Introduction to DynamoDB

DynamoDB FAQs

What is 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.

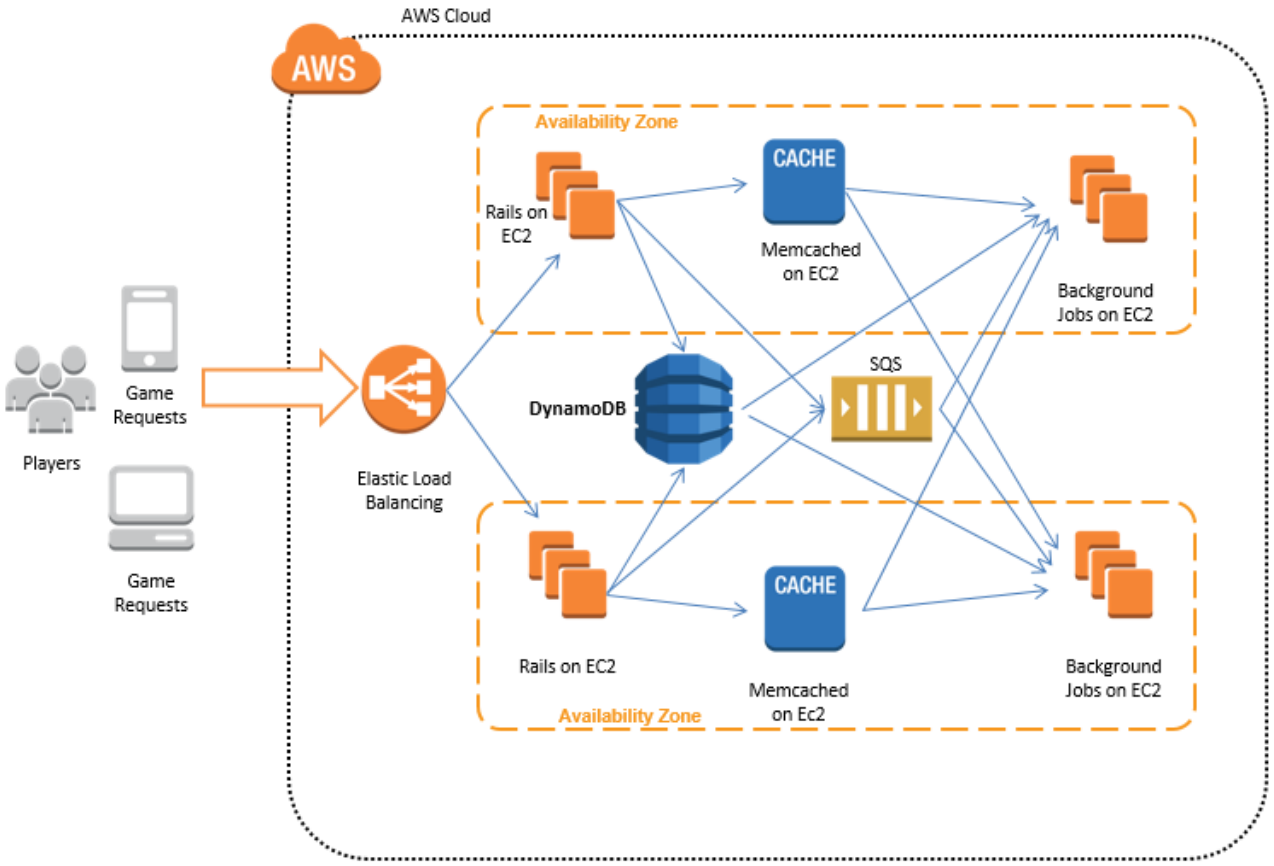

Image Source

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
- Dec 2019 — On-demand capacity mode for global tables
- Sep 2020 — Support for PartiQL (SQL-like language)
- Apr 2021 — DynamoDB Streams extended to 2 TB per table
- Dec 2022 — Flexible Throughput Capacity Mode
- Jul 2023 — Advanced filtering for DynamoDB Streams
- Aug 2023 — Automatic index optimization for global secondary indexes

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.

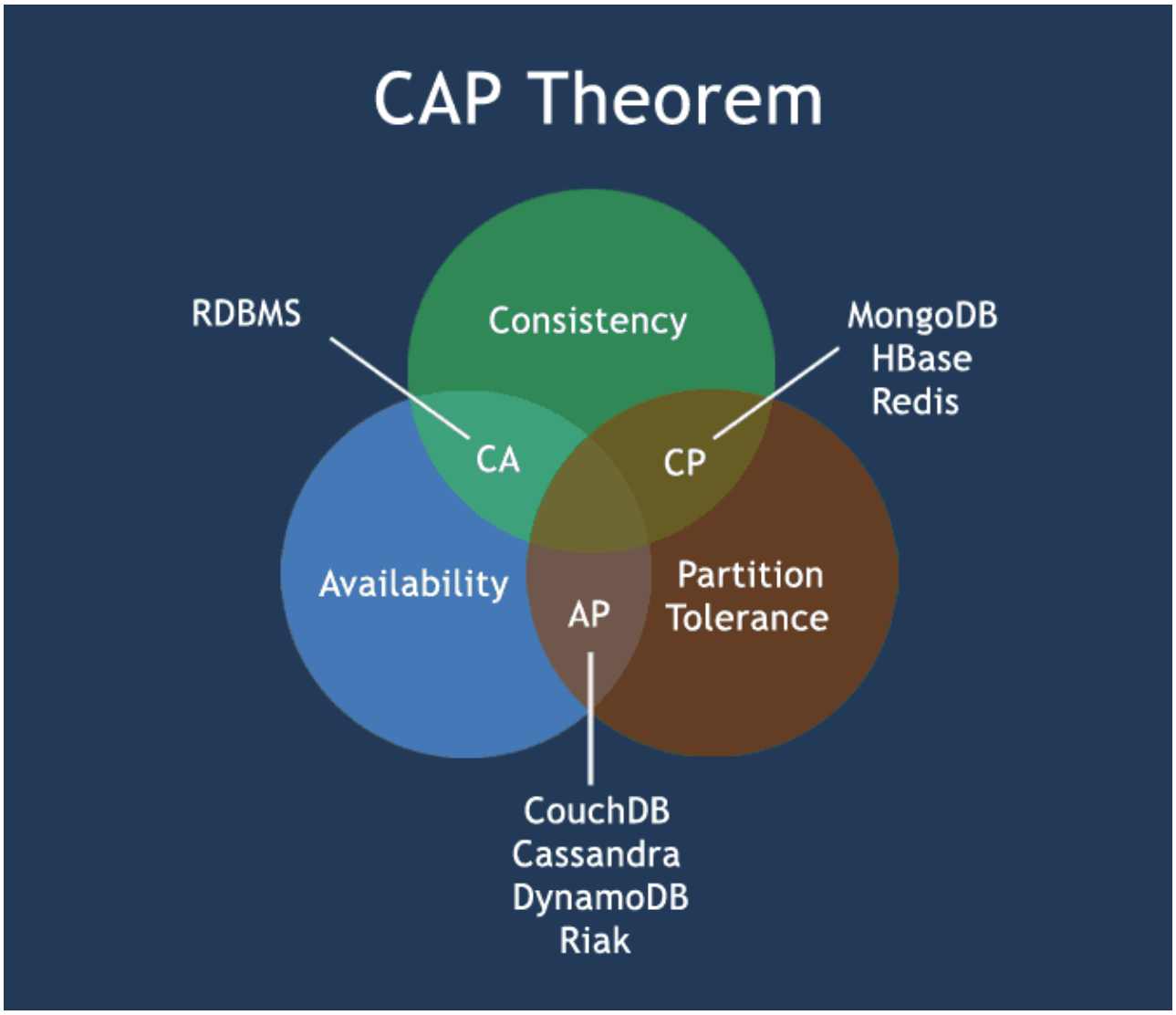

Image Source

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.

Masterclass: Data Modeling for NoSQL Databases

Looking for extensive training on about data modeling for NoSQL Databases? Our experts offer a 3-hour masterclass that assists practitioners wanting to migrate from SQL to NoSQL or advance their understanding of NoSQL data modeling. This free, self-paced class covers techniques and best practices on NoSQL data modeling that will help you steer clear of mistakes that could inconvenience any engineering team.

You can access the complete course here.

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 is available as free Open Source software which 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.