Amazon DynamoDB — Basics

Aman Arora · Follow

6 min read · Jun 30, 2022

Listen Share More

This blog is a first in a multi-part series about DynamoDB. Being the first one it has been written considering people who have little to No knowledge about DynamoDB.

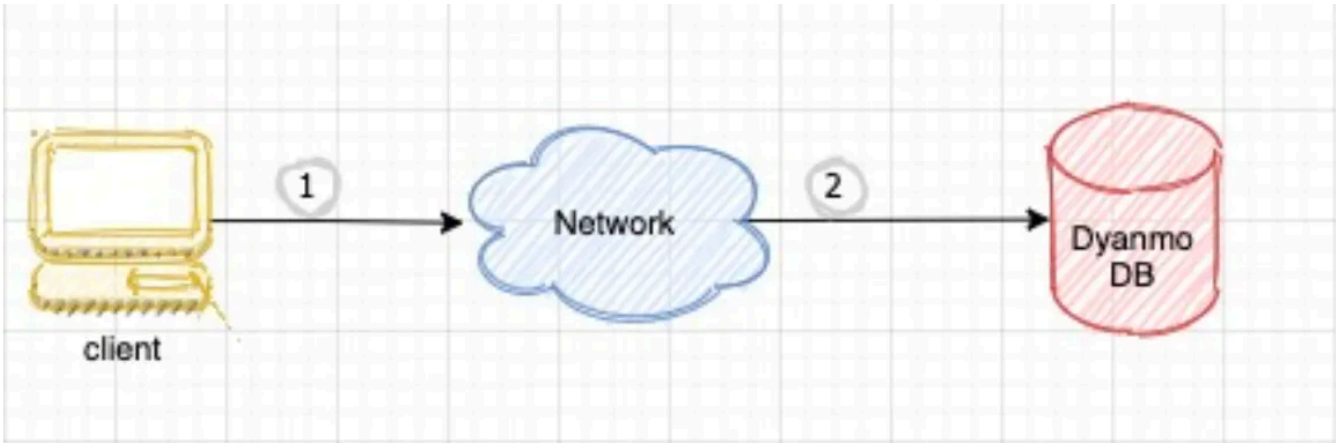
I will be covering a little advanced topics like — ***Data Modelling in DynamoDB*** and ***DynamoDB Internals*** in a later part of the series.

Happy Reading!!

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)

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”).

I hope you found this blog helpful and insightful. Thanks for reading...

You can follow me on LinkedIn by clicking here

Peace out✌️

Dynamodb AWS NoSQL

Follow

Written by Aman Arora

129 Followers · 35 Following

No responses yet

Y Yashavika Singh

What are your thoughts?

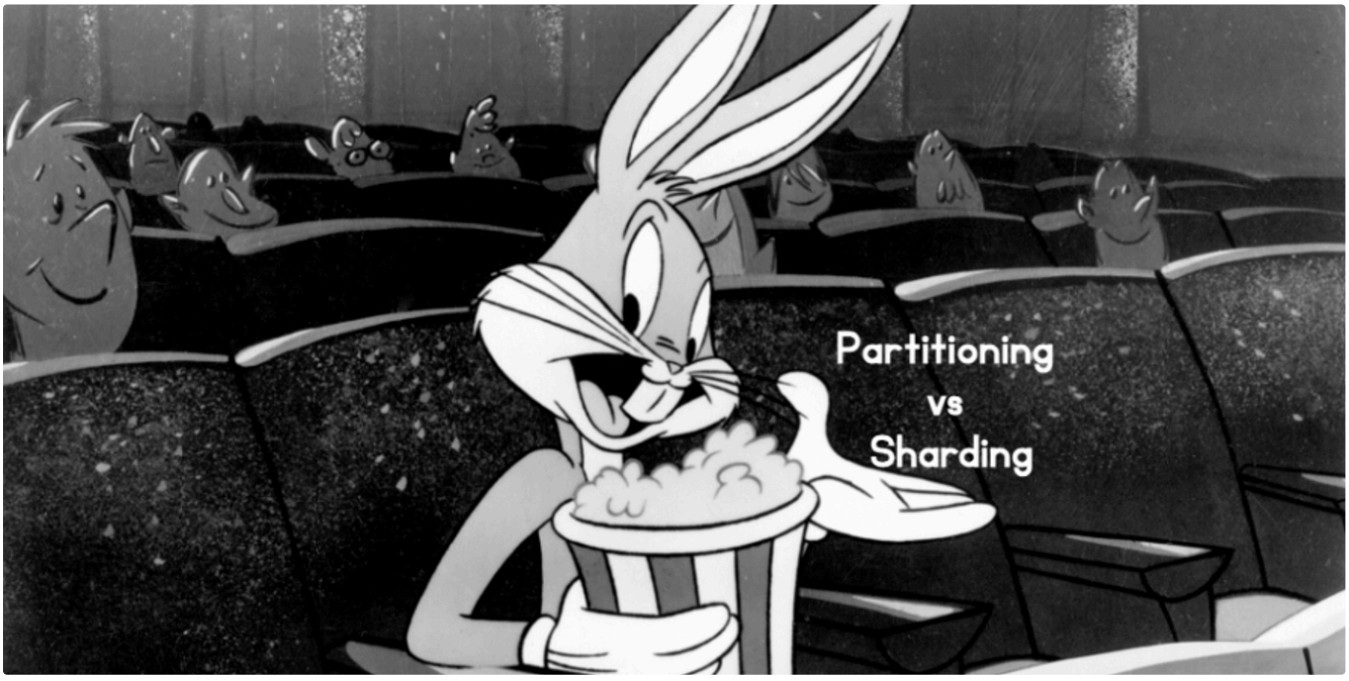
More from Aman Arora

Aman Arora

Replication in Kafka

Replication is the process of having multiple copies of the data for the sole purpose of availability in case one of the brokers goes down.

Apr 18, 2019 254 2



Aman Arora

Partitioning & Sharding — choosing the right scaling method

It is possible to use both sharding and partitioning in a distributed database system, and this approach is known as “partitioned sharding”

Oct 6, 2023 112 1

Aman Arora

What the heck is Apache Kafka ?

In Simpler terms, Apache Kafka is a Message broker i.e. it helps transmit messages from one system to another—in a real time, reliable...

Mar 25, 2019 409 2

Aman Arora

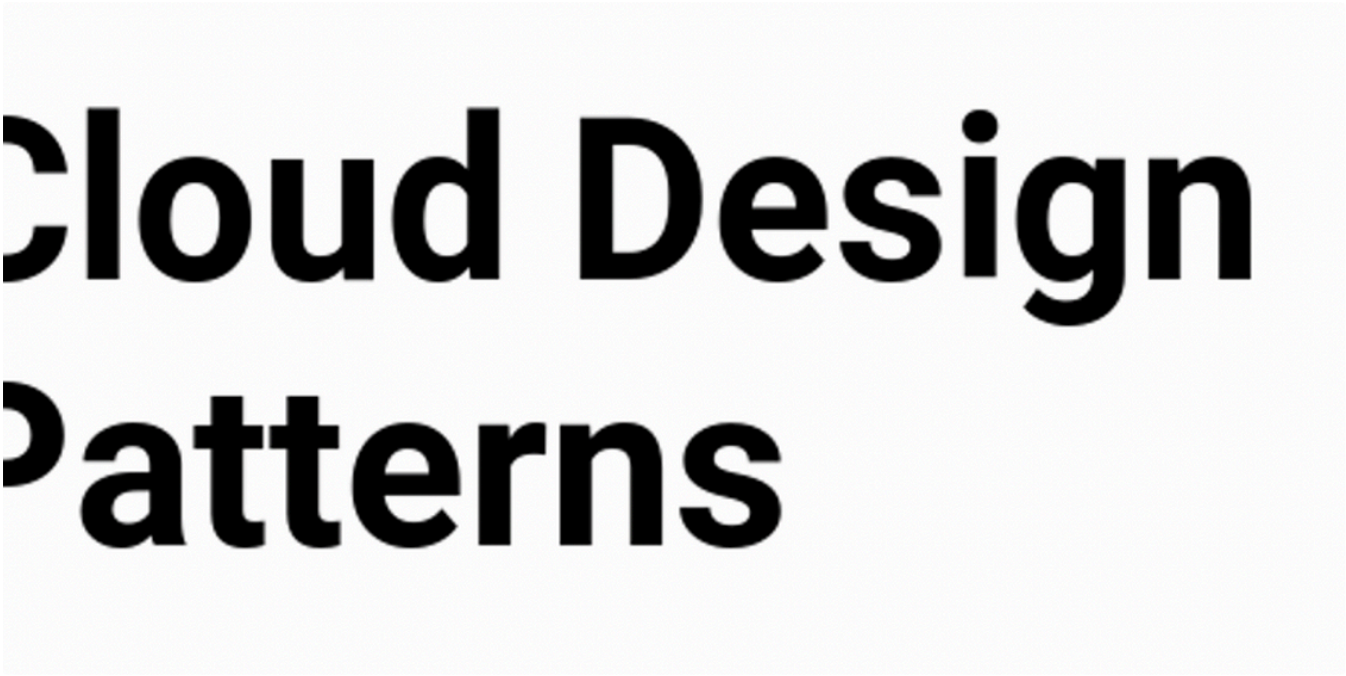
Effective Code Reviews

Code reviews are an essential part of any software development team, it helps you reduce the chances of having buggy code.

Jan 17, 2023 124 3

See all from Aman Arora

Recommended from Medium



In DevOps.dev by Mohamed ElEmam

Advanced Cloud Design Patterns for AWS and Cloud-Native Architectures

Cloud design patterns provide reusable solutions to common challenges in cloud architecture, helping organizations build highly available...

Mar 2 4

Mahtab Haider

DynamoDB User Errors: A Beginner’s Guide with AWS SDK v3 (Node.js) Code Examples

Amazon DynamoDB is a fast, fully managed NoSQL database service used widely in SaaS (Software as a Service) applications, especially...

Sep 16, 2024 1

Derek.Kim

Understanding AWS KMS: When and How to Use It

Cloud security is paramount, especially with sensitive data. AWS Key Management Service (KMS) provides a robust solution for managing...

Sep 16, 2024 9

Aspect	Batch Processing	Streaming Processing
Latency	High latency; results delivered after a delay	Low latency; results delivered in real time
Data Size	Processes large datasets in chunks	Processes smaller, continuous flows of data
Processing Speed	Suitable for periodic, time-insensitive tasks	Suitable for real-time, event-driven tasks
Complexity	Easier to implement; can handle complex transformations	More complex to implement and maintain
Fault Tolerance	Simple to implement; reprocesses entire batches	Requires advanced mechanisms for consistency and fault tolerance
Use Case Examples	Data warehouse ETL, monthly reports	Real-time analytics, fraud detection

Isaac Tonyloi

Batch vs Streaming Data: Use Cases and Trade-offs in Data Engineering

Data engineering today requires managing vast volumes of data from various sources, each with unique characteristics. Two primary...

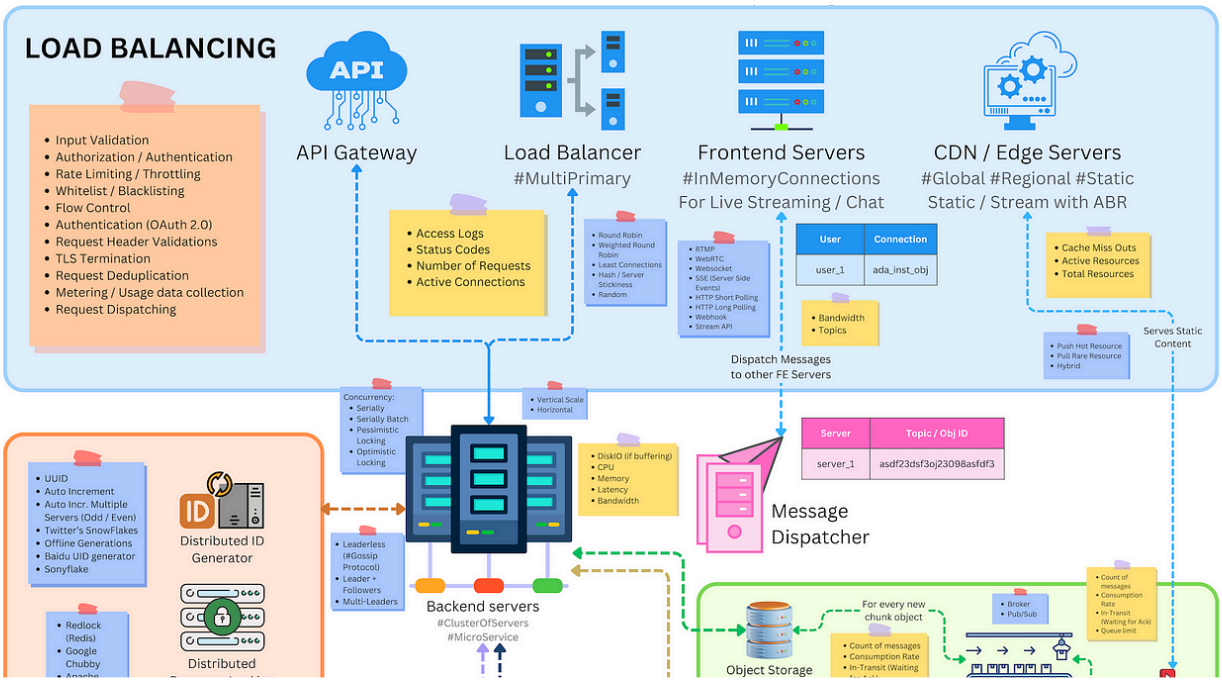
Sep 24, 2024 10

Talha Şahin

High-Level System Architecture of Booking.com

Take an in-depth look at the possible high-level architecture of Booking.com.

Jan 10, 2024 6.2K 51



In ByteByteGo System Design Alliance by Love Sharma

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

Sep 17, 2023 9.1K 57

See more recommendations

Introduction to DynamoDB

Amazon DynamoDB | CQL vs. DynamoDB API | MongoDB Datal

What is DynamoDB?

What is DynamoDB?

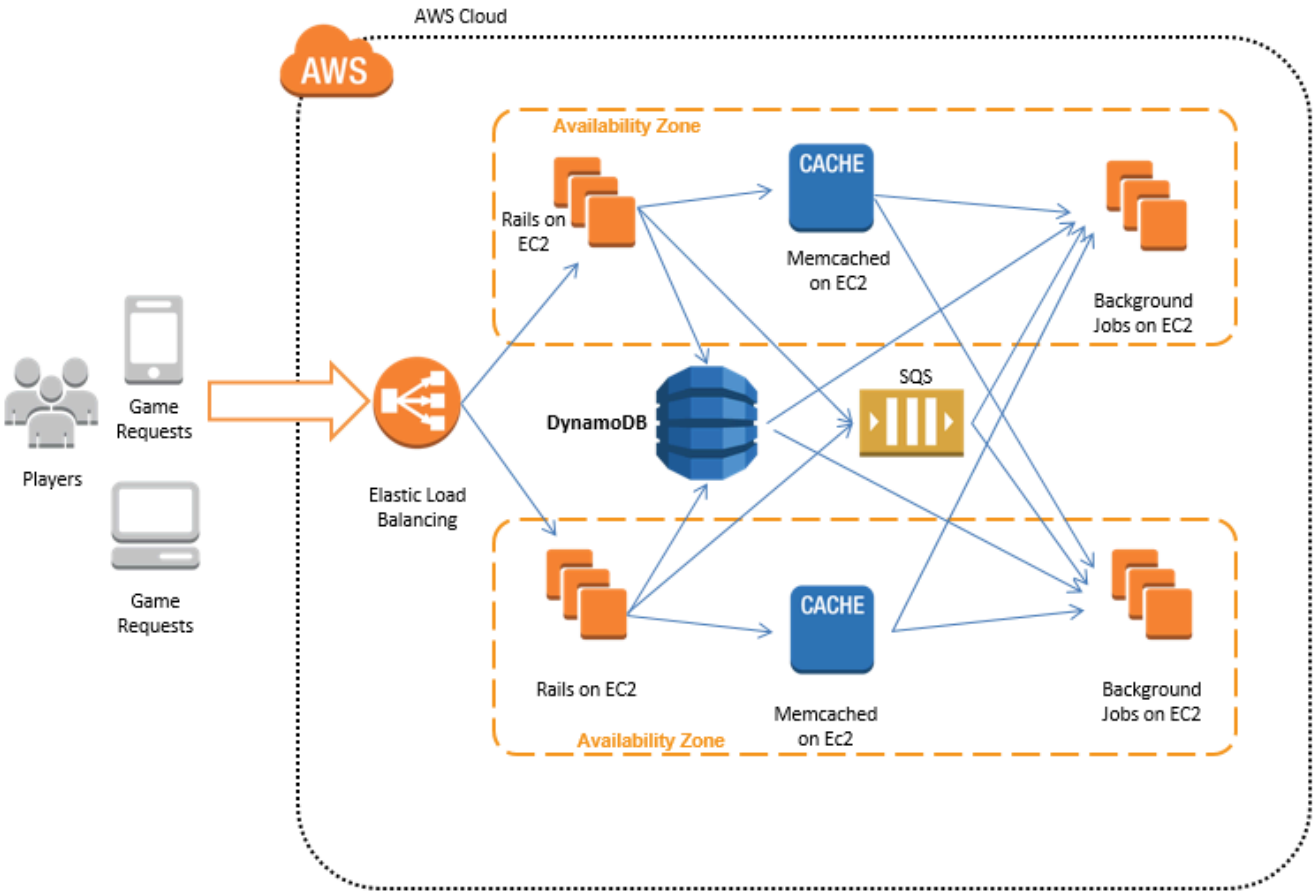
History of Amazon DynamoDB

Amazon DynamoDB is a cloud-native NoSQL primarily key-value database. Let's define each of those terms.

- DynamoDB is **cloud-native** in that it does not run on-premises or even in a hybrid cloud; it only runs on Amazon Web Services (AWS). This enables it to scale as needed without requiring a customer's capital investment in hardware. It also has attributes common to other cloud-native applications, such as elastic infrastructure deployment (meaning that AWS will provision more servers in the background as you request additional capacity).
- DynamoDB is **NoSQL** in that it does not support ANSI Structured Query Language (SQL). Instead, it uses a proprietary API based on JavaScript Object Notation (JSON). This API is generally not called directly by user developers, but invoked through AWS Software Developer Kits (SDKs) for DynamoDB written in various programming languages (C++, Go, Java, JavaScript, Microsoft .NET, Node.js, PHP, Python and Ruby).
- DynamoDB is primarily a **key-value** store in the sense that its data model consists of key-value pairs in a schemaless, very large, non-relational table of rows (records). It does not support relational database management systems (RDBMS) methods to join tables through foreign keys. It can also support a document store data model using JavaScript Object Notation (JSON).

DynamoDB's NoSQL design is oriented towards simplicity and scalability, which appeal to developers and devops teams respectively. It can be used for a wide

variety of semistructured data-driven applications prevalent in modern and emerging use cases beyond traditional databases, from the Internet of Things (IoT) to social apps or massive multiplayer games. With its broad programming language support, it is easy for developers to get started and to create very sophisticated applications using DynamoDB.



[Image Source]

What is a DynamoDB Database?

Outside of Amazon employees, the world doesn't know much about the exact nature of this database. There is a development version known as DynamoDB Local used to run on developer laptops written in Java, but the cloud-native database architecture is proprietary closed-source.

While we cannot describe exactly what DynamoDB *is*, we can describe how you *interact* with it. When you set up DynamoDB on AWS, you do not provision specific servers or allocate set amounts of disk. Instead, you provision throughput — you define the database based on provisioned capacity — how many transactions and how many kilobytes of traffic you wish to support per second. Users specify a service level of read capacity units (RCUs) and write capacity units (WCUs).

As stated above, users generally do not directly make DynamoDB API calls. Instead, they will integrate an AWS SDK into their application, which will handle the back-end communications with the server.

DynamoDB data modeling needs to be denormalized. For developers used to working with both SQL and NoSQL databases, the process of rethinking their data model is nontrivial, but also not insurmountable.

History of the Amazon DynamoDB Database

DynamoDB was inspired by the seminal Dynamo white paper (2007) written by a team of Amazon developers. This white paper cited and contrasted itself from Google's Bigtable (2006) paper published the year before.

The original Dynamo database was used internally at Amazon as a completely in-house, proprietary solution. It is a completely different customer-oriented Database as a Service (DBaaS) that runs on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances. DynamoDB was released in 2012, five years after the original white paper that inspired it.

While DynamoDB was inspired by the original paper, it was not beholden to it. Many things had changed in the world of Big Data over the intervening years since the paper was published. It was designed to build on top of a "core set of

strong distributed systems principles resulting in an ultra-scalable and highly reliable database system."

DynamoDB Release History

Since its initial release, Amazon has continued to expand on the DynamoDB API, extending its capabilities. DynamoDB documentation includes a document history as part of the DynamoDB Developer Guide to track key milestones in DynamoDB's release history:

Jan 2012 — Initial release of DynamoDB
Aug 2012 — Binary data type support
Apr 2013 — New API version, local secondary indexes
Dec 2013 — Global secondary indexes
Apr 2014 — Query filter, improved conditional expressions
Oct 2014 — Larger item sizes, flexible scaling, JSON document model
Apr 2015 — New comparison functions for conditional writes, improved query expressions
Jul 2015 — Scan with strongly-consistent reads, streams, cross-region replication

Feb 2017 — Time-to-Live (TTL) automatic expiration
Apr 2017 — DynamoDB Accelerator (DAX) cache
Jun 2017 — Automatic scaling
Aug 2017 — Virtual Private Cluster (VPC) endpoints
Nov 2017 — Global tables, backup and restore
Feb 2018 — Encryption at rest
Apr 2018 — Continuous backups and Point-in-Time Recovery (PITR)
Nov 2018 — Encryption at rest, transactions, on-demand billing
Feb 2019 — DynamoDB Local (downloadable version for developers)
Aug 2019 — NoSQL Workbench preview
Nov 2019 — PartiQL support – SQL-compatible query language for DynamoDB
May 2020 — NoSQL Workbench general availability
Jul 2020 — Contributor Insights for monitoring hot keys and usage patterns
Nov 2020 — Export to S3 feature for full table exports
Apr 2021 — Increased table limits and improved replica auto-scaling
Jul 2021 — Scheduled automatic backups
Dec 2021 — Standard-Infrequent Access (Standard-IA) table class
Mar 2022 — Increased maximum item size to 400KB
Sep 2022 — Zero-ETL integration with Amazon OpenSearch Service
Apr 2023 — Import from S3 feature
Sep 2023 — Verified Access integration for enhanced security
Nov 2023 — Multi-Region Key-Value API for simplified global tables
Feb 2024 — Enhanced CloudWatch integration for monitoring

DynamoDB Database Overview

DynamoDB Design Principles

As stated in Werner Vogel's initial blog about DynamoDB, the database was designed to build on top of a "core set of strong distributed systems principles resulting in an ultra-scalable and highly reliable database system." It needed to provide these attributes:

- **Managed** — provided 'as-a-Service' so users would not need to maintain the database
- **Scalable** — automatically provision hardware on the backend, invisible to the user
- **Fast** — support predictive levels of provisioned throughput at relatively low latencies
- **Durable and highly available** — multiple availability zones for failures/disaster recovery
- **Flexible** — make it easy for users to get started and continuously evolve their database

- **Low cost** — be affordable for users as they start and as they grow

The database needed to “provide fast performance at any scale,” allowing developers to “start small with just the capacity they need and then increase the request capacity of a given table as their app grows in popularity.” Predictable performance was ensured by provisioning the database with guarantees of throughput, measured in “capacity units” of reads and writes. “Fast” was defined as single-digit milliseconds, based on data stored in Solid State Drives (SSDs).

It was also designed based on lessons learned from the original Dynamo, SimpleDB and other Amazon offerings, “to reduce the operational complexity of running large database systems.” Developers wanted a database that “freed them from the burden of managing databases and allowed them to focus on their applications.” Based on Amazon’s experience with SimpleDB, they knew that developers wanted a database that “just works.” By its design users no longer were responsible for provisioning hardware, installing operating systems, applications, containers, or any of the typical devops tasks for on-premises deployments. On the back end, DynamoDB “automatically spreads the data and traffic for a table over a sufficient number of servers to meet the request capacity specified by the customer.” Furthermore, DynamoDB would replicate data across multiple AWS Availability Zones to provide high availability and durability.

As a commercial managed service, Amazon also wanted to provide a system that would make it transparent for customers to predict their operational costs.

DynamoDB Data Storage Format

To manage data, DynamoDB uses hashing and b-trees. While DynamoDB supports JSON, it only uses it as a transport format; JSON is not used as a storage format. Much of the exact implementation of DynamoDB's data storage format remains proprietary. Generally a user would not need to know about it. Data in DynamoDB is generally exported through streaming technologies or bulk downloaded into CSV files through ETL-type tools like AWS Glue. As a managed service, the exact nature of data on disk, much like the rest of the system specification (hardware, operating system, etc.), remains hidden from end users of DynamoDB.

DynamoDB Query Operations

Unlike Structured Query Language (SQL), DynamoDB queries use a Javascript Object Notation (JSON) format to structure its queries. For example, as seen on this page), if you had the following SQL query:

```
SELECT * FROM Orders
INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
INNER JOIN Products ON Products.Product_ID =
Order_Items.Product_ID
INNER JOIN Inventories ON Products.Product_ID =
Inventories.Product_ID
ORDER BY Quantity_on_Hand DESC
```

In DynamoDB you would redesign the data model so that everything was in a single table, denormalized, and without JOINs. This leads to the lack of deduplication you get with a normalized database, but it also makes a far simpler query. For example, the equivalent of the above in a DynamoDB query could be rendered as simply as:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

It is important to remember that DynamoDB does not use SQL at all. Nor does it use the NoSQL equivalent made popular by Apache Cassandra, Cassandra Query Language (CQL). Instead, it uses JSON to encapsulate queries.

```
{
    TableName: "Table_X"
    KeyConditionExpression: "LastName = :a",
    ExpressionAttributeValues: {
        :a = "smith"
    }
}
```

However, just because DynamoDB keeps data in a single denormalized table doesn't mean that it is simplistic. Just as in SQL, DynamoDB queries can get very complex. This example provided by Amazon shows just how complex a JSON request can get, including data types, conditions, expressions, filters, consistency levels and comparison operators.

NoSQL Data Models

DynamoDB is a key-value NoSQL database. Primarily This makes it Amazon's preferred database for simple and fast data models, such as managing user profile data, or web session information for applications that need to quickly

retrieve any amount of data at Internet scale. This database is so popular for this use case it is ranked one in the top 20 databases listed on DB-Engines.com.

Additional NoSQL Data Models

While DynamoDB models data in JSON format, allowing it to serve in document use cases Amazon recommends users to its Amazon Document DB, which is designed-for-purpose as a document store, and is also compatible with the widely-adopted MongoDB API. For example, DynamoDB doesn't store data internally as JSON. It only uses JSON as a transport method.

The database also supports wide column data models when it acts as an underlying data storage model in its serverless Managed Cassandra Service. However, in this mode users call the database using the Cassandra Query Language (CQL) just as they would with any native Apache Cassandra database. Users cannot make any DynamoDB API calls to the underlying database. You can read a further analysis of this service in this blog.

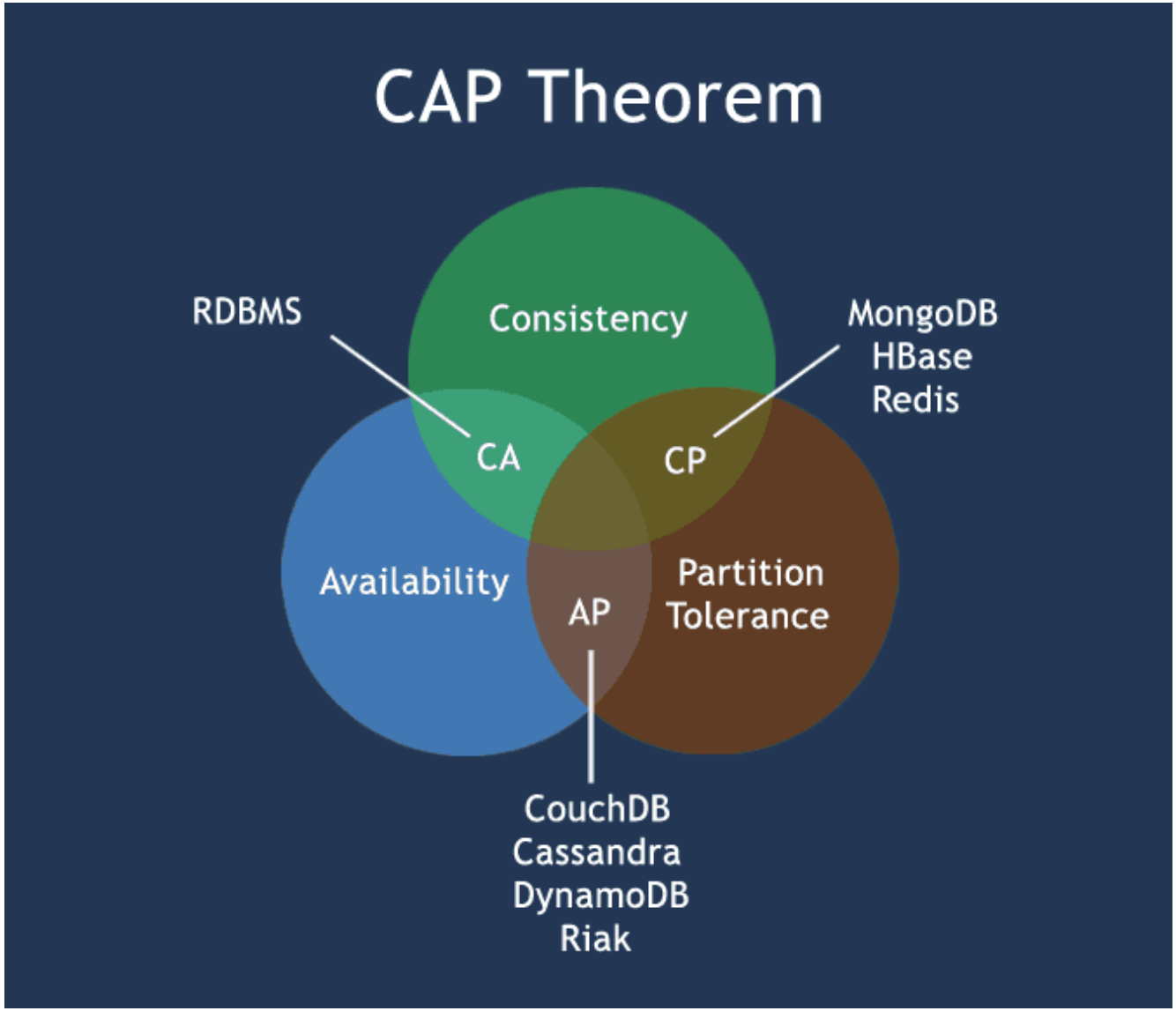

CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

Image Source

DynamoDB and the CAP Theorem

The CAP Theorem (as put forth in a presentation by Eric Brewer in 2000) stated that distributed shared-data systems had three properties but systems could only choose to adhere to two of those properties:

- Consistency
- Availability
- Partition-tolerance

Distributed systems designed for fault tolerance are not much use if they cannot operate in a partitioned state (a state where one or more nodes are unreachable). Thus, partition-tolerance is always a requirement, so the two basic modes that most systems use are either Availability-Partition-tolerant ("AP") or Consistency-Partition-tolerant ("CP").

An "AP"-oriented database remains available even if it was partitioned in some way. For instance, if one or more nodes went down, or two or more parts of the cluster were separated by a network outage (a so-called "split-brain" situation), the remaining database nodes would remain available and continue to respond to requests for data (reads) or even accept new data (writes). However, its data

would become inconsistent across the cluster during the partitioned state. Transactions (reads and writes) in an “AP”-mode database are considered to be “eventually consistent” because they are allowed to write to some portion of nodes; inconsistencies across nodes are settled over time using various anti-entropy methods.

A “CP”-oriented database would instead err on the side of consistency in the case of a partition, even if it meant that the database became unavailable in order to maintain its consistency. For example, a database for a bank might disallow transactions to prevent it from becoming inconsistent and allowing withdrawals of more money than were actually available in an account. Transactions on such systems are referred to as “strongly consistent” because all nodes on a system need to reflect the change before the transaction is considered complete or successful.

It was initially designed to operate primarily as an “AP”-mode database. However later Amazon introduced DynamoDB Transactions to allow the database to act in a manner similar to a “CP”-mode database. These sorts of transactions are important to perform conditional updates — for example, ensuring a record meets a certain condition before setting it to a new value. (Such as having sufficient money in an account before making a withdrawal.) If the condition is not met, then the transaction does not proceed. This type of transaction requires a read-before-write (to check for existing values), and also a subsequent check to ensure the update went through properly. While

providing this level of “all-or-nothing” operational guarantee transaction performance will naturally be slower, and may in fact fail at times. For example, users may notice DynamoDB system errors such as returning HTTP 500 server errors.

DynamoDB, ACID and BASE

An ACID database is a database that provides the following properties:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

DynamoDB, when using DynamoDB Transactions, displays ACID properties.

However, without the use of transactions, DynamoDB is usually considered to display BASE properties:

- **B**asically **A**vailable

- **S**oft-state
- **E**ventually consistent

DynamoDB Scalability and High Availability

DynamoDB scalability includes methods such as autosharding and load-balancing. Autosharding means that when load on a single Amazon server gets to a certain point, the database can select a certain amount of records and place that data on a new node. Traffic between the new and existing servers is load-balanced so that, ideally, no one node is impacted with more traffic than others. However, the exact methods of how the database supports autosharding and load-balancing are proprietary, part of its internal operational mechanics, and are not visible to nor controllable by users.

Amazon DynamoDB Data Modeling

As mentioned before, DynamoDB is a key-value store database that uses a documented-oriented JSON data model. Data is indexed using a primary key composed of a partition key and a sort key. There is no set schema to data in the same table; each partition can be very different from others. Unlike traditional SQL systems where data models can be created long before needing to know how the data will be analyzed, with DynamoDB, like many other NoSQL databases, data should be modeled based on the types of queries you seek to run.

- Documentation for Partitions and Data Distribution in DynamoDB.
- Learn more about the DynamoDB data model and recommendations for partition keys.
- Read more DynamoDB best practices for designing and using partition keys effectively.

DynamoDB Architecture for Data Distribution

Amazon Web Services (AWS) guarantees that DynamoDB tables span Availability Zones. You can also distribute your data across multiple regions in the database global tables to provide greater resiliency in case of a disaster. However, with global tables you need to keep your data eventually consistent.

When to Use DynamoDB

Amazon DynamoDB is most useful when you need to rapidly prototype and deploy a key-value store database that can seamlessly scale to multiple gigabytes or terabytes of information — what are often referred to as “Big Data” applications. Because of its emphasis on scalability and high availability DynamoDB is also appropriate for “always on” use cases with high volume transactional requests (reads and writes).

DynamoDB is inappropriate for extremely large data sets (petabytes) with high frequency transactions where the cost of operating DynamoDB may make it prohibitive. It is also important to remember DynamoDB is a NoSQL database that uses its own proprietary JSON-based query API, so it should be used when data models do not require normalized data with JOINs across tables which are more appropriate for SQL RDBMS systems.

Amazon DynamoDB Ecosystem

DynamoDB can be developed using Software Development Kits (SDKs) available from Amazon in a number of programming languages.

- C++
- Clojure
- Coldfusion
- Erlang
- F#

- Go
- Groovy/Rails

- Java
- JavaScript
- .NET

- Node.js
- PHP
- Python
- Ruby
- Scala

There are also a number of integrations for DynamoDB to connect with other AWS services and open source big data technologies, such as Apache Kafka, and Apache Hive or Apache Spark via Amazon EMR.

DynamoDB Alternatives and Variants

DynamoDB is a proprietary, closed source offering. There is currently only one DynamoDB API-compatible database alternative on the marketplace: ScyllaDB, the fastest NoSQL database. ScyllaDB's Project Alternator interface allows it to

appear like a DynamoDB database to clients; users can migrate their data to ScyllaDB without changing their data model or queries.

ScyllaDB can run on any cloud or on premises, as well as an Enterprise version and a hosted Database as a Service: ScyllaDB Cloud.

There are no other direct alternatives to DynamoDB. While different NoSQL databases can provide key-value or document data models, it would require re-architecting data models and redesigning queries to achieve a DynamoDB migration to another database.

For example, you could migrate from DynamoDB to ScyllaDB using its Cassandra Query Language (CQL) interface, but it would require redesign of your data model, as well as completely rewriting your existing queries from DynamoDB's JSON format to CQL. While this may be advantageous to take advantage of various features of ScyllaDB currently available only through its CQL interface, this requires more of a reengineering effort than a simple "lift and shift" migration.

DynamoDB Local

DynamoDB local enables teams to develop and test applications in a dev/test environment. It is not intended or suitable for running DynamoDB on-premises in production.

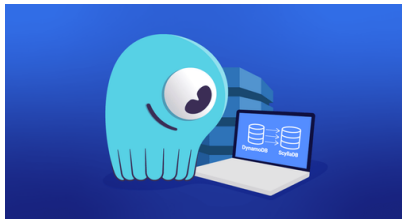
DynamoDB local lets you develop and test applications without accessing the DynamoDB web service. As Amazon's DynamoDB local documentation explains, this "helps you save on throughput, data storage, and data transfer fees." It also enables you to work on your application while you're offline.

When working with DynamoDB local, it is important to note that it is not identical to the cloud version. There are a number of differences, some trivial and some rather significant. The key differences are outlined in Amazon's DynamoDB Developer's Guide. Additionally, since DynamoDB local is not designed for production, it does not provide high availability.

If you want to run DynamoDB on-prem in production, this can be accomplished with an open source DynamoDB compatible-API such as ScyllaDB Alternator. That's exactly what GE Healthcare did when they needed to take their DynamoDB-based Edison AI workbench on-prem to support potential research customers who needed to run in the hospitals' own networks.

For a step-by-step look at what's involved in running a DynamoDB app locally on-prem, watch the video: Build DynamoDB-Compatible Apps with Python.

Trending DynamoDB Resources

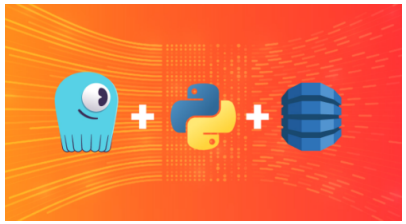


Moving From DynamoDB To ScyllaDB

Watch Video >

DynamoDB Cost Optimization Masterclass

Access Now >



Workshop: Run-Anywhere DynamoDB-Compatible Apps W/ Python

Watch Video >

Next Article:
DynamoDB - Local Installation

DynamoDB – Introduction

Last Updated : 02 Feb, 2022

DynamoDB allows users to create databases capable of storing and retrieving any amount of data and comes in handy while serving any amount of traffic. It dynamically manages each customer's requests and provides high performance by automatically distributing data and traffic over servers. It is a fully managed NoSQL database service that is fast, predictable in terms of performance, and seamlessly scalable. It relieves the user from the administrative burdens of operating and scaling a distributed database as the user doesn't have to worry about hardware provisioning, patching Softwares, or cluster scaling. It also eliminates the operational burden and complexity involved in protecting sensitive data by providing encryption at REST.

The below table provides us with core differences between a conventional relational database management system and AWS DynamoDB:

Operations	DynamoDB	RDBMS
Source connection	It uses HTTP requests and API operations.	It uses a persistent connection and SQL commands.
Create Table	It mainly requires the Primary key and no schema on the creation and can have various data sources.	It requires a well-defined table for its operations.
Getting Table Information	Only Primary keys are revealed.	All data inside the table is accessible.

Operations	DynamoDB	RDBMS
Loading Table Data	In tables, it uses items made of attributes.	It uses rows made of columns.
Reading Table Data	It uses GetItem, Query, and Scan	It uses SELECT statements and filtering statements.
Managing Indexes	It uses a secondary index to achieve the same function. It requires specifications (partition key and sort key).	Standard Indexes created by SQL is used.
Modifying Table Data	It uses a UpdateItem operation.	It uses an UPDATE statement.

Operations	DynamoDB	RDBMS
Deleting Table Data	It uses a DeleteItem operation.	It uses a DELETE statement.
Deleting Table	It uses a DeleteTable operation.	It uses a DROP TABLE statement.

Advantage of DynamoDB:

The main advantages of opting for Dynamodb are listed below:

- It has fast and predictable performance.
- It is highly scalable.
- It offloads the administrative burden operation and scaling.
- It offers encryption at REST for data protection.
- Its scalability is highly flexible.
- AWS Management Console can be used to monitor resource utilization and performance metrics.
- It provides on-demand backups.
- It enables point-in-time recovery for your Amazon

operations. With point-in-time recovery, you can restore that table to any point in time during the last 35 days.
- It can be highly automated.

Limitations of DynamoDB –

The below list provides us with the limitations of Amazon DynamoDB:

- It has a low read capacity unit of 4kB per second
- All tables and global secondary indexes must have a minimum of one read and one write capacity unit.
- Table sizes have no limits, but accounts have a 256 table limit unless you request a higher cap.
- Only Five local and twenty global secondary (default quota) indexes per table are permitted.
- DynamoDB does not prevent the use of reserved words as names.
- Partition key length and value minimum length sits at 1 byte, and maximum at 2048 bytes, however, DynamoDB places no limit on values

Next Article

DynamoDB - Local Installation

Similar Reads

DynamoDB - Introduction

DynamoDB allows users to create databases capable of storing and retrieving any amount of data and comes in handy while serving any amoun...

3 min read

DynamoDB - Local Installation

The DynamoDB setup only includes the access of your AWS account through which the DynamoDB GUI console can be accessed. However, a local...

3 min read

AWS DynamoDB - Introduction to DynamoDB Accelerator (DAX)

DynamoDB is a fast NoSQL Database that is managed by Amazon Web Services (AWS). It was developed by Amazon Web Services (AWB)....

4 min read

AWS DynamoDB - Working with Indexes

2 min read

DynamoDB - Using the Console

The AWS management console for Amazon DynamoDB can be accessed from here. The AWS management console can be used for the following:...

4 min read

AWS DynamoDB - PartiQL Insert Statement

PartiQL is a SQL-compatible query language that supports querying, modifying, and inserting data. It makes it easier to interact with DynamoDB...

1 min read

AWS DynamoDB - Working with Scans

Amazon DynamoDB is NoSQL managed database that stores semi-structured data like key-value pairs and document data. When creating...

3 min read

AWS DynamoDB - Working with Tables

In this article, we will work on DynamoDB tables. DynamoDB is a NoSQL database that stores document data or key-value pairs. A Dynamodb table...

3 min read

Delete Table In DynamoDB

DynamoDB allows users to create databases capable of storing and retrieving any amount of data and comes in handy while serving any amoun...

4 min read

DynamoDB Streams is a DynamoDB feature that allows users to keep track of any changes made to the data in DynamoDB. It is an "ordered flow of dat...

3 min read

Understanding the basics of using DynamoDB

Part 1 of 3: An introduction to using and cost optimizing DynamoDB

Srivatssan Srinivasan

June 15, 2022

Amazon DynamoDB is a fully managed NoSQL database service that lets you offload the administrative burdens of operating and scaling a distributed database. In this three-part series, I am going to walk you through the basics of DynamoDB and show you some best practices that could save you some operational expense while using this Amazon service.

This article–the first of a three part series–is an introductory refresher focusing on the basic components of DynamoDB. In this blog post I will cover:

- The basic components of DynamoDB
- The basic structure of DynamoDB
- The importance of keys
- Indexes

- Instance types

If you want to jump deeper into the series, use the links below:

- Part 2: 10 DynamoDB choices that will impact your costs
- Part 3: 9 recommendations to minimize DynamoDB operational costs

Core components of DynamoDB tables, items and attributes

In DynamoDB, tables, items and attributes are the core components that you work with. Simply put, a table is a collection of items and each item is a collection of attributes. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.

Tables

Similar to other database systems, DynamoDB stores data in tables. A table is a collection of data.

Items

Each table contains zero or more items. An item is a group of attributes that is uniquely identifiable among all of the other items. In DynamoDB, there is no limit to the number of items you can store in a table. Items are like rows in a relational database.

Attributes

Each item is composed of one or more attributes. An attribute is a fundamental data element, something that does not need to be broken down any further. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

Basic structure of DynamoDB

Here are some things to understand about the basic structure of DynamoDB.

- Each item in the table has a unique identifier, or primary key, that distinguishes the item from all of the others in the table.
- Other than the primary key, a table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- Most of the attributes are scalar, which means that they can have only one value. Strings and numbers are common examples of scalars.
- Some of the items have a nested attribute (Address). DynamoDB supports nested attributes up to 32 levels deep.

Significance of Keys in DynamoDB

When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key.

DynamoDB supports two different kinds of primary keys–partition keys and composite primary keys.

Partition key

A simple primary key is composed of one attribute. DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. In a table that has only a partition key, no two items can have the same partition key value.

Composite primary key

A composite key contains a partition key and sort key, this type of key is composed of two attributes. The first attribute is the partition key, and the second attribute is the sort key. Here DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. All items with the same partition key value are stored together, in sorted order by sort key value.In a table that has a partition key and a sort key, it's possible for multiple items to have the same partition key value. However, those items must have different sort key values.

Indexes in DynamoDB

DynamoDB supports two kinds of indexes–global secondary index (GSI) and local secondary index (LSI):

Global secondary index (GSI)

A GSI is an index with a partition key and sort key that can be different from those on the table.

Local secondary index (LSI)

A LSI is an index that has the same partition key as the table, but a different sort key.

Each table in DynamoDB has a quota of 20 global secondary indexes (default quota) and 5 local secondary indexes.

Instance types–on-demand and provisioned

On-demand instances

When to choose an on-demand instance? On-demand mode is a good option if any of the following are true:

- You are creating new tables with unknown workloads.
- You have unpredictable application traffic.
- You prefer the ease of paying for only what you use.

For on-demand mode tables, you don't need to specify how much read and write throughput you expect your application to perform. DynamoDB charges you for the reads and writes that your application performs on your tables in terms of read request units and write request units.

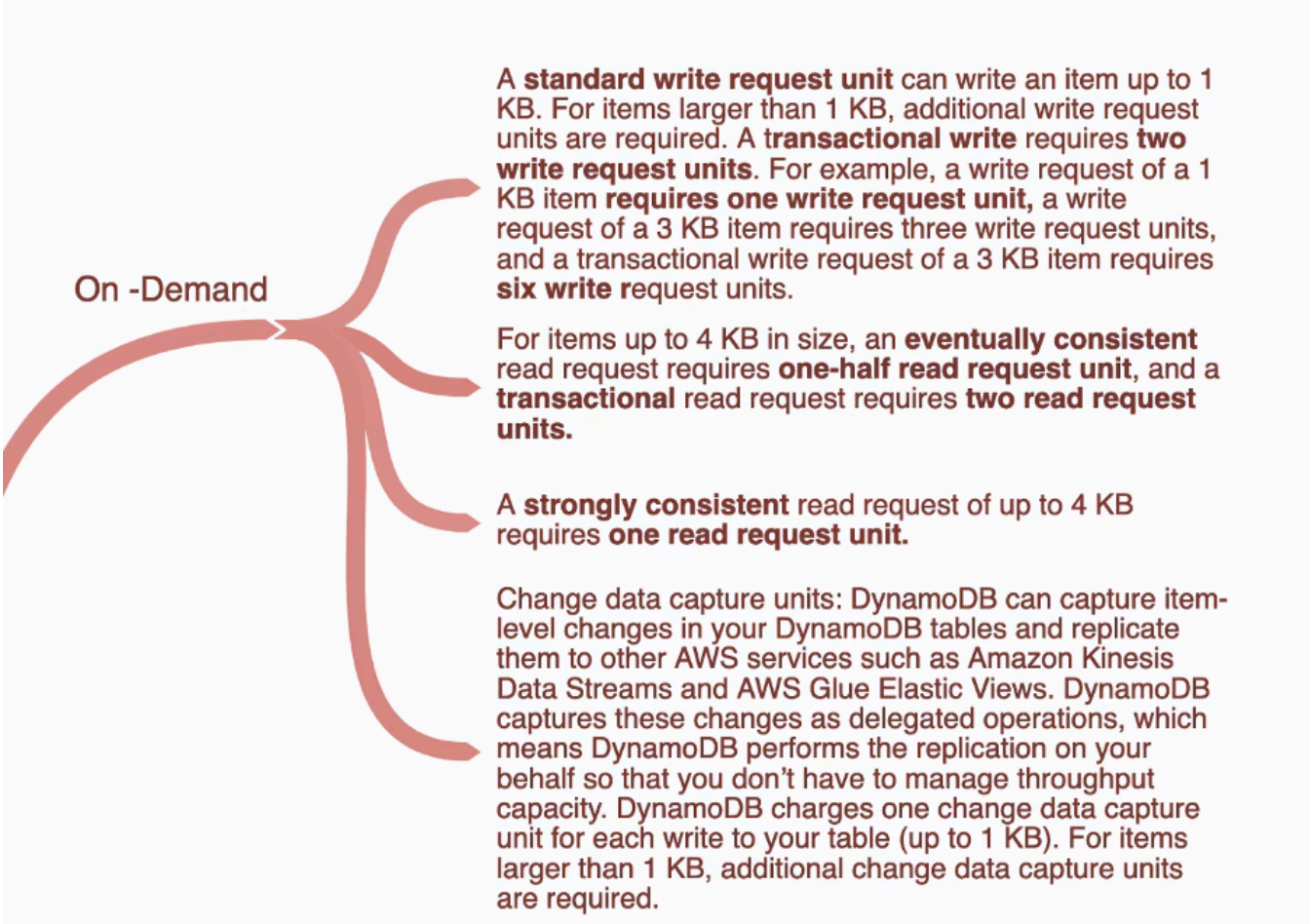


Figure 1: Cost of on-demand usage from Reads and Writes

Key information about on-demand

Read request unit

- One read request unit represents one strongly consistent read request, or two eventually consistent read requests, for an item up to 4 KB in size.
- Two read request units represent one transactional read for items up to 4 KB.
- If you need to read an item that is larger than 4 KB, DynamoDB needs additional read request units. The total number of read request units required depends on the item size, and whether you want an eventually consistent or strongly consistent read.

- For example, if your item size is 8 KB, you require two read request units to sustain one strongly consistent read, one read request unit if you choose eventually consistent reads, or four read request units for a transactional read request.

Write request unit

- One write request unit represents one write for an item up to 1 KB in size.
- If you need to write an item that is larger than 1 KB, DynamoDB needs to consume additional write request units.
- Transactional write requests require two write request units to perform one write for items up to 1 KB.
- The total number of write request units required depends on the item size.
- For example, if your item size is 2 KB, you require two write request units to sustain one write request or four write request units for a transactional write request.

Initial throughput for on-demand capacity mode

- Newly created table with on-demand capacity mode:
  - Assume: previous peak was 2,000 write request units or 6,000 read request units.
  - You can drive up to double the previous peak immediately, which enables newly created on-demand tables to serve up to 4,000 write request units or 12,000 read request units, or any linear combination of the two.
- Existing table switched to on-demand capacity mode:
  - The previous peak is half the maximum write capacity units and read capacity units provisioned since the table was created,
  - or the settings for a newly created table with on-demand capacity mode, whichever is higher.
  - In other words, your table will deliver at least as much throughput as it did prior to switching to on-demand capacity mode.

Capacity switching impacts

- When you switch a table from provisioned capacity mode to on-demand capacity mode, DynamoDB makes several changes to the structure of your table and partitions. This process can take several minutes.
- During the switching period, your table delivers throughput that is consistent with the previously provisioned write capacity unit and read capacity unit amounts.
- When switching from on-demand capacity mode back to provisioned capacity mode, your table delivers throughput consistent with the previous peak reached when the table was set to on-demand capacity mode.

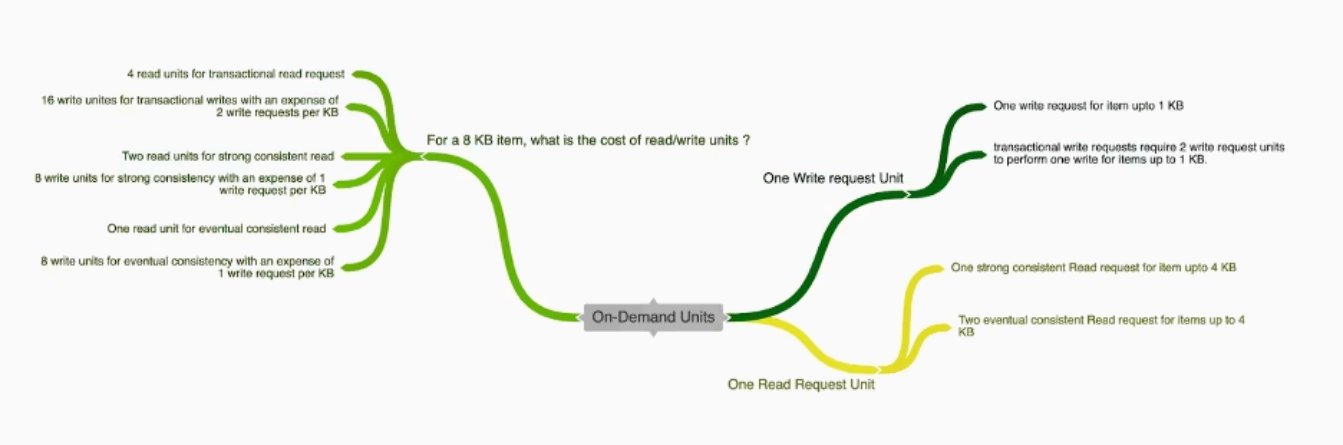


Figure 2: Cost of on-demand usage from Reads and Writes

Provisioned instances

If you choose provisioned mode, you must specify the number of reads and writes per second that you require for your application. You can use auto scaling to adjust your table's provisioned capacity automatically in response to traffic changes. This helps you govern your DynamoDB use to stay at or below a defined request rate in order to obtain cost predictability. You can use provisioned capacity when:

- You have predictable application traffic.
- You run applications whose traffic is consistent or ramps gradually.
- You can forecast capacity requirements to control costs.

Request throttling

Provisioned throughput is the maximum amount of capacity that an application can consume from a table or index. If your application exceeds your provisioned throughput capacity on a table or index, it is subject to request throttling.

Throttling prevents your application from consuming too many capacity units. When a request is throttled, it fails with an HTTP 400 code (Bad Request) and a ProvisionedThroughputExceededException.

DynamoDB auto-scaling

This is enabled by default. Make sure you configure the function to your needs.

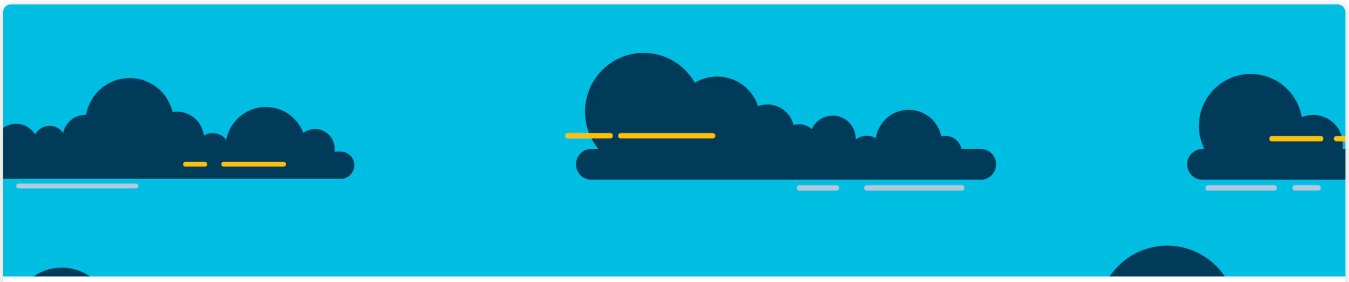
***

Continue to Part 2: 10 DynamoDB choices that will impact your costs

Srivatssan Srinivasan, Director, Enterprise Architect, Card Tech–Canada

Srivatssan “Sri” Srinivasan is currently fulfilling the role of an Enterprise Architect at Capital one. During his career, he has developed a deep understanding of business processes in the Insurance, E-Commerce and Financial domains. During free time, Sri loves to read books on psychology, personal development, leadership training and has a special interest in the field of cognitive psychology. Sri is also an advocate in developing modern microservice-based event-driven distributed applications.

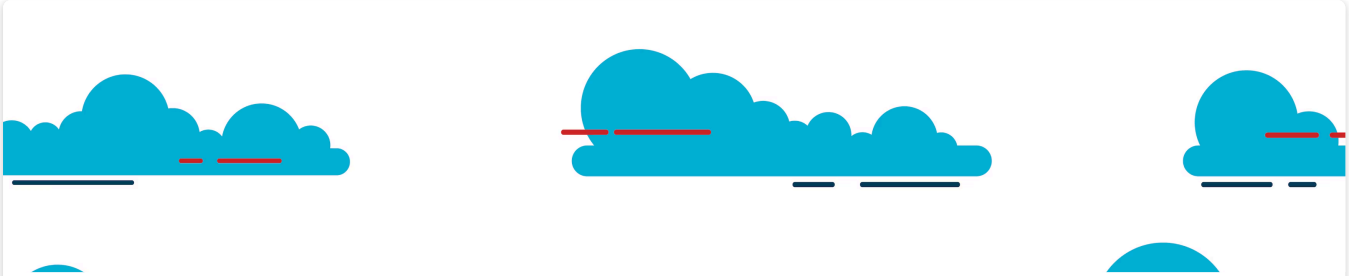
RELATED CONTENT



SOFTWARE ENGINEERING

DynamoDB: 10 tips to improve performance and lower cost

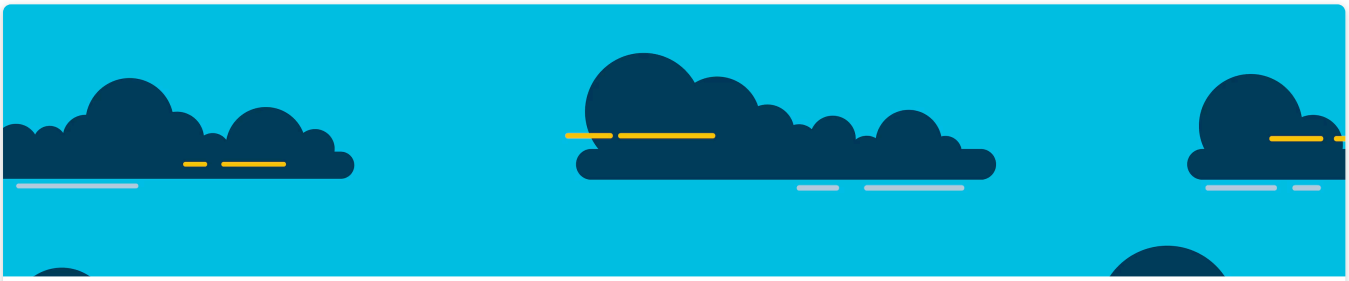
Article | July 24, 2023



SOFTWARE ENGINEERING

9 recommendations to minimize DynamoDB operational costs

Article | August 16, 2022



CLOUD

How automated deployment works with DynamoDB global tables

Article | March 31, 2023 | 4 min read

Auto

Business

Commercial

Capital One Shopping

CreditWise

Newsroom

Technology

Investors

Careers & Jobs

Diversity & Inclusion

Canada

UK

Mobile App

Meet Eno

Digital Tools

Legal

Privacy

Patriot Act Certification

Wolfsberg Questionnaire

Subpoena Policy

Additional Disclosures

Support

COVID-19

Help Center

Learn & Grow

Resources for Military

Accessibility Assistance

Tweet @AskCapitalOne

Security

Contact Us

Footnotes

Learn more about FDIC insurance coverage.