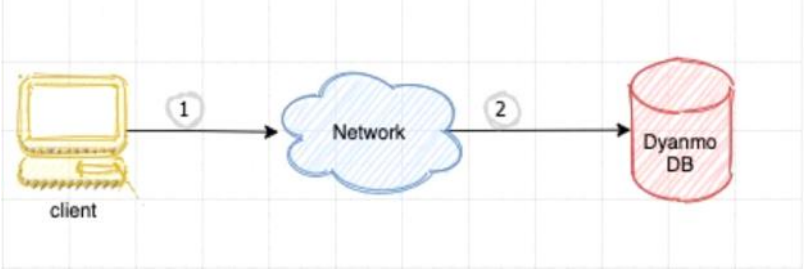
DynamoDB

16 June 2023 23:25

What is DynamoDB?

Amazon DynamoDB is a **fully managed**, **proprietary NoSQL** Database service provided by Amazon under its AWS services portfolio.
DynamoDB provides APIs allowing clients to query or make changes to the DynamoDB over the network.
DynamoDB doesn't care whether the traffic is public, from an EC2, or through a VPC.



When to use DynamoDB?

1. No Analytics is required on the data
2. Data is significantly large and consistent SLAs are required
3. **Access patterns are known**
   we exactly know the fields that will be used to query the data right now and in the foreseeable future

In relational DBMS, Data can be queried flexibly but queries are relatively expensive and also don't scale well in high traffic situations. Whereas in the case of DynamoDB, data can be queried very efficiently but only in a limited number of ways.

When Not to use DynamoDB?

1. Access patterns are not clear before designing
2. You need analytics-based queries

DynamoDB vs Other NoSQL Database (MongoDB)

MongoDB	DynamoDB
Platform Agnostic	AWS Native
Requires Infra management (unless using MongoDB Atlas)	Dedicated Infra Management not required
document sizes up to 16MB	item sizes up to 400KB

- DynamoDB also provides an option for TTL on a field
- Inbuilt support for backup and security
- Provides AWS IAM (identity and access management)

Core Components

1. Tables, items, and attributes — analogous to table, row and column in other databases
2. Primary Keys
3. Secondary Indexes
4. Streams

Primary Key

Attribute that uniquely identifies the item in the table. Primary can be of 2 types:

- **Simple primary key**: 1 attribute *(Only Partition Key)*
- **Composite primary key**: 2 attributes *(Partition Key + Sort Key)*

It is called a **partition key** because the attribute is used to find the partition that the item will belong to. The second attribute is **sort key** because for the same partition key, the items are sorted by the order of the sort key attribute.

DynamoDB hash function

DynamoDB internally employs a hash function (that is not made public) which takes the partition key as input to find out the physical storage partition that will be used for storing and querying items.
So, there are only 2 ways of querying a DynamoDB table — using partition key using a combination of partition key and sort key.

Secondary Indexes

DynamoDB table only allows us to query based on either partition key or a combination of partition + sort key.
So, what if we need to query based on some other field?
To solve this problem, DynamoDB allows us to create secondary indexes. There are 2 types of secondary indexes —
Local Secondary Index (LSI): Index that has the same partitionKey but a different sortKey

Global Secondary Index (GSI): Index with a partition key and sort key different from that of the main table

Key points

- There can only be a maximum of 20 GSIs and 5 LSIs per DynamoDB table
- Every index belongs to a table (base table)
- Indexes are maintained automatically i.e. you don't need to add/update/delete index data whenever you add/update/delete main table data.
- When Creating an index, we need to specify the projection that needs to be used. Projection defines the attributes that need to be copied from the base table to the index. Projection can be any one of — ALL, INCLUDE, and KEYS_ONLY (default)

Primary keys help uniquely identify the items in the collection(or table) whereas secondary indexes provide us with querying flexibility

DynamoDB Streams

1. Captures data modification events on the DynamoDB tables
2. Events are **almost real-time** and are in order of their occurrence
3. Each event is a stream record that contains the name of the table, metadata, and timestamp along with the operation-specific data *(ex: New data in case of insert, new+old data in case of update, etc)*
4. Each stream record is only valid for 24 hours, post which it is automatically removed

DynamoDB APIs

Control plane

APIs that help us create, manage, list tables, indexes, streams, etc

Data plane

CRUD operations on the data in the table

- putItem
- batchWriteItems: Max 25 Items can be added/deleted
- batchGetItems: Max 100 items from 1 or more table
- query: Retrieves all items that have a specific partition key
- updateItem
- deleteItem
- transactWriteItems
- transactGetItems

Read Consistency

DynamoDB supports both eventual and strongly consistent reads and the consistency level can be passed as an argument for the read operations.

By default, the reads are eventually consistent.

One can always opt for strong consistent reads but it has the following drawbacks/ disadvantages —

1. Consistent reads might not be available. In case of network delay or outage you might not get any response at all (HTTP 500)
2. relatively higher latency
3. Not supported on GSIs
4. These reads consume more throughput capacity than eventual consistent

Provisioning Capacity

Read/Write Capacity mode

We can run DynamoDB in 2 modes — on-demand and provision.

The mode decides how AWS charges us and also allows us to manage capacity.

This mode is decided at the time of table creation but can be changed later as well. However, this switch is only possible once every 24 hours.

In the case of provisioned mode, we are charged based on Read capacity units (RCU) and Write capacity units (WCU)

https://zaccharles.github.io/dynamodb-calculator/

Read capacity units (RCU) — For one 4 KB item

1 RCU will be consumed 1 strongly consistent read (or 2 eventual consistent reads)

2 RCU will be consumed for 1 transaction read

Write capacity units (WCU) — For one 1KB item

1 WCU will be consumed for 1 write

2 WCU will be consumed for 1 transactional write

Running DynamoDB Locally

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.DownloadingAndRunning.htm

High Availability and Durability

Data in DynamoDB is stored on SSDs and DynamoDB automatically spreads the data and traffic for your tables over several servers across multiple availability zones to handle your throughput and storage requirements.

Amazon DynamoDB Accelerator (DAX)

DAX is a fully managed, highly available, in-memory cache for Amazon DynamoDB that delivers up to a 10 times performance improvement.

DAX does all the heavy lifting required to add in-memory acceleration to your DynamoDB tables, developers don't need to manage cache invalidation, data population, or cluster management.

DAX is compatible with existing DynamoDB API calls, we just need to enable it through the AWS console.

Connection Aware DDB configuration

The SDKs provided by AWS for using DynamoDB allows us to configure the underlying HTTP connection properties which needs to be tuned for getting maximum performance —

- ConnectionTimeout
  ConnectionTimeout is the maximum amount of time that the client waits for the underlying HTTP client in the SDK to establish a TCP connection with the DynamoDB endpoint. The connection is an end-to-end, two-way communication link between the client and server, and it is used and reused to make API calls and receive responses. The default value of this setting is 10 seconds
- ClientExecutionTimeout
  ClientExecutionTimeout is the maximum allowed total time spent to perform an end-to-end operation and receive the desired response, including any retries that might occur. Essentially, this is the SLA of your DynamoDB operation
- RequestTimeout
  RequestTimeout is the time it takes for the client to perform a single HTTP request
- SocketTimeout
  SocketTimeout defines the maximum amount of time that the HTTP client waits to read data from an already established TCP connection. This is the time between when an HTTP POST ends and the entire response of the request is received, and it includes the service and network round-trip times.
- DynamoDB default retry policy for HTTP API calls with a custom maximum error retry count
  The default retry policy available in the AWS Java SDK for DynamoDB is a good starting point to define client-side retry strategies for the underlying HTTP client. The default policy starts with a maximum of 10 retries with a predefined base delay of 25 milliseconds for any 5XX server-side exceptions (such as "HTTP status code — 500 Internal Server Error" or "HTTP status code — 503 Service Unavailable"), and 500 milliseconds for any 4XX client-side exceptions (such as "HTTP status code 400 — ProvisionedThroughputExceededException").