other metadata
  + Stream records are organized into groups, or shards. Each shard acts as a container for multiple stream records, and contains information required for accessing and iterating through these records.
  + Stream records have a lifetime of 24 hours; after that they are automatically removed from the stream
  + You can use dynamoDb streams together with aWS lambda to create a trigger, which is a code that executes automatically whenever an event of interest appears in a stream
  + DynamoDB streams enable powerful solutions such as data replication within and across regions, materialized views of data in dynamoDb tables, data analysis using kinesis materialized views and much more.\

DATA TYPES FOR ATTRIBUTES

* Scalar Types
  + Can represent exactly one value. IT may be a number, string, binary, boolean, and null --Primary keys should be a scalar type
* Document Types
  + A document type can represent a complex structure with nested attributes. It may be a JSON document. The document types are lists and maps
* Set Types
  + Represent multiple scalar values. The set types are string set, number set, and binary set

OTHER NOTES

* Upon reading from a database the response may not reflect the results of a recently completed write operation. This is because of dynamodb eventually consistent reads
* 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.
* DynamoDB does not support consistent reads across AWS regions
* When you create a table or index in DynamDB, you must specify your throughput capacity requirements for read and write activity in terms of:
  + One read capacity unit represents one strongly consistent read pers eonc, or two eventually consistent reads per second, for an item up to 4KB in size. If you need to read an item what is larger than 4KB DynamoDB will need to consume additional read capacity units
  + One Write capacity represents one write per second for an item up to 1KB in size. If you need to write an item that is larger than 2B , DynamoDb will need to consume additional write capacity units.
* Throttling prevents your application from consuming too many capacity units. DynamoDB can throttle read or write requests that exceed the throughput settings for a table and can also throttle read requests exceeds for an index
* When a request is throttled it fails with an HTTP 400 code ( bad request) and aprovisioneThroughputexceededexception

Throughput Management

* DynamoDB auto scaling
  + Define a range (upper and lower limits) for read and write capacity units, and define a target utilization percentage within that range
  + A table or global secondary index can increase its provisioned read and write capacity to handle sudden increases in traffic, without request throttling
  + DynamoDB auto scaling can decrease the throughput when the workload decreases so that you don’t pay for unused provisioned capacity
* Provisioned Throughput
  + Manually defined maximum amount of capacity that an application can consume from a table or index. I your application exceeds your provisioned throughput settings, it is subject to request throttling
* Reserved capacity
  + With reserved capacity, you pay a one-time upfront fee and commit to a minimum usage level over a period of time, for cost-saving solutions.
* Amazon DynamoDB on-demand is a flexible capacity mode for DynamoDB capable of serving thousands of requests per second without capacity planning. When you choose on-demand capacity mode, DynamoDB instantly accommodates your workload as they ramp up or down to any previously reached traffic level. If a workload’s traffic level hits a new peak, DynamoDB adapts rapidly to accommodate the workload. DynamoDB on-demand offers simple pay-per-request pricing for read and write requests so tahat you only pay for what you use, making it easy to balance costs and performance

Capacity Unit Consumption

* CUC Reads
  + Strongly consistent read requests consume one read capacity unit, while an eventually consistent read request consumes .5 of a read capacity unit
  + GetItem
    - Reads a single item from a table
  + BatchGetItem
    - Reads up to 100 items, from one or more tables
  + Query
    - Reads multiple items that have teh same partition key value
  + Scan
    - Reads all of the items in a table
* CUC Writes
  + PutItem
    - Writes a single item to a table
  + UpdateItem
    - Modifies a single item in the table
  + DeleteItem
    - Removes a single item from a table
  + BatchWriteItem
    - Writes up to 25 items to one or more tables

DynamoDB Auto Scaling

* When yo use the AWS Management console to create a new table dynamodb auto scaling is enabled for that table by default
* Uses the aws application auto scaling service to dynamically adjust provisioned throughput capacity on your behalf, in response to actual traffic patterns
* You create a scaling policy for a table or a global secondary index. The scaling policy specifies whether you want to scale read capacity or write capacity and the min and max provisioned capacity unit settings for the table or index. Th4e scaling policy also contains a target utilization which is the percentage of consumed provision through put at a point in time
* DynamoDB auto scaling doesn't prevent you from manually modifying provisioned throughput settings.
* If you enable DynamoDB auto scaling for a table that has one or more global secondary indexes, aws highly recommends that you also apply auto scaling uniformly to those indexes

DynamoDB Items

* You can use the UpdateItem operation to implement an **atomic counter**
  + A numeric attribute that is incremented, unconditionally, without interfering with other write requests
* DynamoDB oiptionally supports conditional writes for these operations
  + Putitem
  + Updateitem
  + Deleteitem
  + A conditional write will succeed only if the item attributes meet one or more expected conditions
* Conditional writes can be idempotent if the conditional if the conditional check is on the same attributes that is being updated. DynamoDB performs a given write request only if certain attribute values in the item match wha you expect them to be at the time fo the request
* Expressions
  + To get only a few attributes of an item use a **projection expression**
  + An expression attribute name is a placeholder that you use in an expression, as an alternative to an actual attribute name. An expression attribute name must begin with a #, and be followed by one or mor alphanumeric characters
  + Expression attribute values are substitutes for the actual values that you want to compare
    - Values that you might not kow until runtime.
    - An expression attribute value must begin with a :, and be followed by one or more alphanumeric characters
  + For putitem, updateitem and delete item operations you can specify a condition expression to determine which items should be modified. If the condition expression evaluate to true, the operation succeeds; otherwise the operation fails
  + An update expression specifies how updateitem will modify the attributes of an item

Time To Live (TTL)

* Allows one to define when items in a table expire so that they can be automatically deleted from teh database

DynamoDB Queries

* The query operation finds items based on primary key values
* You can query any table or secondary index that has a composite primary key ( a partition key and a sort key)
* A key condition expression is a search criteria that determines the items to bread from the table or index
* You must specify the partition key name and value as equality condition
* You can optionally provide a second condition for the sort key.
* The sort key condition must use one of the comparison operators or BETWEEN, AND
* A single query operation can retrieve a max of 1MB of data
* For further refining of query results you can optionally provide a filter expression to determine which items within the query results should be retired to you . All other results are discarded
* The query operation allows you to limit the number of items that it returns in the the result by setting the limit parameter to the max number of items that you want
* DynamoDB paginates the results from query operations, where query results are divided into pages of data that are 1mb in size (or less)
* ScannedCount is the number of items that matched the key condition expression before a filter expression if present was applied
* Count is the number of items that remain, after a filter expression (if present) was applied

DynamoDB Scans

* AS Scan opertiaon dreads every item in a table or a secondary index. By default a scan operation returns all of the data attributes for evey item in the table or index.
* Scan always returns a result set. If there are no matching items are found the result set will be empty
* A single scan request can retrieve a max of 1mb of data
* You can optionally provide a filter expression
* You can limit thenumber of items that is returned in the result
* Paginates the results from scan operations
* Scannedcount is the number of items evaluated, before any scan filter is applied
* Count is the number of items that remain, after a filter expression was applied
* A scan operation performs eventually consistent reads, by default
* By default the scan operation processes data sequentially.

On-Demand Backup and Restore

* You can use IAM to restrict DynamoDB backup and restore actions for some resources
* All backup and restore actions are captured and recorded in AWS cloudTrail
* Backups
  + Each time you create an on-demand backup the entire table data is backed up
  + All backups and restores in dynamodb work without consuming any provisioned throughpupt on the table
  + Dynamodb backups to not guarantee causal consistency cacross items; however, hte skew between updates in a backup is usually much less than a second
  + Backup and restore works only in the same AWS region as the source table
  + Included in the backup are:
    - Database data
    - Global secondary indexes
    - Local secondary indexes
    - Streams
    - Provisioned read and write capacity
  + While a backup is in progress you can’t do the following
    - Pause or cancel the backup operation
    - Delte the soure table of the backup
    - Disable backups on a table if a backup for that table is in progress
* Restores
  + You cannot overwrite an existing ttable during a restore operation
  + You restore backups to anew table
  + For tables with even data distribution across your primary keys, the restore time is proportional to the largest single partition by item count and not the overall table size
  + If your soure table contains data with significant skew, the time to restore may increase

DynamoDB Transactions

* Amazon DynamoDB transactions simplify the developer experience of making coordinated, all or nothing changes to multiple items both within and across tables
* Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB helping you to maintain data correctness in your applications
* You can group multiple Put, Update, Delete, and ConditionCheck actions.
  + You can then submit the actions as a single TransactWriteItems operation that either succeeds or fails as a unit
  + YOu can group and submit multiple get actions as a single transactgetitems operation
  + Amazon dynamoDB supports up to 25 unique items and 4MB of data per transactional request

Global Times

* Global tables provide a solution for deploying a multi-region, multi-master database, without having to build and maintain your own replication solution
* You specify the AWS regions where you want the table to be available. DynamoDB peforms all the tasks to create identical tables in these regions and propagate ongoing data changes to all of them
* Replica Table (Replica for short)
  + A single DynamoDb table that functions as a part of a global table
  + Each replica stores the same set of data items
  + Any given global table can only have one replica table per region
  + You can add new oir delta replicas from global tables
* To ensure eventual consistency, DynamoDB gloabl tables use a “last writer wins” reconciliation between concurrent updates where dynamodb makes a best effort to determine the last writer
* If a single aws region becomes isolated or degraded your application can redirect to a different region and perform readsand writes against a different replica table. Dynamodb also keeps track of any writes that have been performed but have not yet been propagated to all of the replica tables