Q: Can I get a history of all EC2 API calls made on my account for security analysis and operational troubleshooting purposes?

Yes. To receive a history of all EC2 API calls (including VPC and EBS) made on your account, you simply turn on CloudTrail in the AWS Management Console. For more information, visit the CloudTrail home page.

Q: Why am I charged when my Elastic IP address is not associated with a running instance?

In order to help ensure our customers are efficiently using the Elastic IP addresses, we impose a small hourly charge for each address when it is not associated to a running instance.

Q: Do I need one Elastic IP address for every instance that I have running?

No. You do not need an Elastic IP address for all your instances. By default, every instance comes with a private IP address and an internet routable public IP address.

Q: Does DynamoDB support in-place atomic updates?

Amazon DynamoDB supports fast in-place updates. You can increment or decrement a numeric attribute in a row using a single API call. Similarly, you can atomically add or remove to sets, lists, or maps.

Q: How do I update and query data items with Amazon DynamoDB?

After you have created a table using the AWS Management Console or CreateTable API, you can use the PutItem or BatchWriteItem APIs to insert items. Then you can use the GetItem, BatchGetItem, or, if composite primary keys are enabled and in use in your table, the Query API to retrieve the item(s) you added to the table.

Q: Are expressions supported for key conditions?

Yes, you can specify an expression as part of the Query API call to filter results based on values of primary keys on a table using the KeyConditionExpression parameter.

Q: Are expressions supported for partition and partition-sort keys?

Yes, you can use expressions for both partition and partition-sort keys. Refer to the documentation page for more information on which expressions work on partition and partition-sort keys.

Q: Does Amazon DynamoDB support increment or decrement operations?

Yes, Amazon DynamoDB allows atomic increment and decrement operations on scalar values.

Q: When should I use Amazon DynamoDB vs Amazon S3?

Amazon DynamoDB stores structured data, indexed by primary key, and allows low latency read and write access to items ranging from 1 byte up to 400KB. Amazon S3 stores unstructured blobs and suited for storing large objects up to 5 TB. In order to optimize your costs across AWS services, large objects or infrequently accessed data sets should be stored in Amazon S3, while smaller data elements or file pointers (possibly to Amazon S3 objects) are best saved in Amazon DynamoDB.

Q: Can DynamoDB be used by applications running on any operating system?
Yes. DynamoDB is a fully managed cloud service that you access via API. DynamoDB can be used by applications running on any operating system (e.g. Linux, Windows, iOS, Android, Solaris, AIX, HP-UX, etc.). We recommend using the AWS SDKs to get started with DynamoDB. You can find a list of the AWS SDKs on our Developer Resources page. If you have trouble installing or using one of our SDKs, please let us know by posting to the relevant AWS Forum.

Q: What is the Data Model?

The data model for Amazon DynamoDB is as follows:

Table: A table is a collection of data items - just like a table in a relational database is a collection of rows. Each table can have an infinite number of data items. Amazon DynamoDB is schema-less, in that the data items in a table need not have the same attributes or even the same number of attributes. Each table must have a primary key. The primary key can be a single attribute key or a "composite" attribute key that combines two attributes. The attribute(s) you designate as a primary key must exist for every item as primary keys uniquely identify each item within the table.

Row

Item: An Item is composed of a primary or composite key and a flexible number of attributes. There is no explicit limitation on the number of attributes associated with an individual item, but the aggregate size of an item, including all the attribute names and attribute values, cannot exceed 400KB.

Column

Attribute: Each attribute associated with a data item is composed of an attribute name (e.g. "Color") and a value or set of values (e.g. "Red" or "Red, Yellow, Green"). Individual attributes have no explicit size limit, but the total value of an item (including all attribute names and values) cannot exceed 400KB.

Q: Is there a limit on the size of an item?

The total size of an item, including attribute names and attribute values, cannot exceed 400KB.

Q: Is there a limit on the number of attributes an item can have?

There is no limit to the number of attributes that an item can have. However, the total size of an item, including attribute names and attribute values, cannot exceed 400KB.

Q: What are the APIs?

CreateTable - Creates a table and specifies the primary index used for data access.
UpdateTable - Updates the provisioned throughput values for the given table.
DeleteTable - Deletes a table.
DescribeTable - Returns table size, status, and index information.
ListTables - Returns a list of all tables associated with the current account and endpoint.
PutItem - Creates a new item, or replaces an old item with a new item (including all the attributes). If an item already exists in the specified table with the same primary key, the new item completely replaces the existing item. You can also use conditional operators to replace an item only if its attribute values match certain conditions, or to insert a new item only if that item doesn't already exist.
BatchWriteItem - Inserts, replaces, and deletes multiple items across multiple tables in a single request, but not as a single transaction. Supports batches of up to 25 items to Put or Delete, with a maximum total request size of 16 MB.
UpdateItem - Edits an existing item's attributes. You can also use conditional operators to perform an update only if the item's attribute values match certain conditions.
DeleteItem - Deletes a single item in a table by primary key. You can also use conditional operators to perform a delete an item only if the item's attribute values match certain conditions.
GetItem - The GetItem operation returns a set of Attributes for an item that matches the primary key. The GetItem operation provides an eventually consistent read by default. If eventually consistent reads are not acceptable for your application, use ConsistentRead.
BatchGetItem - The BatchGetItem operation returns the attributes for multiple items from multiple tables using their primary keys. A single response has a size limit of 16 MB and returns a maximum of 100 items. Supports both strong and eventual consistency.
Query - Gets one or more items using the table primary key, or from a secondary index using the index key. You can narrow the scope of the query on a table by using comparison operators or expressions. You can also filter the query results using filters on non-key attributes. Supports both strong and eventual consistency.

A single response has a size limit of 1 MB.

Scan - Gets all items and attributes by performing a full scan across the table or a secondary index. You can limit the return set by specifying filters against one or more attributes.

Q: What is the consistency model of the Scan operation?

The Scan operation supports eventually consistent and consistent reads. By default, the Scan operation is eventually consistent. However, you can modify the consistency model using the optional ConsistentRead parameter in the Scan API call. Setting the ConsistentRead parameter to true will enable you make consistent reads from the Scan operation. For more information, read the documentation for the Scan operation.

Q: How does the Scan operation work?

You can think of the Scan operation as an iterator. Once the aggregate size of items scanned for a given Scan API request exceeds a 1 MB limit, the given request will terminate and fetched results will be returned along with a LastEvaluatedKey (to continue the scan in a subsequent operation).

Q: Are there any limitations for a Scan operation?

A Scan operation on a table or secondary index has a limit of 1MB of data per operation. After the 1MB limit, it stops the operation and returns the matching values up to that point, and a LastEvaluatedKey to apply in a subsequent operation, so that you can pick up where you left off.

Q: How many read capacity units does a Scan operation consume?

The read units required is the number of bytes fetched by the scan operation, rounded to the nearest 4KB, divided by 4KB. Scanning a table with consistent reads consumes twice the read capacity as a scan with eventually consistent reads.

Q: What data types does DynamoDB support?

DynamoDB supports four scalar data types: Number, String, Binary, and Boolean. Additionally, DynamoDB supports collection data types: Number Set, String Set, Binary Set, heterogeneous List and heterogeneous Map. DynamoDB also supports NULL values.

Q: What types of data structures does DynamoDB support?

DynamoDB supports key-value and document data structures.

Q: What is a key-value store?

A key-value store is a database service that provides support for storing, querying and updating collections of objects that are identified using a key and values that contain the actual content being stored.

Q: What is a document store?

A document store provides support for storing, querying and updating items in a document format such as JSON, XML, and HTML.

Q: Is there a limit to how much data I can store in Amazon DynamoDB?

No. There is no limit to the amount of data you can store in an Amazon DynamoDB table. As the size of your data set grows, Amazon DynamoDB will automatically spread your data over sufficient machine resources to meet your storage requirements.

Q: Is there a limit to how much throughput I can get out of a single table?

No, you can increase the maximum capacity limit setting for Auto Scaling or increase the throughput you have manually provisioned for your table using the API or the AWS Management Console. DynamoDB is able to operate at massive scale and there is no theoretical limit on the maximum throughput you can achieve. DynamoDB automatically divides your table across multiple partitions, where each partition is an independent parallel computation unit. DynamoDB can achieve increasingly high throughput rates by adding more partitions.

Q: Does Amazon DynamoDB remain available when Auto Scaling triggers scaling or when I ask it to scale up or down by changing the provisioned throughput?

Yes. Amazon DynamoDB is designed to scale its provisioned throughput up or down while still remaining available, whether managed by Auto Scaling or manually.

Q: Do I need to manage client-side partitioning on top of Amazon DynamoDB?

No. Amazon DynamoDB removes the need to partition across database tables for throughput scalability.

Q: How highly available is Amazon DynamoDB? (3 Facility Replication)

The service runs across Amazon's proven, high-availability data centers. The service replicates data across three facilities in an AWS Region to provide fault tolerance in the event of a server failure or Availability Zone outage.

Q: How does Amazon DynamoDB achieve high uptime and durability?

To achieve high uptime and durability, Amazon DynamoDB synchronously replicates data across three facilities within an AWS Region.

Q: What are global secondary indexes? (This is Similar to Alternate Keys) to increase query flexibility.

Global secondary indexes are indexes that contain a partition or partition-and-sort keys that can be different from the table's primary key.

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

Amazon DynamoDB supports two types of secondary indexes:

Local secondary index – an index that has the same partition key as the table, but a different sort key. A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a table partition that has the same partition key.

Global secondary index – an index with a partition or a partition-and-sort key that can be different from those on the table. A global secondary index is considered "global" because queries on the index can span all items in a table, across all partitions.

Secondary indexes are automatically maintained by Amazon DynamoDB as sparse objects. Items will only appear in an index if they exist in the table on which the index is defined. This makes queries against an index very efficient, because the number of items in the index will often be significantly less than the number of items in the table.

Global secondary indexes support non-unique attributes, which increases query flexibility by enabling queries against any non-key attribute in the table.

Consider a gaming application that stores the information of its players in a DynamoDB table whose primary key consists of UserId (partition) and GameTitle (sort). Items have attributes named TopScore, Timestamp, ZipCode, and others. Upon table creation, DynamoDB provides an implicit index (primary index) on the primary key that can support efficient queries that return a specific user's top scores for all games.

However, if the application requires top scores of users for a particular game, using this primary index would be inefficient, and would require scanning through the entire table. Instead, a global secondary index with GameTitle as the partition key element and TopScore as the sort key element would enable the application to rapidly retrieve top scores for a game.

"A GSI does not need to have a sort key element. For instance, you could have a GSI with a key that only has a partition element GameTitle." In the example below, the GSI has no projected attributes, so it will just return all items (identified by

primary key) that have an attribute matching the GameTitle you are querying on.

Q: When should I use global secondary indexes?

Global secondary indexes are particularly useful for tracking relationships between attributes that have a lot of different values. For example, you could create a DynamoDB table with CustomerID as the primary partition key for the table and ZipCode as the partition key for a global secondary index, since there are a lot of zip codes and since you will probably have a lot of customers. Using the primary key, you could quickly get the record for any customer. Using the global secondary index, you could efficiently query for all customers that live in a given zip code.

To ensure that you get the most out of your global secondary index's capacity, please review our best practices documentation on uniform workloads.

Q: How do I create a global secondary index for a DynamoDB table?

GSIs associated with a table can be specified at any time. For detailed steps on creating a Table and its indexes, see here. You can create a maximum of 5 global secondary indexes per table.

Q: Does the local version of DynamoDB support global secondary indexes?

Yes. The local version of DynamoDB is useful for developing and testing DynamoDB-backed applications. You can download the local version of DynamoDB here.

Q: What are projected attributes?

The data in a secondary index consists of attributes that are projected, or copied, from the table into the index. When you create a secondary index, you define the alternate key for the index, along with any other attributes that you want to be projected in the index. Amazon DynamoDB copies these attributes into the index, along with the primary key attributes from the table. You can then query the index just as you would query a table.

Q: Can a global secondary index key be defined on non-unique attributes?

Yes. Unlike the primary key on a table, a GSI index does not require the indexed attributes to be unique. For instance, a GSI on GameTitle could index all items that track scores of users for every game. In this example, this GSI can be queried to return all users that have played the game "TicTacToe."

Q: How do global secondary indexes differ from local secondary indexes?

Both global and local secondary indexes enhance query flexibility. An LSI is attached to a specific partition key value, whereas a GSI spans all partition key values. Since items having the same partition key value share the same partition in DynamoDB, the "Local" Secondary Index only covers items that are stored together (on the same partition). Thus, the purpose of the LSI is to query items that have the same partition key value but different sort key values. For example, consider a DynamoDB table that tracks Orders for customers, where CustomerId is the partition key.

An LSI on OrderTime allows for efficient queries to retrieve the most recently ordered items for a particular customer.

In contrast, a GSI is not restricted to items with a common partition key value. Instead, a GSI spans all items of the table just like the primary key. For the table above, a GSI on ProductId can be used to efficiently find all orders of a particular product. Note that in this case, no GSI sort key is specified, and even though there might be many orders with the same ProductId, they will be stored as separate items in the GSI.

In order to ensure that data in the table and the index are co-located on the same partition, LSIs limit the total size of all elements (tables and indexes) to 10 GB per partition key value. GSIs do not enforce data co-location, and have no such restriction.

When you write to a table, DynamoDB atomically updates all the LSIs affected. In contrast, updates to any GSIs defined on the table are eventually consistent.

LSIs allow the Query API to retrieve attributes that are not part of the projection list. This is not supported behavior for GSIs.

Q: How do global secondary indexes work?

In many ways, GSI behavior is similar to that of a DynamoDB table. You can query a GSI using its partition key element, with conditional filters on the GSI sort key element. However, unlike a primary key of a DynamoDB table, which must be unique, a GSI key can be the same for multiple items. If multiple items with the same GSI key exist, they are tracked as separate GSI items, and a GSI query will retrieve all of them as individual items. Internally, DynamoDB will ensure that the contents of the GSI are updated appropriately as items are added, removed or updated.

DynamoDB stores a GSI's projected attributes in the GSI data structure, along with the GSI key and the matching items' primary keys. GSI's consume storage for projected items that exist in the source table. This enables queries to be issued against the GSI rather than the table, increasing query flexibility and improving workload distribution. Attributes that are part of an item in a table, but not part of the GSI key, primary key of the table, or projected attributes are thus not returned on querying the GSI index. Applications that need additional data from the table after querying the GSI, can retrieve the primary key from the GSI and then use either the GetItem or BatchGetItem APIs to retrieve the desired attributes from the table. As GSI's are eventually consistent, applications that use this pattern have to accommodate item deletion (from the table) in between the calls to the GSI and GetItem/BatchItem.

DynamoDB automatically handles item additions, updates and deletes in a GSI when corresponding changes are made to the table. When an item (with GSI key attributes) is added to the table, DynamoDB updates the GSI asynchronously to add the new item. Similarly, when an item is deleted from the table, DynamoDB removes the item from the impacted GSI.

Q: Can I create global secondary indexes for partition-based tables and partition-sort schema tables?

Yes, you can create a global secondary index regardless of the type of primary key the DynamoDB table has. The table's primary key can include just a partition key, or it may include both a partition key and a sort key.

Q: What is the consistency model for global secondary indexes?

GSIs support eventual consistency. When items are inserted or updated in a table, the GSIs are not updated synchronously. Under normal operating conditions, a write to a global secondary index will propagate in a fraction of a second. In unlikely failure scenarios, longer delays may occur. Because of this, your application logic should be capable of handling GSI query results that are potentially out-of-date. Note that this is the same behavior exhibited by other DynamoDB APIs that support eventually consistent reads.

Consider a table tracking top scores where each item has attributes UserId, GameTitle and TopScore. The partition key is UserId, and the primary sort key is GameTitle. If the application adds an item denoting a new top score for GameTitle "TicTacToe" and UserId "GAMER123," and then subsequently queries the GSI, it is possible that the new score will not be in the result of the query. However, once the GSI propagation has completed, the new item will start appearing in such queries on the GSI.

Q: Can I provision throughput separately for the table and for each global secondary index?

Yes. GSIs manage throughput independently of the table they are based on. When you enable Auto Scaling for a new or existing table from the console, you can optionally choose to apply the same settings to GSIs. You can also provision

different throughput for tables and global secondary indexes manually.
Depending upon on your application, the request workload on a GSI can vary significantly from that of the table or other GSIs. Some scenarios that show this are given below:

A GSI that contains a small fraction of the table items needs a much lower write throughput compared to the table.
A GSI that is used for infrequent item lookups needs a much lower read throughput, compared to the table.
A GSI used by a read-heavy background task may need high read throughput for a few hours per day.
As your needs evolve, you can change the provisioned throughput of the GSI, independently of the provisioned throughput of the table.

Consider a DynamoDB table with a GSI that projects all attributes, and has the GSI key present in 50% of the items. In this case, the GSI's provisioned write capacity units should be set at 50% of the table's provisioned write capacity units. Using a similar approach, the read throughput of the GSI can be estimated. Please see DynamoDB GSI Documentation for more details.

Q: How does adding a global secondary index impact provisioned throughput and storage for a table?

Similar to a DynamoDB table, a GSI consumes provisioned throughput when reads or writes are performed to it. A write that adds or updates a GSI item will consume write capacity units based on the size of the update. The capacity consumed by the GSI write is in addition to that needed for updating the item in the table.

Note that if you add, delete, or update an item in a DynamoDB table, and if this does not result in a change to a GSI, then the GSI will not consume any write capacity units. This happens when an item without any GSI key attributes is added to the DynamoDB table, or an item is updated without changing any GSI key or projected attributes.

A query to a GSI consumes read capacity units, based on the size of the items examined by the query.

Storage costs for a GSI are based on the total number of bytes stored in that GSI. This includes the GSI key and projected attributes and values, and an overhead of 100 bytes for indexing purposes.

Q: Can DynamoDB throttle my application writes to a table because of a GSI's provisioned throughput?

Because some or all writes to a DynamoDB table result in writes to related GSIs, it is possible that a GSI's provisioned throughput can be exhausted. In such a scenario, subsequent writes to the table will be throttled. This can occur even if the table has available write capacity units.

Q: How often can I change provisioned throughput at the index level?

Tables with GSIs have the same daily limits on the number of throughput change operations as normal tables.

Q: How am I charged for DynamoDB global secondary index?

You are charged for the aggregate provisioned throughput for a table and its GSIs by the hour. When you provision manually, while not required, You are charged for the aggregate provisioned throughput for a table and its GSIs by the hour. In addition, you are charged for the data storage taken up by the GSI as well as standard data transfer (external) fees. If you would like to change your GSI's provisioned throughput capacity, you can do so using the DynamoDB Console or the UpdateTable API or the PutScalingPolicy API for updating Auto Scaling policy settings.
Q: Can I specify which global secondary index should be used for a query?

Yes. In addition to the common query parameters, a GSI Query command explicitly

includes the name of the GSI to operate against. Note that a query can use only one GSI.

Q: What API calls are supported by a global secondary index?

The API calls supported by a GSI are Query and Scan. A Query operation only searches index key attribute values and supports a subset of comparison operators. Because GSIs are updated asynchronously, you cannot use the ConsistentRead parameter with the query. Please see here for details on using GSIs with queries and scans.

Q: What is the order of the results in scan on a global secondary index?

For a global secondary index, with a partition-only key schema there is no ordering. For global secondary index with partition-sort key schema the ordering of the results for the same partition key is based on the sort key attribute.

Q. Can I change Global Secondary Indexes after a table has been created?

Yes, Global Secondary Indexes can be changed at any time, even after the table has been created.

Q. How can I add a Global Secondary Index to an existing table?

You can add a Global Secondary Indexes through the console or through an API call. On the DynamoDB console, first select the table for which you want to add a Global Secondary Index and click the "Create Index" button to add a new index. Follow the steps in the index creation wizard and select "Create" when done. You can also add or delete a Global Secondary Index using the UpdateTable API call with the GlobalSecondaryIndexes parameter.You can learn more by reading our documentation page.

Q. How can I delete a Global Secondary Index?

You can delete a Global Secondary Index from the console or through an API call. On the DynamoDB console, select the table for which you want to delete a Global Secondary Index. Then, select the "Indexes" tab under "Table Items" and click on the "Delete" button next to delete the index. You can also delete a Global Secondary Index using the UpdateTable API call.You can learn more by reading our documentation page.

Q. Can I add or delete more than one index in a single API call on the same table?

You can only add or delete one index per API call.

Q. What happens if I submit multiple requests to add the same index?

Only the first add request is accepted and all subsequent add requests will fail till the first add request is finished.

Q. Can I concurrently add or delete several indexes on the same table?

No, at any time there can be only one active add or delete index operation on a table.

Q. Should I provision additional throughput to add a Global Secondary Index?

With Auto Scaling, it is recommended that you apply the same settings to Global Secondary Index as the table. When you provision manually, while not required, it is highly recommended that you provision additional write throughput that is separate from the throughput for the index. If you do not provision additional write throughput, the write throughput from the index will be consumed for adding the new index. This will affect the write performance of the index while the index is being created as well as increase the time to create the new index.

Q. Do I have to reduce the additional throughput on a Global Secondary Index once the index has been created?

Yes, you would have to dial back the additional write throughput you provisioned for adding an index, once the process is complete.

Q. Can I modify the write throughput that is provisioned for adding a Global Secondary Index?

Yes, you can dial up or dial down the provisioned write throughput for index creation at any time during the creation process.

Q. When a Global Secondary Index is being added or deleted, is the table still available?

Yes, the table is available when the Global Secondary Index is being updated.

Q. When a Global Secondary Index is being added or deleted, are the existing indexes still available?

Yes, the existing indexes are available when the Global Secondary Index is being updated.

Q. When a Global Secondary Index is being created added, is the new index available?

No, the new index becomes available only after the index creation process is finished.

Q. How long does adding a Global Secondary Index take?

The length of time depends on the size of the table and the amount of additional provisioned write throughput for Global Secondary Index creation. The process of adding or deleting an index could vary from a few minutes to a few hours. For example, let's assume that you have a 1GB table that has 500 write capacity units provisioned and you have provisioned 1000 additional write capacity units for the index and new index creation. If the new index includes all the attributes in the table and the table is using all the write capacity units, we expect the index creation will take roughly 30 minutes.

Q. How long does deleting a Global Secondary Index take?

Deleting an index will typically finish in a few minutes. For example, deleting an index with 1GB of data will typically take less than 1 minute.

Q. How do I track the progress of add or delete operation for a Global Secondary Index?

You can use the DynamoDB console or DescribeTable API to check the status of all indexes associated with the table. For an add index operation, while the index is being created, the status of the index will be "CREATING". Once the creation of the index is finished, the index state will change from "CREATING" to "ACTIVE". For a delete index operation, when the request is complete, the deleted index will cease to exist.

Q. Can I get a notification when the index creation process for adding a Global Secondary Index is complete?

You can request a notification to be sent to your email address confirming that the index addition has been completed. When you add an index through the console, you can request a notification on the last step before creating the index. When the index creation is complete, DynamoDB will send an SNS notification to your email.

Q. What happens when I try to add more Global Secondary Indexes, when I already have 5?

You are currently limited to 5 GSIs. The "Add" operation will fail and you will get an error.

Q. Can I reuse a name for a Global Secondary Index after an index with the same name has been deleted?

Yes, once a Global Secondary Index has been deleted, that index name can be used again when a new index is added.

Q. Can I cancel an index add while it is being created?

No, once index creation starts, the index creation process cannot be canceled.

Q: Are GSI key attributes required in all items of a DynamoDB table?

No. GSIs are sparse indexes. Unlike the requirement of having a primary key, an item in a DynamoDB table does not have to contain any of the GSI keys. If a GSI key has both partition and sort elements, and a table item omits either of them, then that item will not be indexed by the corresponding GSI. In such cases, a GSI can be very useful in efficiently locating items that have an uncommon attribute.

Q: Can I retrieve all attributes of a DynamoDB table from a global secondary index?

A query on a GSI can only return attributes that were specified to be included in the GSI at creation time. The attributes included in the GSI are those that are projected by default such as the GSI's key attribute(s) and table's primary key attribute(s), and those that the user specified to be projected. For this reason, a GSI query will not return attributes of items that are part of the table, but not included in the GSI. A GSI that specifies all attributes as projected attributes can be used to retrieve any table attributes. See here for documentation on using GSIs for queries.

Q: How can I list GSIs associated with a table?

The DescribeTable API will return detailed information about global secondary indexes on a table.

Q: What data types can be indexed?

All scalar data types (Number, String, Binary, and Boolean) can be used for the sort key element of the local secondary index key. Set, list, and map types cannot be indexed.

Q: Are composite attribute indexes possible?

No. But you can concatenate attributes into a string and use this as a key.

Q: What data types can be part of the projected attributes for a GSI?

You can specify attributes with any data types (including set types) to be projected into a GSI.

Q: What are some scalability considerations of GSIs?

Performance considerations of the primary key of a DynamoDB table also apply to GSI keys. A GSI assumes a relatively random access pattern across all its keys. To get the most out of secondary index provisioned throughput, you should select a GSI partition key attribute that has a large number of distinct values, and a GSI sort key attribute that is requested fairly uniformly, as randomly as possible.

Q: What new metrics will be available through CloudWatch for global secondary indexes?

Tables with GSI will provide aggregate metrics for the table and GSIs, as well as breakouts of metrics for the table and each GSI.

Reports for individual GSIs will support a subset of the CloudWatch metrics that are supported by a table. These include:

Read Capacity (Provisioned Read Capacity, Consumed Read Capacity)
Write Capacity (Provisioned Write Capacity, Consumed Write Capacity)
Throttled read events
Throttled write events
For more details on metrics supported by DynamoDB tables and indexes see here.

Q: How can I scan a Global Secondary Index?

Q. What is DynamoDB Triggers?
DynamoDB Triggers is a feature which allow you to execute custom actions based on item-level updates on a DynamoDB table. You can specify the custom action in code.

Q. What can I do with DynamoDB Triggers?
There are several application scenarios where DynamoDB Triggers can be useful. Some use cases include sending notifications, updating an aggregate table, and connecting DynamoDB tables to other data sources.

Q. How does DynamoDB Triggers work? (AWS lambda code + Dynamo Streams)
The custom logic for a DynamoDB trigger is stored in an AWS Lambda function as code. To create a trigger for a given table, you can associate an AWS Lambda function to the stream (via DynamoDB Streams) on a DynamoDB table. When the table is updated, the updates are published to DynamoDB Streams. In turn, AWS Lambda reads the updates from the associated stream and executes the code in the function.

Q. What is DynamoDB Triggers?
DynamoDB Triggers is a feature which allows you to execute custom actions based on item-level updates on a DynamoDB table. You can specify the custom action in code.

Q. What can I do with DynamoDB Triggers?
There are several application scenarios where DynamoDB Triggers can be useful. Some use cases include sending notifications, updating an aggregate table, and connecting DynamoDB tables to other data sources.

Q. How does DynamoDB Triggers work?
The custom logic for a DynamoDB trigger is stored in an AWS Lambda function as code. To create a trigger for a given table, you can associate an AWS Lambda function to the stream (via DynamoDB Streams) on a DynamoDB table. When the table is updated, the updates are published to DynamoDB Streams. In turn, AWS Lambda reads the updates from the associated stream and executes the code in the function.

Q: How long are changes to my DynamoDB table available via DynamoDB Streams?
DynamoDB Streams keep records of all changes to a table for 24 hours. After that, they will be erased.

Q: Do I need to manage the capacity of a stream in DynamoDB Streams?
No, capacity for your stream is managed automatically in DynamoDB Streams. If you significantly increase the traffic to your DynamoDB table, DynamoDB will automatically adjust the capacity of the stream to allow it to continue to accept all updates.

Q: If I delete my DynamoDB table, does the stream also get deleted in DynamoDB Streams?
No, not immediately. The stream will persist in DynamoDB Streams for 24 hours to give you a chance to read the last updates that were made to your table. After 24 hours, the stream will be deleted automatically from DynamoDB Streams.

Q: What happens if I turn off DynamoDB Streams for my table?
If you turn off DynamoDB Streams, the stream will persist for 24 hours but will not be updated with any additional changes made to your DynamoDB table.

Q: Will there be duplicates or gaps in DynamoDB Streams?
No, DynamoDB Streams is designed so that every update made to your table will be represented exactly once in the stream.

Q: What is the DynamoDB Storage Backend for Titan?
The DynamoDB Storage Backend for Titan is a plug-in that allows you to use DynamoDB as the underlying storage layer for Titan graph database. It is a client side solution that implements index free adjacency for fast graph traversals on top of DynamoDB.

Q: Does Amazon DynamoDB report CloudWatch metrics?
Yes, Amazon DynamoDB reports several table-level metrics on CloudWatch. You can make operational decisions about your Amazon DynamoDB tables and take specific actions, like setting up alarms, based on these metrics.

Q: How often are metrics reported?
Most CloudWatch metrics for Amazon DynamoDB are reported in 1-minute intervals while the rest of the metrics are reported in 5-minute intervals.

Q: What is a tag?
A tag is a label you assign to an AWS resource. Each tag consists of a key and a value, both of which you can define. AWS uses tags as a mechanism to organize your resource costs on your cost allocation report.

Q: What DynamoDB resources can I tag?
You can tag DynamoDB tables. Local Secondary Indexes and Global Secondary Indexes associated with the tagged tables are automatically tagged with the same tags. Costs for Local Secondary Indexes and Global Secondary Indexes will show up under the tags used for the corresponding DynamoDB table.

Q: Why should I use Tagging for DynamoDB?
You can use Tagging for DynamoDB for "cost allocation". Using tags for cost allocation enables you to label your DynamoDB resources so that you can easily track their costs against projects or other criteria to reflect your own cost structure.

Q: What is DynamoDB Time-to-Live (TTL)?
DynamoDB Time-to-Live (TTL) is a mechanism that lets you set a specific timestamp to delete expired items from your tables. Once the timestamp expires, the corresponding item is marked as expired and is subsequently deleted from the table. By using this functionality, you do not have to track expired data and delete it manually. TTL can help you reduce storage usage and reduce the cost of storing data that is no longer relevant.

Q: Can I set TTL on existing tables?
Yes. If a table is already created and has an attribute that can be used as TTL for its items, then you only need to enable TTL for the table and designate the appropriate attribute for TTL. If the table does not have an attribute that can be used for TTL, you will have to create such an attribute and update the items with values for TTL.

Q: Can I delete an entire table by setting TTL on the whole table?
No. While you need to define an attribute to be used for TTL at the table level, the granularity for deleting data is at the item level. That is, each item in a table that needs to be deleted after expiry will need to have a value defined for the TTL attribute. There is no option to automatically delete the entire table.

Q: Can I create an index based on the TTL values assigned to items in a table?
Yes. TTL behaves like any other item attribute. You can create indexes the same as with other item attributes.

DynamoDB Accelerator (DAX)
Q. What is Amazon DynamoDB Accelerator (DAX)?
Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that enables you to benefit from fast in-memory performance for demanding applications. DAX improves the performance of read-intensive DynamoDB workloads so repeat reads of cached data can be served immediately with extremely low latency, without needing to be re-queried from DynamoDB. DAX will automatically retrieve data from DynamoDB tables upon a cache miss. Writes are designated as write-through (data is written to DynamoDB first and then updated in the DAX cache). Just like DynamoDB, DAX is fault-tolerant and scalable. A DAX cluster has a primary node and zero or more read-replica nodes. Upon a failure for a primary node, DAX will automatically fail over and elect a new primary. For scaling, you may add or remove read replicas.

Q. What are the common use cases for DAX?
Applications that require the fastest possible response times for reads. Some examples include real-time bidding, social gaming, and trading applications. DAX delivers fast, in-memory read performance for these use cases.

Q. Does DAX support pre-warming of the cache?
DAX utilizes lazy-loading to populate the cache. What this means is that on the

first read of an item, DAX will fetch the item from DynamoDB and then populate the cache. While DAX does not support cache pre-warming as a feature, the DAX cache can be pre-warmed for an application by running an external script/application that reads the desired data.

Q. How does DAX handle data eviction? (TTL)
DAX handles cache eviction in three different ways. First, it uses a Time-to-Live (TTL) value that denotes the absolute period of time that an item is available in the cache. Second, when the cache is full, a DAX cluster uses a Least Recently Used (LRU) algorithm to decide which items to evict. Third, with the write-through functionality, DAX evicts older values as new values are written through DAX. This helps keep the DAX item cache consistent with the underlying data store using a single API call.

Q. What happens if a DAX node fails?
If the primary node fails, DAX automatically detects the failure, selects one of the available read replicas, and promotes it to become the new primary. In addition, DAX provisions a new node in the same availability zone of the failed primary; this new node replaces the newly-promoted read replica. If the primary fails due to a temporary availability zone disruption, the new replica will be launched as soon as the AZ has recovered. If a single-node cluster fails, DAX launches a new node in the same availability zone.

Q. How do I monitor the performance of my DAX cluster?
Metrics for CPU utilization, cache hit/miss counts and read/write traffic to your DAX cluster are available via the AWS Management Console or Amazon CloudWatch APIs. You can also add additional, user-defined metrics via Amazon CloudWatch's custom metric functionality. In addition to CloudWatch metrics, DAX also provides information on cache hit, miss, query and cluster performance via the AWS Management Console.

Q. What is DynamoDB Global Tables?
DynamoDB Global Tables is a new multi-master, cross-region replication capability of DynamoDB to support data access locality and regional fault tolerance for database workloads. Applications can now perform reads and writes to DynamoDB in AWS regions around the world, with changes in any region propagated to every region where a table is replicated.

Q. Why should I use Global Tables?
Global Tables enable you to build applications that take advantage of data locality to reduce overall latency. Your applications can read/write data to the region closest to your end users, thereby improving the overall responsiveness of your application. In addition, Global Tables enables your applications to stay highly available even in the unlikely event of isolation or degradation of an entire region.

Q. Which AWS regions support Global Tables?
Global Tables is currently supported in five regions: US East (Ohio), US East (N. Virginia), US West (Oregon), EU (Ireland), and EU (Frankfurt).

Q. How many AWS regions can I replicate to?
You can have one replica table per region, for as many regions in which Global Tables is supported.

Q. What consistency guarantees does Global Table provide?
Global Tables ensures eventual consistency, meaning that any update made to any item in any replica table will be replicated to all other replicas in the same global table. If there are multiple updates to the same item, all replicas in the global table will agree on the latest update, and hence all replicas will converge continually towards a state in which they store identical data.

Q. What latency can I expect when accessing a replica table?
If your application is hosted in a region with a global table, accessing the local replica table through its regional endpoint will exhibit the same single-digit millisecond latencies that you have come to expect from DynamoDB.

Q. What is On-Demand Backup?
On-Demand Backup allows you to create backups of DynamoDB table data and its

settings. You can initiate an On-Demand Backup any time with a single-click from the AWS Management Console or a single API call. DynamoDB encrypts, catalogs, and stores the backups automatically. You can restore the backups to a new DynamoDB table in the same region anytime.

Q. What are VPC Endpoints for DynamoDB (VPCE for DynamoDB)?
Amazon Virtual Private Cloud (VPC) is an AWS service that provides users a virtual private cloud, by provisioning a logically isolated section of Amazon Web Services (AWS) Cloud. VPC Endpoint (VPCE) for DynamoDB is a logical entity within a VPC that creates a private connection between a VPC and DynamoDB without requiring access over the Internet, through a NAT device, or a VPN connection.

Q. Why should I use VPCE for DynamoDB?
In the past, the main way of accessing DynamoDB from within a VPC was to traverse the Internet, which may have required complex configurations such as firewalls and VPNs. VPC Endpoints for DynamoDB improves privacy and security for customers, especially those dealing with sensitive workloads with compliance and audit requirements, by enabling private access to DynamoDB from within a VPC without the need for an Internet Gateway or NAT Gateway.

Performance Optimization

This section discusses Amazon S3 best practices for optimizing performance in the following topics.

Topics

- Request Rate and Performance Considerations (p. 562)
- TCP Window Scaling (p. 565)
- TCP Selective Acknowledgement (p. 565)

> **Note**
> For more information about high performance tuning, see Enabling High Performance Data Transfers at the Pittsburgh Supercomputing Center (PSC) website.

Request Rate and Performance Considerations

This topic discusses Amazon S3 best practices for optimizing performance depending on your request rates. If your workload in an Amazon S3 bucket routinely exceeds 100 PUT/LIST/DELETE requests per second or more than 300 GET requests per second, follow the guidelines in this topic to ensure the best performance and scalability.

Amazon S3 scales to support very high request rates. If your request rate grows steadily, Amazon S3 automatically partitions your buckets as needed to support higher request rates. However, if you expect a rapid increase in the request rate for a bucket to more than 300 PUT/LIST/DELETE requests per second or more than 800 GET requests per second, we recommend that you open a support case to prepare for the workload and avoid any temporary limits on your request rate. To open a support case, go to Contact Us.

> **Note**
> The Amazon S3 best practice guidelines in this topic apply only if you are routinely processing 100 or more requests per second. If your typical workload involves only occasional bursts of 100 requests per second and less than 800 requests per second, you don't need to follow these guidelines.
> If your workload in Amazon S3 uses Server-Side Encryption with AWS Key Management Service (SSE-KMS), go to Limits in the *AWS Key Management Service Developer Guide* to get more information on the request rates supported for your use case.

The Amazon S3 best practice guidance given in this topic is based on two types of workloads:

- **Workloads that include a mix of request types –** If your requests are typically a mix of GET, PUT, DELETE, or GET Bucket (list objects), choosing appropriate key names for your objects will ensure better performance by providing low-latency access to the Amazon S3 index. It will also ensure scalability regardless of the number of requests you send per second.
- **Workloads that are GET-intensive –** If the bulk of your workload consists of GET requests, we recommend using the Amazon CloudFront content delivery service.

Topics

- Workloads with a Mix of Request Types (p. 563)
- GET-Intensive Workloads (p. 565)

Workloads with a Mix of Request Types

i.e Sequence related Performance Problems.

When uploading a large number of objects, customers sometimes use sequential numbers or date and time values as part of their key names. For example, you might choose key names that use some combination of the date and time, as shown in the following example, where the prefix includes a timestamp:

```
examplebucket/2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo3.jpg
...
examplebucket/2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo6.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

The sequence pattern in the key names introduces a performance problem. To understand the issue, let's look at how Amazon S3 stores key names.

Amazon S3 maintains an index of object key names in each AWS region. Object keys are stored in UTF-8 binary ordering across multiple partitions in the index. The key name dictates which partition the key is stored in. Using a sequential prefix, such as timestamp or an alphabetical sequence, increases the likelihood that Amazon S3 will target a specific partition for a large number of your keys, overwhelming the I/O capacity of the partition. If you introduce some randomness in your key name prefixes, the key names, and therefore the I/O load, will be distributed across more than one partition.

If you anticipate that your workload will consistently exceed 100 requests per second, you should avoid sequential key names. If you must use sequential numbers or date and time patterns in key names, add a random prefix to the key name. The randomness of the prefix more evenly distributes key names across multiple index partitions. Examples of introducing randomness are provided later in this topic.

> **Note**
> The guidelines provided for the key name prefixes in the following section also apply to the bucket name. When Amazon S3 stores a key name in the index, it stores the bucket names as part of the key name (for example, `examplebucket/object.jpg`).

Example 1: Add a Hex Hash Prefix to Key Name

One way to introduce randomness to key names is to add a hash string as prefix to the key name. For example, you can compute an MD5 hash of the character sequence that you plan to assign as the key name. From the hash, pick a specific number of characters, and add them as the prefix to the key name. The following example shows key names with a four-character hash.

> **Note**
> A hashed prefix of three or four characters should be sufficient. We strongly recommend using a hexadecimal hash as the prefix.

```
examplebucket/232a-2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/7b54-2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/921c-2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/ba65-2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/8761-2013-26-05-15-00-00/cust1248473/photo3.jpg
examplebucket/2e4f-2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/9810-2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/7e34-2013-26-05-15-00-01/cust1248473/photo6.jpg
```

```
examplebucket/c34a-2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

Note that this randomness does introduce some interesting challenges. Amazon S3 provides a GET Bucket (List Objects) operation, which returns a UTF-8 binary ordered list of key names. Here are some side-effects:

- Because of the hashed prefixes, however, the listing will appear randomly ordered.
- The problem gets compounded if you want to list object keys with specific date in the key name. The preceding example uses 4 character hex hash, so there are 65536 possible character combinations (4 character prefix, and each character can be any of the hex characters 0-f). So you will be sending 65536 List Bucket requests each with a specific prefix that is a combination of 4-digit hash and the date. For example, suppose you want to find all keys with 2013-26-05 in the key name. Then you will send List Bucket requests with prefixes such `[0-f][0-f][0-f][0-f]2013-26-05`.

You can optionally add more prefixes in your key name, before the hash string, to group objects. The following example adds `animations/` and `videos/` prefixes to the key names.

```
examplebucket/animations/232a-2013-26-05-15-00-00/cust1234234/animation1.obj
examplebucket/animations/7b54-2013-26-05-15-00-00/cust3857422/animation2.obj
examplebucket/animations/921c-2013-26-05-15-00-00/cust1248473/animation3.obj
examplebucket/videos/ba65-2013-26-05-15-00-00/cust8474937/video2.mpg
examplebucket/videos/8761-2013-26-05-15-00-00/cust1248473/video3.mpg
examplebucket/videos/2e4f-2013-26-05-15-00-01/cust1248473/video4.mpg
examplebucket/videos/9810-2013-26-05-15-00-01/cust1248473/video5.mpg
examplebucket/videos/7e34-2013-26-05-15-00-01/cust1248473/video6.mpg
examplebucket/videos/c34a-2013-26-05-15-00-01/cust1248473/video7.mpg
...
```

In this case, the ordered list returned by the GET Bucket (List Objects) operation will be grouped by the prefixes `animations` and `videos`.

> **Note**
> Again, the prefixes you add to group objects should not have sequences, or you will again overwhelm a single index partition.

Example 2: Reverse the Key Name String

Suppose your application uploads objects with key names whose prefixes include an increasing sequence of application IDs.

```
examplebucket/2134857/data/start.png
examplebucket/2134857/data/resource.rsrc
examplebucket/2134857/data/results.txt
examplebucket/2134858/data/start.png
examplebucket/2134858/data/resource.rsrc
examplebucket/2134858/data/results.txt
examplebucket/2134859/data/start.png
examplebucket/2134859/data/resource.rsrc
examplebucket/2134859/data/results.txt
```

In this key naming scheme, write operations will overwhelm a single index partition. If you reverse the application ID strings, however, you have the key names with random prefixes:

```
examplebucket/7584312/data/start.png
examplebucket/7584312/data/resource.rsrc
examplebucket/7584312/data/results.txt
examplebucket/8584312/data/start.png
```

```
examplebucket/8584312/data/resource.rsrc
examplebucket/8584312/data/results.txt
examplebucket/9584312/data/start.png
examplebucket/9584312/data/resource.rsrc
examplebucket/9584312/data/results.txt
```

Reversing the key name string lays the groundwork for Amazon S3 to start with the following partitions, one for each distinct first character in the key name. The `examplebucket` refers to the name of the bucket where you upload application data.

```
examplebucket/7
examplebucket/8
examplebucket/9
```

This example illustrate how Amazon S3 can use the first character of the key name for partitioning, but for very large workloads (more than 2000 requests per seconds or for bucket that contain billions of objects), Amazon S3 can use more characters for the partitioning scheme. Amazon S3 can automatically split these partitions further as the key count and request rate increase over time.

GET-Intensive Workloads

If your workload is mainly sending GET requests, in addition to the preceding guidelines, you should consider using Amazon CloudFront for performance optimization.

Integrating Amazon CloudFront with Amazon S3, you can distribute content to your users with low latency and a high data transfer rate. You will also send fewer direct requests to Amazon S3, which will reduce your costs.

For example, suppose you have a few objects that are very popular. Amazon CloudFront will fetch those objects from Amazon S3 and cache them. Amazon CloudFront can then serve future requests for the objects from its cache, reducing the number of GET requests it sends to Amazon S3. For more information, go to the Amazon CloudFront product detail page.

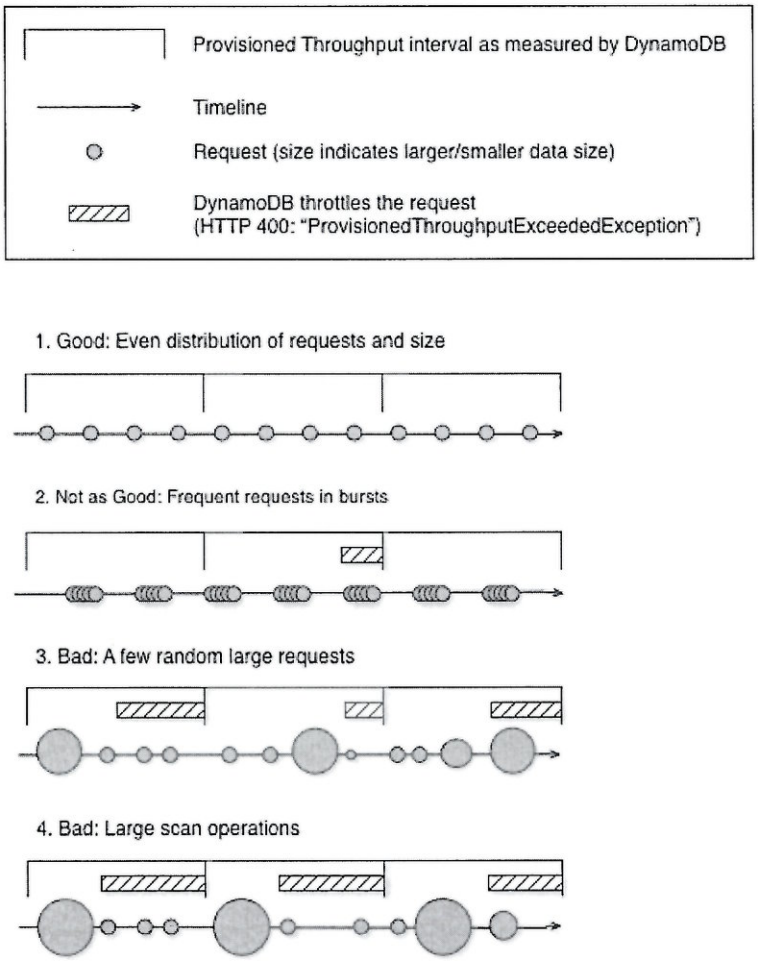
TCP Window Scaling

TCP window scaling allows you to improve network throughput performance between your operating system and application layer and Amazon S3 by supporting window sizes larger than 64 KB. At the start of the TCP session, a client advertises its supported receive window WSCALE factor, and Amazon S3 responds with its supported receive window WSCALE factor for the upstream direction.

Although TCP window scaling can improve performance, it can be challenging to set correctly. Make sure to adjust settings at both the application and kernel level. For more information about TCP window scaling, refer to your operating system's documentation and go to RFC 1323.

TCP Selective Acknowledgement

TCP selective acknowledgement is designed to improve recovery time after a large number of packet losses. TCP selective acknowledgement is supported by most newer operating systems, but might have to be enabled. For more information about TCP selective acknowledgements, refer to the documentation that accompanied your operating system and go to RFC 2018.



Instead of using a large `Scan` operation, you can use the following techniques to minimize the impact of a scan on a table's provisioned throughput.

- **Reduce Page Size** *for Fewer read operation to Create Pause b/w requests.

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

- **Isolate Scan Operations** Mission Critical + Shadow Table.

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

You should configure your application to retry any request that receives a response code that indicates you have exceeded your provisioned throughput, or increase the provisioned throughput for your table using the `UpdateTable` operation. If you have temporary spikes in your workload that cause your throughput to exceed, occasionally, beyond the provisioned level, retry the request with exponential backoff. For more information about implementing exponential backoff, see Error Retries and Exponential Backoff (p. 193).

Take Advantage of Parallel Scans

Many applications can benefit from using parallel `Scan` operations rather than sequential scans. For example, an application that processes a large table of historical data can perform a parallel scan much faster than a sequential one. Multiple worker threads in a background "sweeper" process could scan a table at a low priority without affecting production traffic. In each of these examples, a parallel `Scan` is used in such a way that it does not starve other applications of provisioned throughput resources.

Although parallel scans can be beneficial, they can place a heavy demand on provisioned throughput. With a parallel scan, your application will have multiple workers that are all running `Scan` operations concurrently, which can very quickly consume all of your table's provisioned read capacity. In that case, other applications that need to access the table might be throttled.

A parallel scan can be the right choice if the following conditions are met:

- The table size is 20 GB or larger.
- The table's provisioned read throughput is not being fully utilized.
- Sequential `Scan` operations are too slow.

Choosing *TotalSegments*

The best setting for `TotalSegments` depends on your specific data, the table's provisioned throughput settings, and your performance requirements. You will probably need to experiment to get it right. We recommend that you begin with a simple ratio, such as one segment per 2 GB of data. For example, for a 30 GB table, you could set `TotalSegments` to 15 (30 GB / 2 GB). Your application would then use fifteen workers, with each worker scanning a different segment.

You can also choose a value for `TotalSegments` that is based on client resources. You can set `TotalSegments` to any number from 1 to 1000000, and DynamoDB will allow you to scan that number of segments. If, for example, your client limits the number of threads that can run concurrently, you can gradually increase `TotalSegments` until you get the best `Scan` performance with your application.

You will need to monitor your parallel scans to optimize your provisioned throughput utilization, while also making sure that your other applications aren't starved of resources. Increase the value for `TotalSegments` if you do not consume all of your provisioned throughput but still experience throttling in your `Scan` requests. Reduce the value for `TotalSegments` if the `Scan` requests consume more provisioned throughput than you want to use.

Best Practices for Local Secondary Indexes

Topics

- Use Indexes Sparingly (p. 704)
- Choose Projections Carefully (p. 704)
- Optimize Frequent Queries To Avoid Fetches (p. 704)

```
aws elb set-load-balancer-policies-for-backend-server --load-balancer-name my-
loadbalancer --instance-port 80 --policy-names "[]"
```

2. (Optional) Use the following describe-load-balancers command to verify that the policy is disabled:

```
aws elb describe-load-balancers --load-balancer-name my-loadbalancer
```

The response includes the following information, which shows that no ports are associated with a policy.

```
{
    "LoadBalancerDescriptions": [
        {
            ...
            "BackendServerDescriptions": [],
            ...
        }
    ]
}
```

Configure Sticky Sessions for Your Classic Load Balancer (Session Affinity – Cookie Named AWSELB)

By default, a Classic Load Balancer routes each request independently to the registered instance with the smallest load. However, you can use the *sticky session* feature (also known as *session affinity*), which enables the load balancer to bind a user's session to a specific instance. This ensures that all requests from the user during the session are sent to the same instance.

The key to managing sticky sessions is to determine how long your load balancer should consistently route the user's request to the same instance. If your application has its own session cookie, then you can configure Elastic Load Balancing so that the session cookie follows the duration specified by the application's session cookie. If your application does not have its own session cookie, then you can configure Elastic Load Balancing to create a session cookie by specifying your own stickiness duration.

Elastic Load Balancing creates a cookie, named AWSELB, that is used to map the session to the instance.

Requirements

- An HTTP/HTTPS load balancer.
- At least one healthy instance in each Availability Zone.

Compatibility

- The RFC for the path property of a cookie allows underscores. However, Elastic Load Balancing URI encodes underscore characters as `%5F` because some browsers, such as Internet Explorer 7, expect underscores to be URI encoded as `%5F`. Because of the potential to impact browsers that are currently working, Elastic Load Balancing continues to URI encode underscore characters. For example, if the cookie has the property `path=/my_path`, Elastic Load Balancing changes this property in the forwarded request to `path=/my%5Fpath`.
- You can't set the `secure` flag or `HttpOnly` flag on your duration-based session stickiness cookies. However, these cookies contain no sensitive data. Note that if you set the `secure` flag or `HttpOnly` flag on an application-controlled session stickiness cookie, it is also set on the AWSELB cookie.

- If you have a trailing semicolon in the `Set-Cookie` field of an application cookie, the load balancer ignores the cookie.

Contents

- Duration-Based Session Stickiness (p. 78)
- Application-Controlled Session Stickiness (p. 79)

Duration-Based Session Stickiness

Stickiness Policy — Defines Cookie Expiration

The load balancer uses a special cookie to track the instance for each request to each listener. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the instance specified in the cookie. If there is no cookie, the load balancer chooses an instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that instance. The stickiness policy configuration defines a cookie expiration, which establishes the duration of validity for each cookie. After a cookie expires, the session is no longer sticky.

If an instance fails or becomes unhealthy, the load balancer stops routing requests to that instance, and chooses a new healthy instance based on the existing load balancing algorithm. The request is routed to the new instance as if there is no cookie and the session is no longer sticky.

If a client switches to a listener with a different backend port, stickiness is lost.

To enable duration-based sticky sessions for a load balancer using the console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
3. Select your load balancer.
4. On the **Description** tab, choose **Edit stickiness**.
5. On the **Edit stickiness** page, select **Enable load balancer generated cookie stickiness**.
6. (Optional) For **Expiration Period**, type the cookie expiration period, in seconds. If you do not specify an expiration period, the sticky session lasts for the duration of the browser session.
7. Choose **Save**.

To enable duration-based sticky sessions for a load balancer using the AWS CLI

1. Use the following create-lb-cookie-stickiness-policy command to create a load balancer-generated cookie stickiness policy with a cookie expiration period of 60 seconds:

```
aws elb create-lb-cookie-stickiness-policy --load-balancer-name my-loadbalancer --
policy-name my-duration-cookie-policy --cookie-expiration-period 60
```

2. Use the following set-load-balancer-policies-of-listener command to enable session stickiness for the specified load balancer:

```
aws elb set-load-balancer-policies-of-listener --load-balancer-name my-loadbalancer --
load-balancer-port 443 --policy-names my-duration-cookie-policy
```

> **Note**
> The `set-load-balancer-policies-of-listener` command replaces the current set of policies associated with the specified load balancer port. Every time you use this command, specify the `--policy-names` option to list all policies to enable.

3. (Optional) Use the following describe-load-balancers command to verify that the policy is enabled:

```
aws elb describe-load-balancers --load-balancer-name my-loadbalancer
```

The response includes the following information, which shows that the policy is enabled for the listener on the specified port:

```
{
    "LoadBalancerDescriptions": [
        {
            ...
            "ListenerDescriptions": [
                {
                    "Listener": {
                        "InstancePort": 443,
                        "SSLCertificateId": "arn:aws:iam::123456789012:server-
certificate/my-server-certificate",
                        "LoadBalancerPort": 443,
                        "Protocol": "HTTPS",
                        "InstanceProtocol": "HTTPS"
                    },
                    "PolicyNames": [
                        "my-duration-cookie-policy",
                        "ELBSecurityPolicy-2016-08"
                    ]
                },
                ...
            ],
            ...
            "Policies": {
                "LBCookieStickinessPolicies": [
                 {
                        "PolicyName": "my-duration-cookie-policy",
                        "CookieExpirationPeriod": 60
                    }

                ],
                "AppCookieStickinessPolicies": [],
                "OtherPolicies": [
                    "ELBSecurityPolicy-2016-08"
                ]
            },
            ...
        }
    ]
}
```

Session Stickiness

Application-Controlled Session Stickiness (controlled by the applicati.

The load balancer uses a special cookie to associate the session with the instance that handled the initial request, but follows the lifetime of the application cookie specified in the policy configuration. The load balancer only inserts a new stickiness cookie if the application response includes a new application cookie. The load balancer stickiness cookie does not update with each request. If the application cookie is explicitly removed or expires, the session stops being sticky until a new application cookie is issued.

If an instance fails or becomes unhealthy, the load balancer stops routing requests to that instance, and chooses a new healthy instance based on the existing load balancing algorithm. The load balancer treats the session as now "stuck" to the new healthy instance, and continues routing requests to that instance even if the failed instance comes back.

To enable application-controlled session stickiness using the console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
3. Select your load balancer.
4. On the **Description** tab, choose **Edit stickiness**.
5. On the **Edit stickiness** page, select **Enable application generated cookie stickiness**.
6. For **Cookie Name**, type the name of your application cookie.
7. Choose **Save**.

To enable application-controlled session stickiness using the AWS CLI

1. Use the following create-app-cookie-stickiness-policy command to create an application-generated cookie stickiness policy:

```
aws elb create-app-cookie-stickiness-policy --load-balancer-name my-loadbalancer --
policy-name my-app-cookie-policy --cookie-name my-app-cookie
```

2. Use the following set-load-balancer-policies-of-listener command to enable session stickiness for a load balancer:

```
aws elb set-load-balancer-policies-of-listener --load-balancer-name my-loadbalancer --
load-balancer-port 443 --policy-names my-app-cookie-policy
```

> **Note**
> The `set-load-balancer-policies-of-listener` command replaces the current set of policies associated with the specified load balancer port. Every time you use this command, specify the `--policy-names` option to list all policies to enable.

3. (Optional) Use the following describe-load-balancers command to verify that the sticky policy is enabled:

```
aws elb describe-load-balancers --load-balancer-name my-loadbalancer
```

4. The response includes the following information, which shows that the policy is enabled for the listener on the specified port:

```
{
    "LoadBalancerDescriptions": [
        {
            ...
            "ListenerDescriptions": [
                {
                    "Listener": {
                        "InstancePort": 443,
                        "SSLCertificateId": "arn:aws:iam::123456789012:server-
certificate/my-server-certificate",
                        "LoadBalancerPort": 443,
                        "Protocol": "HTTPS",
                        "InstanceProtocol": "HTTPS"
                    },
                    "PolicyNames": [
                        "my-app-cookie-policy",
                        "ELBSecurityPolicy-2016-08"
                    ]
                },
                {
```

http://169.254.169.254/latest/meta-data/

SDK -

IOS, Android, Browser (Java scripts)

Java, .NET,

Node.js, PHP, Python, Ruby

Go, C++

SQS - message oriented API

SQS - Message can contain upto 256KB of text, billed at 64KB chunks,

Single request can have 1 to 10 messages unto maximum of 256KB payload

Even though there is one message of 256Kb its basically 4 request for billing since (4 * 64KB)

NO ORDER - SQS messages can be delivered multiple times in any order

Design - you can have 2 priority queues for priority based message one for higher and other for lower priority

EC2 instances always poll for messages from the queue (pull from the queue and not push)

Visibility timeout always start from when the application instance polled the message.

Great design - Visibility timeout expires that means there is a failure somewhere since that message was polled but not processed and hence not deleted so other some other process will poll the message again and visibility timeout starts again.

Visibility timeout by default is 30 Seconds up to 12 hour maximum (ChangeMessageVisibility) / maximum visibility

Maximum long polling timeout 20 seconds (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-long-polling.html) –ReceiveMessageWaitTimeSeconds

Messages in the Queue can be retained for up to 14 days

First 1 million request ares free, then $0.50 PER EVERY MILLION REQUESTS

SNS

works on a publish - subscribe model, SNS notifies the message, and hence push based approach. Inexpensive pay as you go

CloudWatch or Autoscaling triggers SNS

SNS can notify to Email, Text / SMS, SQS or any HTTP end point.

protocols: HTTP, HTTPS, EMAIL, EMAIL-JSON, SQS or Application - messages can be customized for each protocol

SNS messages are stored redundantly to multiple AZs

SNS Dataformat - JSON (Subject, Message, TopicArn, MessageId, unsubscribeURL etc..)

$0.50 per 1 million SNS request

Different price for different recipient types

to HTTP: $0.06 / 100,000 notifications deliveries

to EMAIL: $2 / 100,000 notifications deliveries

to SMS: $0.75 / 100 notifications deliveries

http://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-devicetoken.html (CreatePlatformEndpoint API)

SWF - task oriented API

Simple Work flow - human interaction to complete order or collection of services to complete a work order.

Workers - interact with SWF to get task, process received task and return the result

Deciders - program that co-ordinates the tasks, i.e. - ordering, concurrency and scheduling

Workers and Deciders can run independently

TASK is only assigned ONCE and NEVER DUPLICATED (key difference from SQS where messages can be processed multiple times)

SWF Domain - think of it as a container for the work flow. you can register a domain by Console or API

Maximum workflow processing time can 1 year (equivalent seconds) - SQS is 12 hours processing time

CloudFormation

Use of CFT, Beanstalk and Autoscaling are free but you pay for the AWS resources that these services create.

Fn::GetAtt - values that you can use to return result for an AWS created resource or used to display in output

By Default - rollback everything on error

Infrastructure as a code, Version controlled, declarative and flexible

ElasticBeanstalk

Its uses ASG,ELB,EC2,RDS,SNS and S3 to provision things.

Environment Tier - Webserver, Worker

Predefined Configurations - IIS, Node.JS, PHP, Python, Ruby, Tomcat, Go, .NET,

preconfigured docker: Glassfish, Python or generic docker

Environment URL - has to be unique

Dashboard - Recent events, Monitor, Logs, Alarms, Upload and Deploy and Configurations

Configuration - Scaling, Instances (DIRTMCG instance types, key pair), Notifications, Software configuration (e.g. PHP.ini), Networking tier (ELB, VPC config), Data tier(RDS)

Environment properties (Access key and secret key as parameters)

DynamoDB

fast - flexible No sql database - single digit ms latency, fully managed, supports document and key-value (web, gaming, ad-tech, IOT)..

Table, Item (row), attribute (key - value)

Eventual Consistent Reads vs Strongly Consistent Reads

Read Capacity Units, Write Capacity Units (can be scaled up) - push button scalability

Writes are written to 3 different location / facilities/ datacenter (synchronous) - Amazon DynamoDB synchronously replicates data across three facilities in an AWS Region, giving you high availability and data durability.

Two types of primary key -

(1) Single Attribute (think unique id) - Partition Key (Hash Key) composted of 1 attribute (no nesting allowed here) - Partition key will help determine the physical location of data.

(2) Composite key (think unique id and range) - Partition Key(Hash Key) & Sort Key (Range key - e.g date) - composed of 2 attributes - if two data have same partition key (same location) it must have a different sort key, and they will be stored together on single location.

Secondary Indexes

(1) Local Secondary Index - Same Partion Key + Different Sort Key (can only be created while creating the table, cannot be added/removed or modified later)

(2) Global Secondary Index - Different Partition Key + Different Sort Key (can be created during the table creation or can be added later or removed / modified later)

DynamoDB Streams

use to capture any kinda modification to the dynamo db table, Lambda can capture events and push notifications thru SES

Table can be exported to csv (either select all items)

Query vs Scan

Query operation finds item in a table using only primary key attribute values , must provide partition attribute name and the value to search for, you can optionally provide a sort key attribute name and value to refine search results (e.g. all the forums with this ID since last 7 days). By default Query returns all the data attributes for those items with specified primary keys. You can further use ProjectionExpression parameter to only return a selected attributes.

Query results are always sorted by the sort key (ascending for both numbers and string by default). To reverse the sort order set the ScanIndexForward parameter to false

By Default Queries are going to be Eventually consistent but can be changed to StronglyConsistent.

Scan operation is basically examines every item - e.g. dumping the entire table, by default Scan returns all the data attributes but we could use ProjectionExpression parameter to only return a selected attributes.

Query operation is more efficient than scan operation

For quick response time design your table in a way that you can use Query Get or BatchGetItem API (read multiple items - can get upto 100 items or up to 1MB of data) ,

Alternatively design your application to use scan operation in a way that minir impact of your table's request rate since it can use up the entire table's provisioned throughput in a single scan operation

DynamoDB Provisioned Throughput calculations

Items == rows

Read Provisioned Throughput

All units are rounded up to 4KB increments

Eventual Consistent reads (default) consist of 2 reads per second

Strongly Consistent reads consist of 1 read per second

(Size of Read Rounded to nearest 4KB Chunk / 4 KB * no of items) / 2 <– if eventual consistency

(Size of Read Rounded to nearest 4KB Chunk / 4 KB * no of items) / 1 <– if strongly consistency

Write Provisioned Throughput

All units are rounded up to 1KB increments

All writes consist of 1 write per second

(Size of write in KB * no of items) / 1

When you exceed your maximum allowed provisioned throughput for a table or one or more global secondary index you will get 400 HTTP Status code - ProvisionedThroughputExceededException

AssumeRolewithWebIdentity role

Idempotent conditional write

Atomic counters - always need to increment so its not idempotent

if data is critical and no margin of error then must use Idempotent conditional write.

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Limits.html#limits-tables

Only Tables(256 table per region) and ProvisionedThroughput(80 K read, 80K write per account for US east, 20K for other regions) limits can be increased

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/QueryAndScanGuidelines.html (Reduce Page Size for Scan operation and Isolate Scan Operation)

S3

secure, durable, highly scalable object store (1 byte to 5TB), universal namespace (must be unique bucket - regardless of regions),object based key value store, VersionID, Metadata, ACL

The total volume of data and number of objects you can store are unlimited. Individual Amazon S3 objects can range in size from 1 byte to 5 terabytes. The largest object that can be uploaded in a single PUT is 5 gigabytes. For objects larger than 100 megabytes, customers should consider using the Multipart Upload capability. it mean the largest single file into S3 is 5G, but after the 5G files are in S3, they can be assembled into a 5T file,

You can use a Multipart Upload for objects from 5 MB to 5 TB in size (Exam question, scenario where more than 5GB file needs to be uploaded)

object based storage vs block based Storage (EFS)

data is spread out in multiple facilities, you can loose two facilities and still have access to files

For PUTS of New Objects (Read after Write Consistency), For Overwrite PUTS and DELETE (Eventual Consistency)

http://docs.aws.amazon.com/general/latest/gr/awsservicelimits.html#limits_s3 (Number of S3 bucket limit per account – 100)

Storage Tiers/ Class

S3 Standard - Durability (11 9s), Availability (99.99 %) - reliable regular for just about everything

S3 IA (Infrequent Access) - Durability (11 9s), Availability (99.9 %) - accessed every 1 month to 6 months or so (infrequent) but rapid access and low retrieval time (few ms)

S3 RRS(Reduced Redundant Storage)- Durability (99.99%), Availability (99.99 %) - less durability (data that can easily be regenerated - e.g thumbnails) - cheapest of all s3, less fault tolerant then the other two since you are willing to loose the data, reproducible data

Glacier - for archival only (3 to 5 hours restore time)

S3 price - charged for Storage, number of requests, data transfer (tiered so more you use less charge)

bucket name has to be all lowercase letters

S3 for static website hosting (Static Website Hosting > Enable website hosting) - no dynamic

Any time you create a bucket nothing is publicly accessible / Any time you add an object to a bucket its private by default (you will get 403) > Make the files public (even for public hosting)

every object inside the bucket can have different storage class (S3 standard, S3-IA, S3-RRS) and you can turn on server side encryption (AES - 256)

regular bucket link: https://s3-eu-west-1.amazonaws.com/ankittest <– https

bucket with Static website hosting: http://ankittestsite.s3-website-eu-west-1.amazonaws.com <– http (has to be for static hosting), you can turn it into SSL / https with cloudfront though

CORS (CROSS ORIGIN RESOURCE SHARING) - to avoid the use of proxy

Versioning - once enable you cannot disable versioning / although it can be suspend it , if you want to turn it off delete the bucket and recreate (version id)

Once you delete the delete marker, you can get the file back that you have deleted while versioning on

every version is stored separately in the bucket for each version / might not be a good choice for cost perspective for large media files., multiple updates use case also not ideal for versioning.

Versioning's MFA Delete Capability can be used to provide additional layer of security.

Cross Region Replication - (requires versioning enabled on source and destination buckets)

you can enable - need source and destination bucket (create a new bucket, source bucket will not show up on drop down of destination)

Existing objects will not be replicated, only new objects will be replicated across the region

Lifecycle management in S3

(1) when versioning is disabled

Transition to IA S3 - min 30 days and has a 128KB minimum of object size

Archive to Glacier - min 1 day if IA is not checked, min 60 day if Transition to IA S3 is checked

Permanently Delete - min 2 day if IA is not checked and 1 is selected for Glacier, min 61 day if IA is selected 30, Glacier is selected 60.

(2) when versioning is enabled you have lifecycle management options to take action on previous version as well as current version.

Security and Encryption in S3

by default newly created buckets are private

Access control using Bucket Policies (entire bucket) and ACL(individual objects and folders)

access logs - all the request made to S3 buckets, to another bucket or another account's S3 bucket

Encryption

(1) In Transit - SSL / TLS

(2) Data at rest

Server Side Encryption

SSE- S3 Server Side Encryption with S3 managed keys, (amazon AES 256 handled for you) - click on the object and encrypt

SSE - KMS - AWS Key management services , managed keys - additional charges / audit trail of keys, amazon manage keys

SSE - C - Server side encryption with Customer provided keys - you manage encryption keys

Client Side Encryption

you encrypt the data on client side and upload to s3

Every non-anonymous request to S3 must contain authentication information to establish the identity of the principal making the request. In REST, this is done by first putting the headers in a canonical format, then signing the headers using your AWS Secret Access Key.

You can insert a presigned url into a webpage to download private data directly from S3.

The object creation REST APIs (see Specifying Server-Side Encryption Using the REST API) provide a request header, x-amz-server-side-encryption that you can use to request server-side encryption.

S3 Transfer Acceleration

Utilize local edge locations to upload content to S3 - incur extra cost

further away you are the more benefit you get (faster)

GateWay

(1) Gateway stored volumes - entire dataset is stored onsite and asynchronously backed up to S3

(2) Gateway cached volumes - Most frequently used data is stored onsite and entire dataset is stored on S3

(3) Gateway Virtual Taped library - Used for backup if you don't want to use Tapes, like Netbackups etc..

Import Export

Import / Export Disk

    Import to S3, EBS, Glacier

    export from S3

Import / Export Snowball

    Import to S3

    Export to S3

S3 stored data in alphabetical / lexigraphical order. so if you want to spread the load across S3, filename should not be similar (Optimize performance)

CloudFront

Content Delivery Network - edge locations, reduced latency, traffic serves from the closest nodes

Edge locations - content will be cached (over 50), different from region / AZ. TTL (speed of image // media is quicker - first user suffers the performance), can be not only read only (you can write it)

Origin can be - S3, EC2, ELB, Route53 also NON AWS origin server ,

Distribution - name given to the CDN consist of collection of Edge locations

(1) Web Distribution -

(2) RTMP (media streaming / flash) Distribution - for Adobe flash files only

you can have multiple origins of a distribution

Path Pattern (*)

Restrict viewer access by signed URL or Signed Cookies

Restrict content based on geo location (whitelist and blacklist)

Create invalidate - invalidate TTL (you pay for it) like purge in Akamai

VPC - logical datacenters in AWS

Can span multiple AZ, but can't span multiple regions, PEER VPC, but no Transitive Peering

Custom VPC has to be /16 can't go higher then that /8 is not allowed

When you create Custom VPC it creates default security group, default network ACL and default route table., it doesn't create default Subnet

One Subnet == one AZ, you can have security group spanning multiple AZ, ACL's span across AZ (assign sg and ACL to two different subnets)

any CIDR block 5 reserved IPs (.0, .1, .2, .3, .255)

so for CIRD block /24: 2^8 - 5 = 256 - 5 = 251 available IP address space

when you create internet gateway, by default its detached, attach it to VPC then, only 1 IGW per VPC

When you create a VPC Default Routetable(Main Routable) is created where the default Routes are,

10.0.0.0/16 Local <— all subnets inside VPC will be able to talk to each other

Don't touch Main route table

Create another routetable for route out to internet (0.0.0.0/0 IGW) <— route out to the internet

Last thing you associate this new route table to one of the subnet which will make it public. (you can enable auto assign public IP for the public subnet)

1 subnet can have 1 routetable

ICMP is for ping / monitor

NAT instance and NAT gateway

NAT Instance - disable source / destination check., always behind security group, must be in public subnet, must have an EIP, ,must be a route out of the private subnet to NAT

Increase the instance size if bottleneck

Change the main route table - add a route (0.0.0.0/0 NAT Instance target)

NAT Instance is a single point of failover (put it behind a ASG),

NAT gateway - released in 2016 - amazon handled

Amazon maintains it for you, no need to handle yourself. (security patches applied by AWS)

You can just create the gateway and assign EIP (put it in public subnet) (automatically assigned)

Change the main route table - add a route (0.0.0.0/0 NAT gateway target)

No need for disable source/destination check or no need to put it behind a security group - it handles it for you.

Highly available / redundancy no need for ASG.NAT gateways are little bit costly - always use it in production scale automatically up to 10Gbps

ACL vs SG

Security groups are statefull - any inbound rule , applies to outbound as well (Only Allow rules)

    by default all inbound deny, all outbound allow

    can span across AZ

ACL are stateless -

    For default ACL, all inbound and outbound rules are allowed by default - associated with all subnets in VPC by default

for Custom ACL, all inbound and outbound traffic is denied by default - not associated with any subnet

1 subnet is only associated with ACL. granular rules for ACLs, numbered rules (recommended steps of 100)

rule no. 99 takes precedence over rule no. 100 (if 99 is blocked and 100 is allowed) 99 will be executed.

Can SPAN across AZ

Ephemeral port - 1024 - 65535 should be allowed to take traffic.

if you want to BLOCK IP address then must use ACL, because security group doesn't have deny

Bastion - keep it in public subnet to allow SSH / RDP into instances into private subnets (High availability - Bation in two public subnets and also ASG - Route 53 running Health checks on those Bastion)

VPC Flowlogs: to capture all the traffic information into logs - logs everything (create IAM role and create cloud watch log group - and log stream)

VPC Cleanup: can't delete VPC if you have active running instance or ELB is running

1. EC2 placement Group
When your fleet of EC2 instances requires high network throughput and low latency within a single availability zone.

2. You are a systems administrator and you need to monitor the health of your production environment. You decide to do this using Cloud Watch. However, you notice that you cannot see the health of every important metric in the default dashboard. When monitoring the health of your EC2 instances, for which of the following metrics do you need to design a custom CloudWatch metric?
Memory Usage

3. You can add multiple volumes to an EC2 instance and then create your own RAID 5, RAID 10, or RAID 0 configurations using those volumes.
True

4. How do you force a Failover from an RDS instance in one Availability Zone to an instance in another AZ?
By manually rebooting your RDS instance

5. I know there is a way to "transfer" an EC2 instance via AMI sharing, but is there a way to transfer ownership or share S3 buckets as well?
S3 Buckets cannot be transferred between accounts

6. You have software on an EC2 instance that needs to access both the private and public IP address of that instance. What's the best way for the software to get that information?
Look it up in the instance meta data.

7. Which of the following is the metric used to monitor the lag between the primary RDS instance and the read replica?
Replica Lag

8. As the AWS platform is PCI DSS Level 1 Certified, I can immediately deploy a website to it that can take and store credit card details without getting a delta accreditation from a QSA.
False.

9. You are planning to deploy a production database to EC2 and need to choose the best storage type. You anticipate that you will need 20,000 IOPS, and an average of 8,000 - 10,000 IOPS. What storage medium should you choose?
Provisioned IOPS

10. Can I move a reserved instance from one region to another?
No

11. I can use the AWS Console to add a role to an EC2 instance after that instance has been created and powered-up.
False

12. What is the maximum number of read replicas can you have for your MySQL database?
Five

13. A member of the Operations team has done some analysis and discovered that up to 70% of all SQS messages fail to process correctly after leaving the queue. Which of the following parameters will you suggest she change?
DelaySeconds

1) Your CloudFormation template launches a two-tier web application in us-east-1. When you attempt to create a development stack in us-west-1, the process fails.

What could be the problem?

A) The AMIs referenced in the template are not available in us-west-1.
B) The IAM roles referenced in the template are not valid in us-west-1.
C) Two ELB Classic Load Balancers cannot have the same Name tag.
D) CloudFormation templates can be launched only in a single region.

2) Your application reads commands from an SQS queue and sends them to web services hosted by your partners. When a partner's endpoint goes down, your application continually returns their commands to the queue. The repeated attempts to deliver these commands use up resources. Commands that can't be delivered must not be lost.

How can you accommodate the partners' broken web services without wasting your resources?

A) Create a delay queue and set DelaySeconds to 30 seconds.
B) Requeue the message with a VisibilityTimeout of 30 seconds.
C) Create a dead letter queue and set the Maximum Receives to 3.
D) Requeue the message with a DelaySeconds of 30 seconds.

3) Your application must write to an SQS queue. Your corporate security policies require that AWS credentials are always encrypted and are rotated at least once a week.

How can you securely provide credentials that allow your application to write to the queue?

A) Have the application fetch an access key from an Amazon S3 bucket at run time.
B) Launch the application's Amazon EC2 instance with an IAM role.
C) Encrypt an access key in the application source code.
D) Enroll the instance in an Active Directory domain and use AD authentication.

4) Which operation could return temporarily inconsistent results?

A) Getting an object from Amazon S3 after it was initially created
B) Selecting a row from an Amazon RDS database after it was inserted
C) Selecting a row from an Amazon RDS database after it was deleted
D) Getting an object from Amazon S3 after it was deleted

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved | aws.amazon.com