AWS Prescriptive Guidance

Modeling data with Amazon DynamoDB

AWS Prescriptive Guidance: Modeling data with Amazon DynamoDB

Table of Contents

Introduction 1
Process flow 2
RACI matrix 2
Process steps 4
Step 1. Identify the use cases and logical data model 4
Objectives 4
Process 4
Tools and resources 4
RACI 4
Outputs 5
Step 2. Create a preliminary cost estimation 5
Objective 5
Process 5
Tools and resources 5
RACI 5
Outputs 5
Step 3. Identify your data access patterns 5
Objective 6
Process 6
Tools and resources 6
RACI 6
Outputs 6
Sample 7
Step 4. Identify the technical requirements 7
Objective 7
Process 7
Tools and resources 7
RACI 7
Outputs 7
Step 5. Create the DynamoDB data model 7
Objective 7
Process 8
Tools and resources 8
RACI 9
Outputs 9
Sample 9
Step 6. Create the data queries 9
Objective 9
Process 9
Tools and resources 10
RACI 10
Outputs 10
Samples 10
Step 7. Validate the data model 10
Objective 10
Process 10
Tools and resources 11
RACI 11
Outputs 11
Step 8. Review the cost estimation 11
Objective 11
Process 11
Tools and resources 11
RACI 12

Outputs 12
Step 9. Deploy the data model 12
Objective 12
Process 12
Tools and resources 12
RACI 12
Outputs 12
Sample 12
Templates 14
Business requirements assessment template 14
Technical requirements assessment template 16
Access patterns matrix 20
Additional resources 22
Document history 24

Modeling data with Amazon DynamoDB

Process, templates, and best practices

Regis Gimenis, Senior Data Architect, AWS Professional Services

Camilo Gonzalez, Data Architect, AWS Professional Services

October 2020

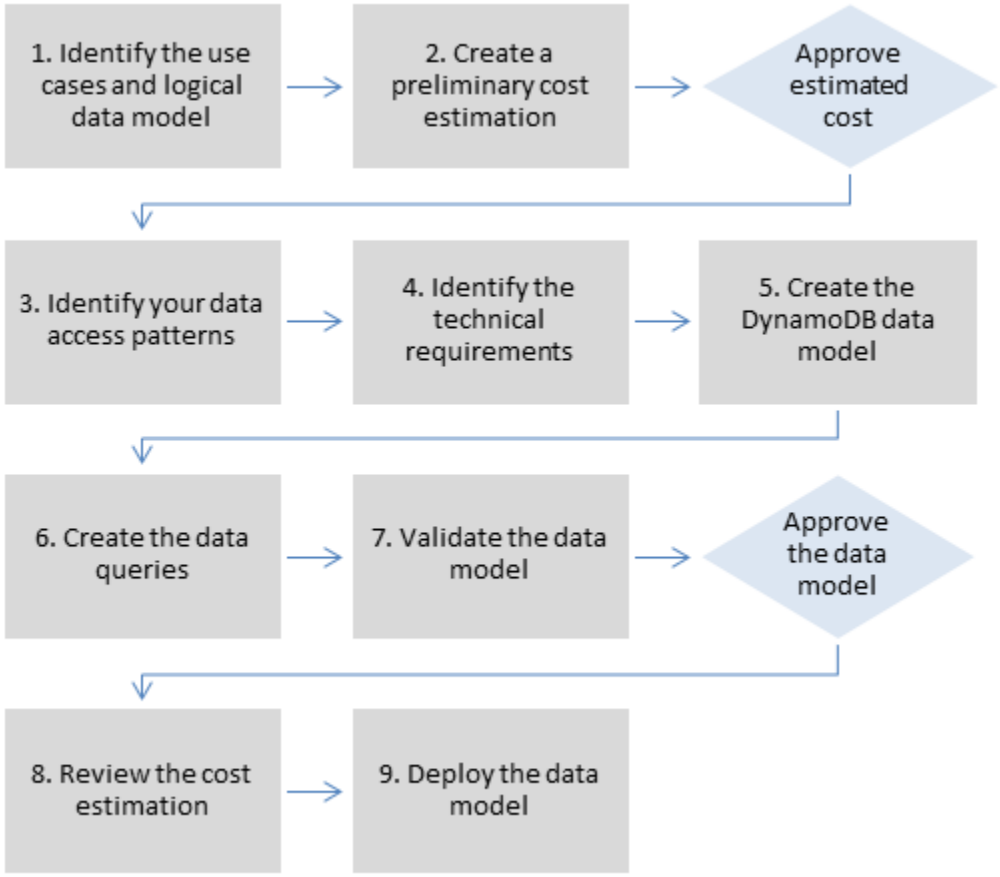
NoSQL databases provide flexible schemas for building modern applications. They are widely recognized for their ease of development, functionality, and performance at scale. Amazon DynamoDB provides fast and predictable performance with seamless scalability for NoSQL databases in the Amazon Web Services (AWS) Cloud. As a fully managed database service, DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

NoSQL schema design requires a different approach from traditional relational database management system (RDBMS) design. RDBMS data model focuses on the structure of data and its relationships with other data. NoSQL data modeling focuses on access patterns, or how the application is going to consume the data, so it stores the data in a way that supports straightforward query operations. For an RDBMS such as Microsoft SQL Server or IBM Db2, you can create a normalized data model without thinking much about access patterns, and extend it to support your patterns and queries later.

This guide presents a data modeling process for using DynamoDB that provides functional requirements, performance, and effective costs. The guide is for database engineers who are planning to use DynamoDB as the operational database for their applications that are running on AWS. AWS Professional Services has used the recommended process to help enterprise companies with DynamoDB data modeling for different use cases and workloads.

Data modeling process flow

We recommend the following process when modeling data using Amazon DynamoDB. The steps are discussed in detail later in this guide (p. 4).



RACI matrix

Some organizations use a responsibility assignment matrix (also known as a *RACI matrix*) to describe the various roles involved in one specific project or business process. This guide presents a suggested RACI matrix that could help your organization identify the right people and right responsibilities for the DynamoDB data modeling process. For each step in the process, it lists the stakeholders and their involvement:

- **R** – responsible for completing the step
- **A** – accountable for approving and signing off on the work
- **C** – consulted to provide input for a task
- **I** – informed of progress, but not directly involved in the task

Depending on the structure of your organization and project team, the roles in the following RACI matrix can be performed by the same stakeholder. In some situations, stakeholders are both responsible and accountable for specific steps. For example, database engineers can be responsible for both creating and approving the data model, because this is their domain area.

Process step	Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
1. Identify the use cases and logical data model	C	R/A	I	R		
2. Create a preliminary cost estimation	C	A	I	R		
3. Identify your data access patterns	C	A	I	R		
4. Identify the technical requirements	C	C	A	R		
5. Create the DynamoDB data model	I	I	I	R/A		
6. Create the data queries	I	I	I	R/A	R	
7. Validate the data model	A	R	I	C		
8. Review the cost estimation	C	A	I	R		
9. Deploy the DynamoDB data model	I	I	C	C		R/A

Data modeling process steps

This section details each step of the recommended data modeling process for Amazon DynamoDB.

Topics

- Step 1. Identify the use cases and logical data model (p. 4)
- Step 2. Create a preliminary cost estimation (p. 5)
- Step 3. Identify your data access patterns (p. 5)
- Step 4. Identify the technical requirements (p. 7)
- Step 5. Create the DynamoDB data model (p. 7)
- Step 6. Create the data queries (p. 9)
- Step 7. Validate the data model (p. 10)
- Step 8. Review the cost estimation (p. 11)
- Step 9. Deploy the data model (p. 12)

Step 1. Identify the use cases and logical data model

Objectives

- Gather the business needs and use cases that require a NoSQL database.
- Define the logical data model by using an entity-relationship (ER) diagram.

Process

- Business analysts interview business users to identify the use cases and the expected outcomes.
- Database engineer creates the conceptual data model.
- Database engineer creates the logical data model.
- Database engineer gathers information about item size, data volume, and expected read and write throughput.

Tools and resources

- Business requirements assessment (see template (p. 14))
- Access patterns matrix (see template (p. 20))
- Your preferred tool for creating diagrams

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
C	R/A	I	R		

Outputs

- Documented use cases and business requirements
- Logical data model (ER diagram)

Step 2. Create a preliminary cost estimation

Objective

- Develop a preliminary cost estimation for DynamoDB.

Process

- Database engineer creates the initial cost analysis using available information and the examples presented on the DynamoDB pricing page.
  - Create a cost estimate for on-demand capacity (see example).
  - Create a cost estimate for provisioned capacity (see example).
    - For the provisioned capacity model, get the estimate cost from the calculator and apply discount for reserved capacity.
  - Compare the estimated costs of the two capacity models.
- Business analyst reviews and approves or rejects the preliminary cost estimate.

Tools and resources

- AWS Simple Monthly Calculator
- AWS Pricing Calculator

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
C	A	I	R		

Outputs

- Preliminary cost estimation

Step 3. Identify your data access patterns

Access patterns or query patterns define how the users and the system access the data to satisfy business needs.

Objective

- Document the data access patterns.

Process

- Database engineer and business analyst interview end users to identify how data will be queried using the data access patterns matrix template.
  - For new applications, they review user stories about activities and objectives. They document the use cases and analyze the access patterns that the use cases require.
  - For existing applications, they analyze query logs to find out how people are currently using the system and to identify the key access patterns.
- Database engineer identifies the following properties of the access patterns:
  - Data size: Knowing how much data will be stored and requested at one time helps determine the most effective way to partition the data (see blog post).
  - Data shape: Instead of reshaping data when a query is processed (as an RDBMS system does), a NoSQL database organizes data so that its shape in the database corresponds with what will be queried. This is a key factor in increasing speed and scalability.
  - Data velocity: DynamoDB scales by increasing the number of physical partitions that are available to process queries, and by efficiently distributing data across those partitions. Knowing the peak query loads in advance might help determine how to partition data to best use I/O capacity.
- Business user prioritizes the access or query patterns.
  - Priority queries usually are the most used or most relevant queries. It is also important to identify queries that require lower response latency.

Tools and resources

- Access patterns matrix (see template (p. 20))
- Choosing the Right DynamoDB Partition Key (AWS Database blog)
- NoSQL Design for DynamoDB (DynamoDB documentation)

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
C	A	I	R		

Outputs

- Data access patterns matrix

Sample

Access Pattern	Priority	Read or Write	Description	Type (Single Item / Multiple Items / All)	Key Attribute	Filters	Result ordering
Create user profile	*High*	*Write*	*User create a new profile*	*Single Item*	*Username*	*NA*	*NA*
Update user profile	*Medium*	*Write*	*User update his/her profile*	*Single Item*	*Username*	*Username = current user*	*NA*

Step 4. Identify the technical requirements

Objective

- Gather the technical requirements for the DynamoDB database.

Process

- Business analysts interview the business user and DevOps team to gather the technical requirements by using the assessment questionnaire.

Tools and resources

- Technical requirements assessment (see sample questionnaire (p. 16))

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
C	C	A	R		

Outputs

- Technical requirements document

Step 5. Create the DynamoDB data model

Objective

- Create the DynamoDB data model.

Process

- Database engineer identifies how many tables will be required for each use case. We recommend maintaining as few tables as possible in a DynamoDB application.
- Based on the most common access patterns, identify the primary key that can be one of two types: a primary key with a simple partition key that identifies data, or a primary key with a partition key and a sort key. A sort key is a secondary key for grouping and organizing data so it can be queried within a partition efficiently. You can use sort keys to define hierarchical relationships in your data that you can query at any level of the hierarchy (see blog post).
  - Partition key design
    - Define the partition key and evaluate its distribution.
    - Identify the need for write sharding to distribute workloads evenly.
  - Sort key design
    - Identify the sort key.
    - Identify the need for a composite sort key.
    - Identify the need for version control.
- Based on the access patterns, identify the secondary indexes to satisfy the query requirements.
  - Identify the need for local secondary indexes (LSIs). These are indexes that have the same partition key as the base table, but a different sort key.
    - For tables with local secondary indexes, there is a 10 GB size limit per partition key value. A table with local secondary indexes can store any number of items, as long as the total size for any one partition key value does not exceed 10 GB.
  - Identify the need for global secondary indexes (GSIs). These are indexes that have a partition key and a sort key that can be different from those on the base table (see blog post).
  - Define the index projections. Consider projecting fewer attributes to minimize the size of items written to the index. In this step, you should determine whether you want to use the following:
    - Sparse indexes
    - Materialized aggregation queries
    - GSI overloading
    - GSI sharding
    - An eventually consistent replica using GSI
- Database engineer determines whether the data will include large items. If so, they design the solution by using compression or by storing data in Amazon Simple Storage Service (Amazon S3).
- Database engineer determines whether time series data will be needed. If so, they use the time series design pattern to model the data.
- Database engineer determines whether the ER model includes many-to-many relationships. If so, they use an adjacency list design pattern to model the data.

Tools and resources

- NoSQL Workbench for Amazon DynamoDB — Provides data modeling, data visualization, and query development/testing features to help you design your DynamoDB database
- NoSQL Design for DynamoDB (DynamoDB documentation)
- Choosing the Right DynamoDB Partition Key (AWS Database blog)
- Best Practices for Using Secondary Indexes in DynamoDB (DynamoDB documentation)
- How to design Amazon DynamoDB global secondary indexes (AWS Database blog)

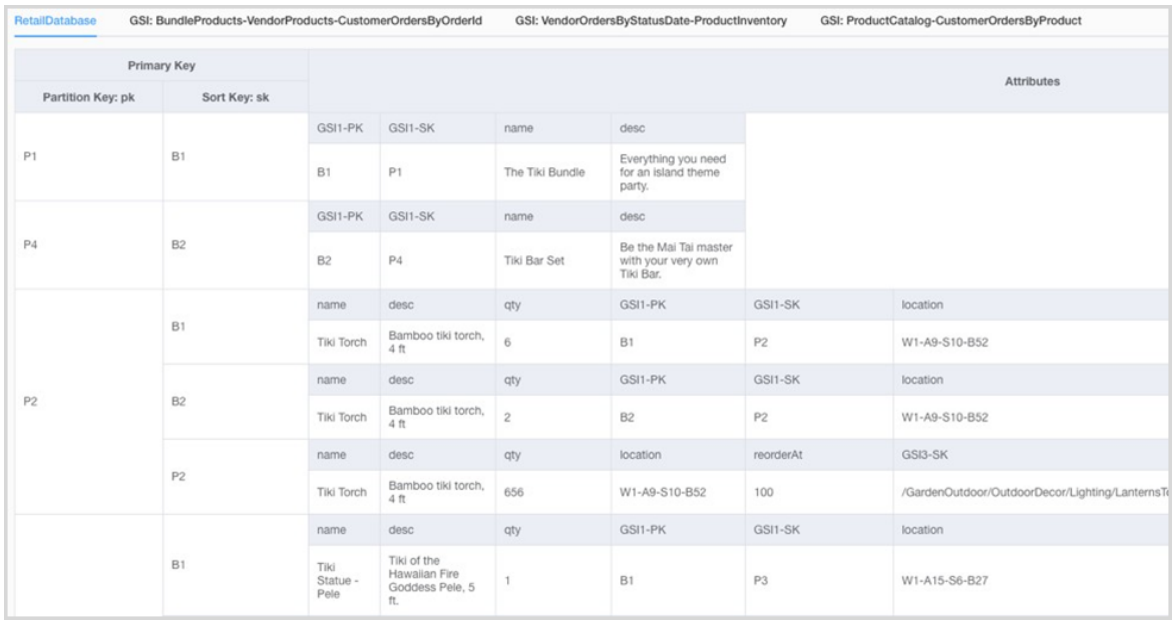
RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
I	I	I	R/A		

Outputs

- DynamoDB table schema that satisfies your access patterns and requirements

Sample



RetailDatabase | GSI: BundleProducts-VendorProducts-CustomerOrdersByOrderId | GSI: VendorOrdersByStatusDate-ProductInventory | GSI: ProductCatalog-CustomerOrdersByProduct

Primary Key		Attributes					
Partition Key: pk	Sort Key: sk						
P1	B1	GSI1-PK	GSI1-SK	name	desc		
		B1	P1	The Tiki Bundle	Everything you need for an island theme party.		
P4	B2	GSI1-PK	GSI1-SK	name	desc		
		B2	P4	Tiki Bar Set	Be the Mai Tai master with your very own Tiki Bar.		
P2	B1	name	desc	qty	GSI1-PK	GSI1-SK	location
		Tiki Torch	Bamboo tiki torch, 4 ft	6	B1	P2	W1-A9-S10-B52
	B2	name	desc	qty	GSI1-PK	GSI1-SK	location
		Tiki Torch	Bamboo tiki torch, 4 ft	2	B2	P2	W1-A9-S10-B52
	P2	name	desc	qty	location	reorderAt	GSI3-SK
		Tiki Torch	Bamboo tiki torch, 4 ft	656	W1-A9-S10-B52	100	/GardenOutdoor/OutdoorDecor/Lighting/LanternsT
	B1	name	desc	qty	GSI1-PK	GSI1-SK	location
		Tiki Statue - Pele	Tiki of the Hawaiian Fire Goddess Pele, 5 ft.	1	B1	P3	W1-A15-S6-B27

Step 6. Create the data queries

Objective

- Create the main queries to validate the data model.

Process

- Database engineer manually creates a DynamoDB table in the AWS Region or on their computer (DynamoDB Local).
- Database engineer adds sample data to the DynamoDB table.
- Database engineer builds facets using the NoSQL Workbench for Amazon DynamoDB or the AWS SDK for Java or Python to build sample queries (see blog post).

Facets are like a view of the DynamoDB table.

- Database engineer and cloud developer build sample queries by using the AWS Command Line Interface (AWS CLI) or AWS SDK for the preferred language.

Tools and resources

- An active AWS account, to gain access to the DynamoDB console
- DynamoDB Local (optional), if you want to build the database on your computer without accessing the DynamoDB web service
- NoSQL Workbench for Amazon DynamoDB (download and documentation)
- NoSQL Workbench for Amazon DynamoDB (AWS News blog)
- AWS SDK in your choice of language

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
I	I	I	R/A	R	

Outputs

- Code to query the DynamoDB table

Samples

- DynamoDB examples using the AWS SDK for Java
- Python samples
- JavaScript samples

Step 7. Validate the data model

Objective

- Ensure that the data model will satisfy your requirements.

Process

- Database engineer populates the DynamoDB table with sample data.
- Database engineer runs the code to query the DynamoDB table.
- Database engineer collects the query results.

- Database engineer collects the query execution metrics.
- Business user validates that query results satisfy business needs.
- Business analysts validate the technical requirements.

Tools and resources

- An active AWS account, to gain access to the DynamoDB console
- DynamoDB Local (optional), if you want to build the database on your computer without accessing the DynamoDB web service
- AWS SDK in your choice of language

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
A	R	I	C		

Outputs

- Approved data model

Step 8. Review the cost estimation

Objective

- Define the capacity model and estimate DynamoDB costs to refine the cost estimation from step 2 (p. 5).

Process

- Database engineer identifies the data volume estimate.
- Database engineer identifies the data transfer requirements.
- Database engineer defines the required read and write capacity units.
- Business analyst decides between on-demand and provisioned capacity models.
- Database engineer identifies the need for DynamoDB auto scaling.
- Database engineer inputs the parameters in the Simple Monthly Calculator tool.

Tools and resources

- AWS Simple Monthly Calculator
- AWS Pricing Calculator

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
C	A	I	R		

Outputs

- Capacity model
- Revised cost estimation

Step 9. Deploy the data model

Objective

- Deploy the DynamoDB table(s) to the AWS Region.

Process

- DevOps architect creates an AWS CloudFormation template or other infrastructure as code (IaC) tool for the DynamoDB table(s). AWS CloudFormation provides an automated way to provision and configure your tables and associated resources.

Tools and resources

- AWS CloudFormation

RACI

Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
I	I	C	C		R/A

Outputs

- CloudFormation template

Sample

```
mySecondDDBTable:
   Type: AWS::DynamoDB::
   Table DependsOn: "myFirstDDBTable"
   Properties:
   AttributeDefinitions:
      -  AttributeName: "ArtistId"
         AttributeType: "S"
      -  AttributeName: "Concert"
         AttributeType: "S"
      -  AttributeName: "TicketSales"
         AttributeType: "S"
   KeySchema:
      -  AttributeName: "ArtistId"
         KeyType: "HASH"
      -  AttributeName: "Concert"
         KeyType: "RANGE"
   ProvisionedThroughput:
      ReadCapacityUnits:
         Ref: "ReadCapacityUnits"
      WriteCapacityUnits:
         Ref: "WriteCapacityUnits"
   GlobalSecondaryIndexes:
      -  IndexName: "myGSI"
         KeySchema:
            - AttributeName: "TicketSales"
            KeyType: "HASH"
         Projection:
            ProjectionType: "KEYS_ONLY"
         ProvisionedThroughput:
         ReadCapacityUnits:
            Ref: "ReadCapacityUnits"
         WriteCapacityUnits:
            Ref: "WriteCapacityUnits"
   Tags:
      -  Key: mykey
         Value: myvalue
```

Templates

The templates provided in this section are based on the Modeling Game Player Data with Amazon DynamoDB on the AWS website.

> **Note**
> The tables in this section use *MM* as an abbreviation for million, and *K* as an abbreviation for thousand.

Topics

- Business requirements assessment template (p. 14)
- Technical requirements assessment template (p. 16)
- Access patterns matrix (p. 20)

Business requirements assessment template

Provide a description for the use case:

> **Description**
> Imagine that you are building an online multiplayer game. In your game, groups of 50 players join a session to play a game, which typically takes around 30 minutes to play. During the game, you have to update a specific player's record to indicate the amount of time the player has been playing, their statistics, or whether they won the game. Users want to see old games they've played, either to view the games' winners or to watch a replay of each game's action.

Provide information about your users:

User	Description	Expected number
Game player	*Online game player.*	*1 MM*
Development team	*Internal team that will use the game statistics to improve the game experience.*	*100*

Provide information about the sources of data and how data will be ingested:

Source	Description	User
Online game	*Game players will create profiles and start new games.*	*Game player*
Game app	*Game app will automatically collect statistics about the games, such as start and end time, number of players, position of each player, and map for the game.*	

Provide information about how data will be consumed:

Consumer	Description	User
Online game	*Game players will view profiles and review their game statistics.*	*Game player*
Data analytics	*The game development team will extract game statistics for data analysis and to improve the user experience. Data will be exported from the data store and imported into Amazon S3 to support analytics through a Spark application.*	*Development team*

Provide a list of entities and how they are identified:

Entity name	Description	Identifier
Game Player	*Stores information such as identification, address, demographics, interests for each user (gamer).*	*Username*
Game Instance	*Provides information about each game played, including creator, start, end, and map Yplayed.*	*Game ID*
Game User Mapping	*Represents the many-to-many relationships between users and games.*	*Game ID AND Username*

Create an ER model for the entities:

Provide high-level statistics about the entities:

Entity Name	Estimated # of records	Record size	Notes
Game Player	*1 MM*	*< 1 KB*	*The game platform has about 1 MM users.*
Game Instance	*6 MM (100,000K/day * 60 days)*	*< 1 KB*	*On average, there are 100K games every day. We need to store the last 60 days.*
Game User Mapping	*300 MM (6 MM games * 50 players)*	*< 1 KB*	*On average, each game has 50 players that we need to store information about.*

Technical requirements assessment template

Provide information about data ingestion types:

Data ingestion type	Y/N	Description	Frequency
Application access	*Y*		
API gateway	*Y*		
Data streaming	*N*		
Batch process	*N*		
ETL	*N*		
Data import	*N*		
Time series	*N*		

Provide information about data consumption types:

Data consumption type	Y/N	Description	Frequency
Application access			
API gateway			
Data export			
Data analytics			
Data aggregation			
Reporting			
Search			
Data streaming			
ETL			

Provide data volume estimates:

Entity name	Estimated # of records	Record size	Data volume
Game Player	*1 MM*	*< 1 KB*	*~ 1 GB (1 MM * 1 KB)*
Game Instance	*6 MM (100K/day * 60 days)*	*< 1 KB*	*~ 6 GB (6 MM * 1 KB)*
Game User Mapping	*300 MM (6 MM games * 50 players)*	*< 1 KB*	*~ 300 GB (300 MM * 1 KB)*

Note
The period for data retention is 60 days. After 60 days, data must be stored in Amazon S3 for analytics, by using DynamoDB Time to Live (TTL) to automatically move data out of DynamoDB to Amazon S3.

Answer these questions about time patterns:

- What time frame is the application available to the user (for example, 24/7 or 9 AM to 5 PM on weekdays)?
- Is there a peak in usage during the day? How many hours? What is the percentage of application usage?

Specify write throughput requirements:

Entity name	Writes/day	Hours/day	Writes/second
Game Player	*10,000 updates*	*18*	*< 1*
Game Instance	*300,000*	*18*	*< 5*
Game User Mapping	*1,800,000,000*	*18*	*~ 27.777*

Notes
Game Player write operations: 1 percent of users update their profiles every day, so we expect 10,000 updates for 1,000,000 users.
Game Instance write operations: 100,000 games/day. For each game we have at least 3 write operations—at creation, start, and end—so the total is 300,000 write operations.
Game User Mapping write operations: 100,000 games/day for each game with 50 players. The average game duration is 30 minutes, and the gamer position is updated every 5 seconds. We estimate an average of 360 updates per gamer, so the total is 100,000 * 50 * 360 = 1,800,000,000 write operations.

Specify read throughput requirements:

Entity name	Reads / day	Hours / day	Reads/sec
Game Player	*200,000*	*18*	*~ 3*
Game Instance	*5,000,000*	*18*	*~ 77*
Game User Mapping	*1,800,000,000*	*18*	*~ 27.777*

Notes
Game Player read operations: 20 percent of users start games, so 1 MM * 0.2 = 200,000.
Game Instance read operations: 100,000 games/day. For each game we have at least 1 read operation per player, and 50 players per game, so the total is 5,000,000 read operations.
Game User Mapping read operations: 100,000 games/day for 50 players. The average game duration is 30 minutes, and the gamer position is updated every 5 seconds. We estimate an average of 360 updates per gamer, and each update requires a read operation, so the total is 100,000 * 50 * 360 = 1,800,000,000 read operations.

Specify data access latency requirements:

Operation	99 percentiles	Maximum latency
Read	*30 ms*	*100 ms*
Write	*10 ms*	*50 ms*

Specify data availability requirements:

Requirement	Y/N	Metric	Notes
High availability	*Y*	*99.9%*	
RTO	*Y*	*1 hour*	*Recovery time objective*
RPO	*Y*	*1 hour*	*Recovery point objective*
Disaster recovery	*N*		
In-Region data replication	*N*		
Cross-Region data replication	*N*	*3 sec latency*	*Which AWS Regions?*

Specify security requirements:

Requirement	Y/N	Notes
Sensitive data store	*N*	*Protected health information (PHI), payment card industry (PCI) information, personally identifiable information (PII)?*
Encryption at rest	*Y*	
Encryption in transit	*Y*	
Client-side encryption	*N*	
Any proprietary or third-vendor encryption library	*N*	
Data access logging	*N*	
Data access auditing	*N*	

Access patterns matrix

Collect and document information about the access patterns for the use case by using the following fields:

Field	Description
Access pattern	Provide a name for the access pattern.
Description	Provide a more detailed description of the access pattern.
Priority	Define a priority for the access pattern (high/medium/low). This defines the most relevant access patterns for the application.
Read or write	Is it a read access or write access pattern?
Type	Does the pattern access a single item, multiple items, or all items?
Key attribute	What is the key attribute used to access data?
Filter	Does the access pattern require any filter?
Sort	Does the result require any sorting?

Template:

Access pattern	Priority	Read or write	Description	Type (single item / multiple items / all)	Key attribute	Filters	Result ordering
Create user profile	*High*	*Write*	*User creates a new profile.*	*Single item*	*Username*	*N/A*	*N/A*
Update user profile	*Medium*	*Write*	*User updates their profile.*	*Single item*	*Username*	*Username = current user*	*N/A*
Get user profile	*High*	*Read*	*User reviews their profile.*	*Single item*	*Username*	*Username = current user*	*N/A*
Create a game	*High*	*Write*	*User creates a new game.*	*Single item*	*GameID*	*N/A*	*N/A*
Find open games	*High*	*Read*	*User searches for open games. Search results are sorted by start timestamp in descending order.*	*Multiple items*		*GameStatus = open*	*Start timestamp descendent*
Find open games by map	*Medium*	*Read*	*User searches for open games by using a specific map sorted by start timestamp in descending order.*	*Multiple items*		*GameStatus = open & Map = XYZ*	*Start timestamp descendent*
View game	*High*	*Read*	*User reviews the details of a game.*	*Single item*	*GameID*	*N/A*	*N/A*
View users in a game	*Medium*	*Read*	*User gets a list of all the users in a game.*	*Multiple items*		*GameID = XYZ*	*N/A*
Join user to a game	*High*	*Write*	*User joins an open game.*	*Single item*	*GameID & Username*	*GameStatus = open*	*N/A*
Start a game	*High*	*Write*	*User starts a new game.*	*Single item*	*GameID*	*N/A*	*N/A*
Update game for user	*Medium*	*Write*	*Update user position in the game.*	*Single item*	*GameID & Username*	*N/A*	*N/A*
Update game	*Medium*	*Write*	*Game ends; update statistics.*	*Single item*	*GameID*	*N/A*	*N/A*
Find all past games for a user	*Low*	*Read*	*List all games that a user played ordered by the start timestamp of the game.*	*Multiple items*	*Username & GameID*	*Username = current user*	*Start timestamp*
Export data for data analytics	*Low*	*Read*	*Development team will run a batch job to export data to Amazon S3.*	*All*	*N/A*	*N/A*	*N/A*

Additional resources

More information about DynamoDB

- DynamoDB pricing
- DynamoDB documentation
- NoSQL design for DynamoDB
- Write sharding
- Local secondary indexes (LSIs)
- Global secondary indexes (GSIs)
- Overloading GSIs
- GSI sharding
- Using GSIs to create an eventually consistent replica
- Sparse indexes
- Materialized aggregation queries
- Time series design pattern
- Adjacency list design pattern
- On-demand and provisioned capacity models
- DynamoDB auto scaling
- DynamoDB Time to Live (TTL)
- Modeling game player data with DynamoDB (lab)

AWS services

- AWS CloudFormation
- Amazon S3

Tools

- AWS Simple Monthly Calendar
- AWS Pricing Calendar
- NoSQL Workbench for DynamoDB
- DynamoDB Local
- DynamoDB and AWS SDKs

Best practices

- Best practices for designing and architecting with DynamoDB (DynamoDB documentation)
- Best practices for using secondary indexes (DynamoDB documentation)
- Best practices for storing large items and attributes (DynamoDB documentation)
- Choosing the right DynamoDB partition key (AWS Database blog)
- How to design Amazon DynamoDB global secondary indexes (AWS Database blog)
- What are facets in NoSQL Workbench for Amazon DynamoDB (*Medium* website)

AWS general resources

- AWS Prescriptive Guidance
- AWS documentation
- AWS general reference
- AWS glossary

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
Initial publication (p. 24)	—	October 26, 2020