Modeling data with Amazon DynamoDB

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Modeling data with Amazon DynamoDB

Table of Contents

Modeling data with Amazon DynamoDB

Process, templates, and best practices

Amazon Web Services (contributors)

December 2023 (document history)

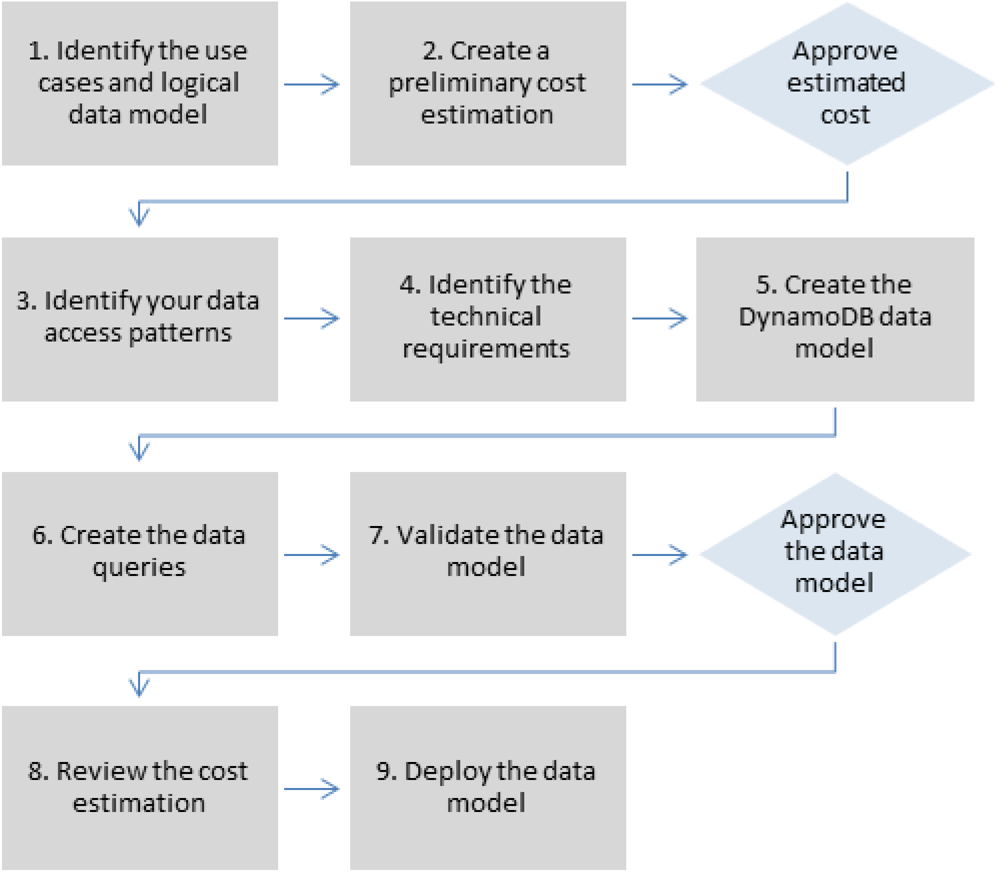
NoSQL databases provide flexible schemas for building modern applications. They are widely recognized for their ease of development, functionality, and performance at scale. Amazon DynamoDB provides fast and predictable performance with seamless scalability for NoSQL databases in the Amazon Web Services (AWS) Cloud. As a fully managed database service, DynamoDB helps you offload the administrative burdens of operating and scaling a distributed database. 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. You can extend the data model 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.


1. Identify the use cases and logical data model
2. Create a preliminary cost estimation
Approve estimated cost
3. Identify your data access patterns
4. Identify the technical requirements
5. Create the DynamoDB data model
6. Create the data queries
7. Validate the data model
Approve the data model
8. Review the cost estimation
9. Deploy the data model

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	I	I	I	R/A		

Process step	Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
DynamoDB data model						
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

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)

- Access patterns matrix (see template)
- 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.
  - Create an estimation for all the environments (Dev, Prod, QA).
- Business analyst reviews and approves or rejects the preliminary cost estimate.

Tools and resources

- 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 the 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)
- 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

Example

Access pattern	Priority	Read or write	Description	Type (single item, multiple items, or 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

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 example questionnaire)

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 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 LSIs, there is a 10 GB size limit per partition key value. A table with LSIs 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 and 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

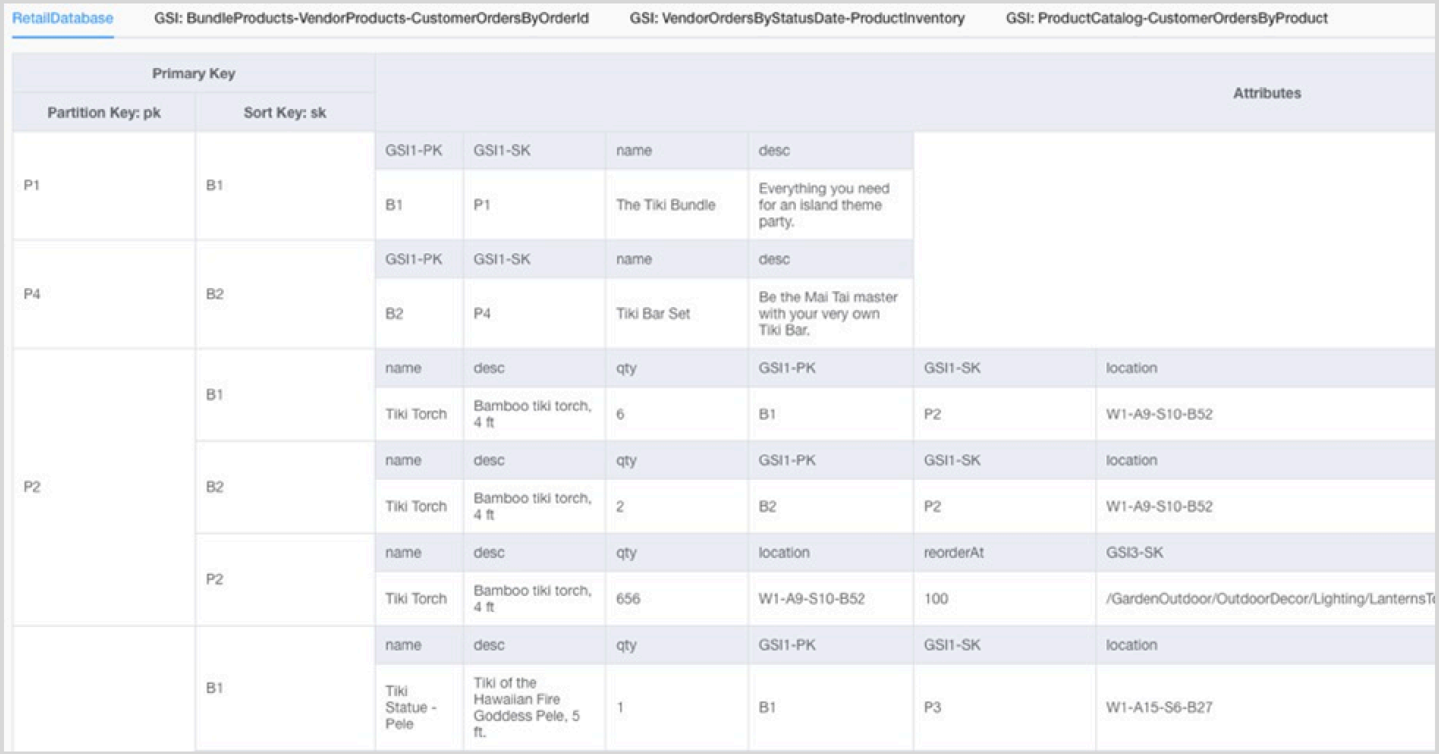
Business user	Business analyst	Solutions architect	Database engineer	Application developer	DevOps engineer
I	I	I	R/A		

Outputs

- DynamoDB table schema that satisfies your access patterns and requirements

Example

The following screenshot shows NoSQL Workbench.



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)
- AWS SDK in your choice of language (JavaScript, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++, and SAP ABAP)

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

Examples

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

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 performance 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

Objectives

- Define the capacity model and estimate DynamoDB costs to refine the cost estimation from step 2.
- Get the final financial approval from the business analyst and stakeholders.

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.
- Database engineer presents the final price estimation to business stakeholders.
- Business analyst and stakeholders approve or reject the solution.

Tools and resources

- 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 (or tables) to the AWS Region.

Process

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

Tools and resources

- CloudFormation

RACI

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

Outputs

- AWS CloudFormation template

Example

```
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
- Technical-requirements assessment template
- Access-patterns template

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 earlier 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 automatic ally collect statistics about the games, such as start and end time, number of players, position of each player, and map for the game.*	

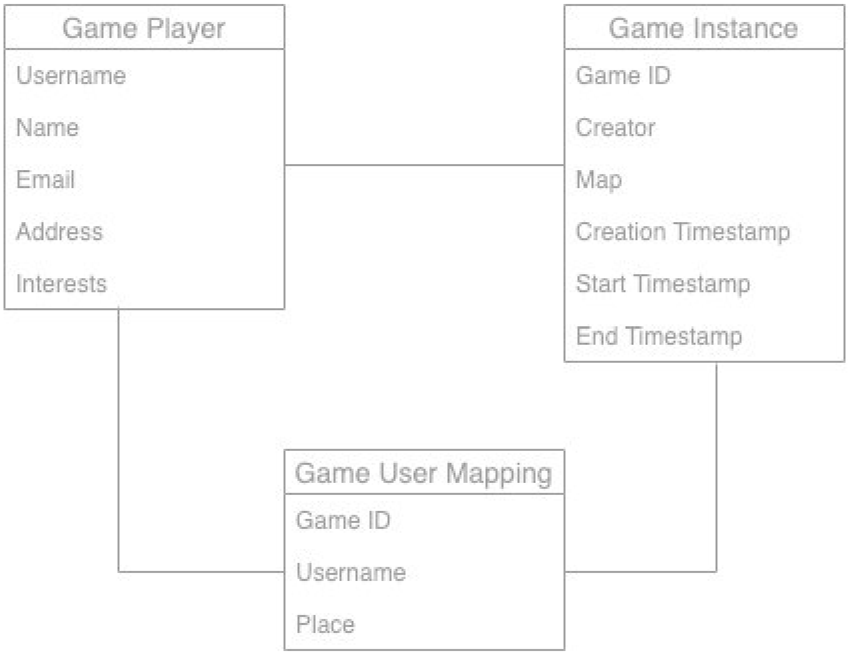
Provide information about how data will be consumed:

Consumer	Description	User
Online game	*Game players will view profiles and review their game statistic s.*	*Game player*
Data analytics	*The game development team will extract game statistic s 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 template

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, or 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?
Filter	Does the access pattern require any filter?
Sort	Does the result require any sorting?

Template

Access pattern	Descripti on	Priority	Read or write	Type (single item, multiple items, or all)	Key attribute	Filters	Result ordering
Create user profile	*User creates a new profile.*	*High*	*Write*	*Single item*	*Username*	*N/A*	*N/A*

Update user profile	*User updates their profile.*	*Medium*	*Write*	*Single item*	*Username*	*Username = current user*	*N/A*
Get user profile	*User reviews their profile.*	*High*	*Read*	*Single item*	*Username*	*Username = current user*	*N/A*
Create a game	*User creates a new game.*	*High*	*Write*	*Single item*	*GameID*	*N/A*	*N/A*
Find open games	*User searches for open games. Search results are sorted by start timestamp in descending order.*	*High*	*Read*	*Multiple items*		*GameStatus = open*	*Start timestamp descendent*

Find open games by map	User searches for open games by using a specific map sorted by start timestamp in descending order.	Medium	Read	Multiple items		GameStatus = open and Map = XYZ	Start timestamp descendent
View game	User reviews the details of a game.	High	Read	Single item	GameID	N/A	N/A
View users in a game	User gets a list of all the users in a game.	Medium	Read	Multiple items		GameID = XYZ	N/A
Join user to a game	User joins an open game.	High	Write	Single item	GameID and Username	GameStatus = open	N/A
Start a game	User starts a new game.	High	Write	Single item	GameID	N/A	N/A

Update game for user	*Update user position in the game.*	*Medium*	*Write*	*Single item*	*GameID and Username*	*N/A*	*N/A*
Update game	*Game ends; update statistics.*	*Medium*	*Write*	*Single item*	*GameID*	*N/A*	*N/A*
Find all past games for a user	*List all games that a user played ordered by the start timestamp of the game.*	*Low*	*Read*	*Multiple items*	*Username and GameID*	*Username = current user*	*Start timestamp*
Export data for data analytics	*Development team will run a batch job to export data to Amazon S3.*	*Low*	*Read*	*All*	*N/A*	*N/A*	*N/A*

Best practices

Consider using the following DynamoDB design best practices:

- Partition key design – Use a high-cardinality partition key to distribute load evenly.
- Adjacency list design pattern – Use this design pattern for managing one-to-many and many-to-many relationships.
- Sparse index – Use sparse index for your global secondary indexes (GSIs). When you create a GSI, you specify a partition key and optionally a sort key. Only items in the base table that contain a corresponding GSI partition key appear in the sparse index. This helps to keep GSIs smaller.
- Index overloading – Use the same GSI for indexing various types of items.
- GSI write sharding – Shard wisely to distribute data across the partitions for efficient and faster queries.
- Large items – Store only metadata inside the table, save the blob in Amazon S3, and keep the reference in DynamoDB. Break large items into multiple items, and efficiently index by using sort keys.

For more design best practices, see the Amazon DynamoDB documentation.

Example of hierarchical data modeling

The following sections use an example automotive company to show how you can use the data modeling process steps to design a multi-level component-management system in DynamoDB.

Topics

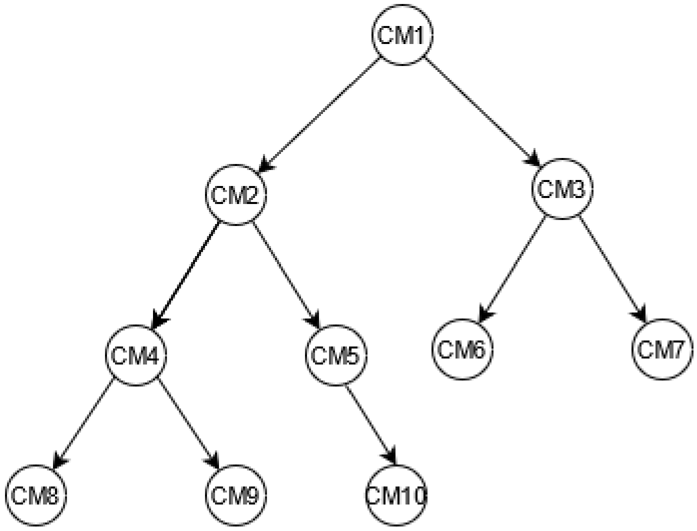
Step 1: Identify the use cases and logical data model

An automotive company wants to build a transactional component management system to store and search for all of the available car parts and to build relationships between different components and parts. For example, a car contains multiple batteries, each battery contains multiple high-level modules, each module contains multiple cells, and each cell contains multiple low-level components.

Generally, for building a hierarchical relationship model, a graph database such as Amazon Neptune is a better choice. In some cases, however, Amazon DynamoDB is a better alternative for hierarchical data modeling because of its flexibility, security, performance, and scale.

For example, you might build a system where 80–90 percent of the queries are transactional, where DynamoDB fits well. In this example, the other 10–20 percent of queries are relational, where a graph database such as Neptune fits better. In this case, including an additional database in the architecture to fulfill only 10–20 percent of the queries could increase cost. It also adds the operational burden of maintaining multiple systems and synchronizing the data. Instead, you can model that 10–20 percent relational queries in DynamoDB.

Diagramming an example tree for car components can help you map the relationship between them. The following diagram shows a dependency graph with four levels. CM1 is the top-level component for the example car itself. It has two subcomponents for two example batteries, CM2 and CM3. Each battery has two subcomponents, which are the modules. CM2 has modules CM4 and CM5, and CM3 has modules CM6 and CM7. Each module has several subcomponents, which are the cells. The CM4 module has two cells, CM8 and CM9. CM5 has one cell, CM10. CM6 and CM7 don't have any associated cells yet.



This guide will use this tree and its component identifiers as a reference. A top component will be referred to as a *parent*, and a subcomponent will be referred to as a *child***.** For example, top component CM1 is the parent of CM2 and CM3. CM2 is the parent of CM4 and CM5. This graphs the parent-child relationships.

From the tree, you can see the complete dependency graph of a component. For example, CM8 is dependent on CM4, which is dependent on CM2, which is dependent on CM1. The tree defines the complete dependency graph as the path. A path describes two things:

- The dependency graph
- The position in the tree

Filling the templates for business requirements:

Provide information about your users:

User	Description
Employee	Internal employee of the automotive company that needs information of cars and its components

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

Source	Description	User
Management system	System that will store all the data related to available car parts and their relationships with other components and parts.	Employee

Provide information about how data will be consumed:

Consumer	Description	User
Management system	Retrieve all the immediate child components for a parent component ID.	Employee
Management system	Retrieve a recursive list of all child components for a component ID.	Employee
Management system	See the ancestors of a component.	Employee

Step 2: Create a preliminary cost estimation

It's important to calculate an estimation of the cost for all environments of your application so you can check if the solution is financially viable. A best practice is to make a high-level estimation and get approval from the business analyst before proceeding with the development and deployment.

- 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 estimated cost from the calculator, and apply the discount for reserved capacity.
  - Compare the estimated costs of the two capacity models.
  - Create an estimation for all the environments (Dev, Prod, QA).
- Business analyst reviews and approves or rejects the preliminary cost estimate.

Using these reference values, you can create an estimated price to submit for approval. To create the budget, you can use the DynamoDB pricing page and AWS Pricing Calculator.

Step 3: Identify your data-access patterns

This example use case has the following access patterns for managing relationships between different car components.

Access pattern	Priority	Read or write	Description	Type	Filters	Result ordering
Immediate child	High	Read	Retrieve all the immediate child components for a parent	Multiple	`Component ID`	N/A

			component ID.			
All child components	High	Read	Retrieve a recursive list of all child components for a component ID.	Multiple	`Component ID`	N/A
Ancestors	High	Read	Retrieve the ancestors of a component.	Multiple	`Component ID`	N/A

Step 4: Identify the technical requirements

This example doesn't have any specific technical requirements, which are outside the scope of this example. In real cases, it's a best practice to complete this step and to validate that all technical requirements are fulfilled before proceeding with development and deployment. You can use the example questionnaire to complete this step in your business case. Additionally, we recommend validating the DynamoDB service quotas to make sure that there are no hard limits in your designed solution.

Step 5: Create a DynamoDB data model

Define the partition keys for your base table and global secondary indexes (GSIs):

- Following the key design best practices, use `ComponentId` as the partition key for the base table in this example. Because it's unique, `ComponentId` can offer granularity. DynamoDB uses the hash value of your partition key to determine the partition where the data is stored physically. The unique component ID generates a different hash value, which can facilitate distribution of data inside the table. You can query the base table by using a `ComponentId` partition key.

- To find immediate children of a component, create a GSI where `ParentId` is the partition key, and `ComponentId` is the sort key. You can query this GSI by using `ParentId` as the partition key.
- To find all recursive children of a component, create a GSI where `GraphId` is the partition key, and `Path` is the sort key. You can query this GSI by using `GraphId` as the partition key and the `BEGINS_WITH(Path, "$path")` operator on the sort key.

	Partition key	Sort Key	Mapping attributes
Base table	`ComponentId`		`ParentId`, `GraphId`, `Path`
GSI1	`ParentId`	`ComponentId`	
GSI2	`GraphId`	`Path`	`ComponentId`

Storing components in the table

The next step is to store each component in the DynamoDB base table. After you insert all the components from the example tree, you get the following base table.

ComponentId	ParentId	GraphId	Path
CM1		CM1#1	CM1
CM2	CM1	CM1#1	CM1|CM2
CM3	CM1	CM1#1	CM1|CM3
CM4	CM2	CM1#1	CM1|CM2|CM4

CM5	CM2	CM1#1	CM1|CM2|CM5
CM6	CM3	CM1#1	CM1|CM3|CM6
CM7	CM3	CM1#1	CM1|CM3|CM7
CM8	CM4	CM1#1	CM1|CM2|CM4|CM8
CM9	CM4	CM1#1	CM1|CM2|CM4|CM9
CM10	CM5	CM1#1	CM1|CM2|CM5|CM10

The GSI1 index

To check all immediate children of a component, you create an index that uses `ParentId` as a partition key and `ComponentId` as a sort key. The following pivot table represents the GSI1 index. You can use this index to retrieve all immediate child components by using a parent component ID. For example, you can find out how many batteries are available in a car (CM1) or which cells are available in a module (CM4).

ParentId	ComponentId
CM1	CM2
	CM3
CM2	CM4

	CM5
CM3	CM6
	CM7
CM4	CM8
	CM9
CM5	CM10

The GSI2 index

The following pivot table represents the GSI2 index. It's configured using `GraphId` as a partition key and `Path` as a sort key. Using `GraphId` and the `begins_with` operation on the sort key (`Path`), you can find the full lineage of a component in a tree.

GraphId	Path	ComponentId
CM1#1	CM1	CM1
	CM1|CM2	CM2
	CM1|CM3	CM3
	CM1|CM2|CM4	CM4
	CM1|CM2|CM5	CM5
	CM1|CM2|CM4|CM8	CM8
	CM1|CM2|CM4|CM9	CM9
	CM1|CM2|CM5|CM10	CM10
	CM1|CM3|CM6	CM6
	CM1|CM3|CM7	CM7

Step 6: Create data queries

After you define your access patterns and design your data model, you can query hierarchical data in the DynamoDB database. As a best practice to save on costs and help ensure performance, the following examples use only the query operation without `Scan`.

- **Find ancestors of a component.**

  To find the ancestors (parent, grandparent, great-grandparent, and so on) of the CM8 component, query the base table using `ComponentId = "CM8"`. The query will return the following record.

  To reduce the size of the result data, you can use a projection expression to return only the `Path` attribute.

ComponentId	ParentId	GraphId	Path
CM8	CM4	CM1#1	CM1|CM2|CM4|CM8

Path
CM1|CM2|CM4|CM8

  Now, split the path using the pipe ("|"), and take the first N-1 components to get ancestors.

  Query result: The ancestors of CM8 are CM1, CM2, CM4.

- **Find immediate children of a component.**

  To get all immediate child, or one-level downstream, components for the CM2 component, query GSI1 using `ParentId = "CM2"`. The query will return the following record.

ParentId	ComponentId
CM2	CM4
	CM5

- **Find all downstream child components using a top-level component.**

  To get all child, or downstream, components for top-level component CM1, query GSI2 using `GraphId = "CM1#1"` and `begins_with("Path", "CM1|")`, and use a projection expression with `ComponentId`. It will return all the components related to that tree.

  This example has a single tree, with CM1 as the top component. In reality, you could have millions of top-level components in the same table.

GraphId	ComponentId
CM1#1	CM2
	CM3
	CM4
	CM5
	CM8
	CM9
	CM10
	CM6
	CM7

- **Find all downstream child components using a middle-level component.**

  To get all child, or downstream, components recursively for component CM2, you have two options. You can query recursively level by level, or you can query the GSI2 index.

  - Query GSI1, level by level, recursively, until reaching the last level of child components.

    1. Query GSI1 using `ParentId = "CM2"`. It will return the following record.

ParentId	ComponentId
CM2	CM4

	CM5

2. Again, query GSI1 using `ParentId = "CM4"`. It will return the following record.

ParentId	ComponentId
CM4	CM8
	CM9

3. Again, query GSI1 using `ParentId = "CM5"`. It will return the following record.

   Continue the loop: Query for each `ComponentId` until you reach the last level. When a query using `ParentId = "<ComponentId>"` doesn't return any results, the previous result was from the last level of the tree.

ParentId	ComponentId
CM5	CM10

4. Merge all results.

   result=[CM4, CM5] + [CM8, CM9] + [CM10]

   =[CM4, CM5, CM8, CM9, CM10]

- Query GSI2, which stores a hierarchical tree for a top-level component (a car, or CM1).

  1. First, find the top-level component or top ancestor and `Path` of CM2. For that, query the base table by using `ComponentId = "CM2"` to find the path of that component in the hierarchical tree. Select the `GraphId` and `Path` attributes. The query will return the following record.

GraphId	Path
CM1#1	CM1|CM2

  2. Query GSI2 by using `GraphId = "CM1#1" AND BEGINS_WITH("Path", "CM1|CM2|")`. The query will return the following results.

GraphId	Path	ComponentId
CM1#1	CM1|CM2|CM4	CM4
	CM1|CM2|CM5	CM5
	CM1|CM2|CM4|CM8	CM8
	CM1|CM2|CM4|CM9	CM9
	CM1|CM2|CM5|CM10	CM10

3. Select the `ComponentId` attribute to return all the child components for CM2.

Step 7: Validate the data model

In this step, the business user validates the query results and checks whether they satisfy business needs. You can use the following table to check the access patterns against the requirements of the user.

Question	Base table / GSI	Query
As a user, I want to retrieve all the immediate child components for a parent component ID.	GSI1	`ParentId = "<ComponentId>"` (Find immediate children of a component.)
As a user, I want to retrieve a recursive list of all child components for a component ID.	GSI1 or GSI2	GSI1: `ParentId = "<ComponentId>"` or GSI2: `GraphId = "<TopLevelComponentId>#N" AND BEGINS_WITH("Path", "<PATH_OF_Component>")`

		(Find all down-level child components using a top- level component. Find all down-level child components using a middle-level component.)
As a user, I want to see the ancestors of a component.	Base table	`ComponentId = "<ComponentId>"` , then select the Path attribute. (Find ancestors of a component.)

You can also implement a script (test) in any programming language to query DynamoDB directly and compare the results with the expected results.

Step 8: Review the cost estimation

Review and refine the cost estimation again. Additionally, it's a good practice to validate it with business stakeholders and get approval to move to the next step.

Objectives

- Define the capacity model, and estimate DynamoDB costs to refine the cost estimation from step 2.
- Get the final financial approval from the business analyst and stakeholders.

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 AWS Pricing Calculator.

- Database engineer presents the final price estimation to business stakeholders.
- Business analyst and stakeholders approve or reject the solution.

Step 9: Deploy the data model

For this specific example, the deployment of the model was done using NoSQL Workbench, an application for modern database development and operation. Using this tool, you have the option of creating a data model, uploading data, and deploying it directly to your AWS account. If you want to implement this example, you can use the following AWS CloudFormation template, which was generated by NoSQL Workbench.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Components:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      KeySchema:
        - AttributeName: ComponentId
          KeyType: HASH
      AttributeDefinitions:
        - AttributeName: ComponentId
          AttributeType: S
        - AttributeName: ParentId
          AttributeType: S
        - AttributeName: GraphId
          AttributeType: S
        - AttributeName: Path
          AttributeType: S
      GlobalSecondaryIndexes:
        - IndexName: GS1
          KeySchema:
            - AttributeName: ParentId
              KeyType: HASH
            - AttributeName: ComponentId
              KeyType: RANGE
          Projection:
            ProjectionType: KEYS_ONLY
        - IndexName: GSI2
          KeySchema:
            - AttributeName: GraphId
              KeyType: HASH
            - AttributeName: Path
              KeyType: RANGE
          Projection:
            ProjectionType: INCLUDE
            NonKeyAttributes:
              - ComponentId
      BillingMode: PAY_PER_REQUEST
      TableName: Components
```

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 Pricing Calculator
- 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 DynamoDBglobal secondary indexes (AWS Database blog)
- What are facets in NoSQL Workbench for Amazon DynamoDB (*Medium* website)

AWS general resources

- AWS Prescriptive Guidance website
- AWS documentation
- AWS general reference

Contributors

Contributors to this guide include:

- Camilo Gonzalez, Senior Data Architect, AWS
- Moinul Al-Mamun, Senior Big Data Architect, AWS
- Santiago Segura, Professional Services Consultant, AWS
- Satheish Kumar Chandraprakasam, Cloud Application Architect, AWS

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.

Change	Description	Date
Added a *Best practices* section and an example for hierarchical data modeling.	We added a summary of DynamoDB best practices and a step-by-step example of designing and validating a hierarchical model.	December 5, 2023
Initial publication	—	October 26, 2020

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there’s no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system’s resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model’s predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries.

Detective guardrails detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub CSPM, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization’s applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application’s features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic `PUT` and `DELETE` requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice’s data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model’s responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that’s used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it’s called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata.

The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization’s capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn’t provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.