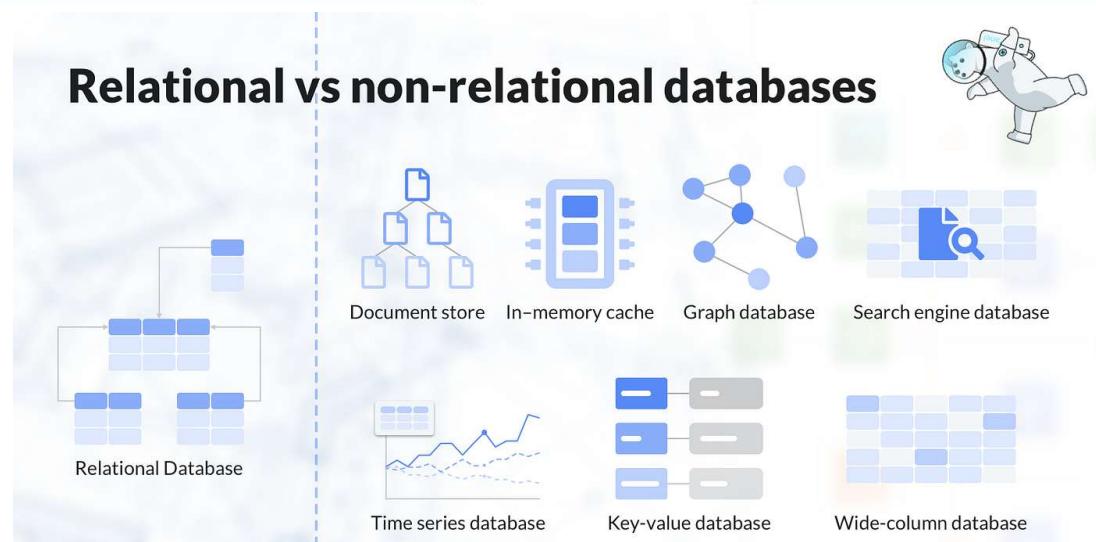
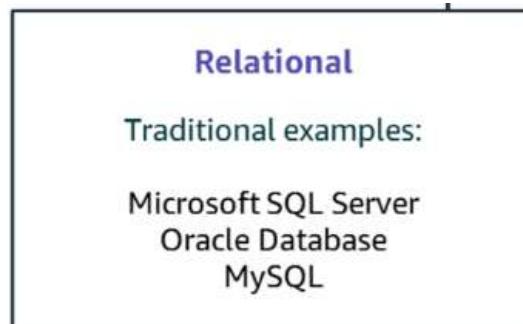


AWS Database Services

Poornima S, Asso Sr, SCOPE

Database Considerations and Types

- Scalability
- Storage requirements
- Size and Type
- Durability



Database to Cloud

- On-premises database -> cloud db
- Advantage of moving DB onto cloud (on instance)
 - Customization – OS, Auto scaling, etc
- Limitations:
 - Manual configuration, maintenance, resource management
- Solution: **Moving to Amazon Database services**

Amazon Database Services for various types

- **Relational** – Amazon RDS, Amazon Aurora
- **Key value** – Amazon DynamoDB
- **In-memory** – Amazon ElastiCache
- **Document** – DocumentDB
- **Graph** - Neptune
- **Time Series** - Timestream
- **Ledger** - QLDB

Relational databases



Amazon
RDS



Amazon
Redshift



Amazon
Aurora

Non-relational databases



Amazon
DynamoDB

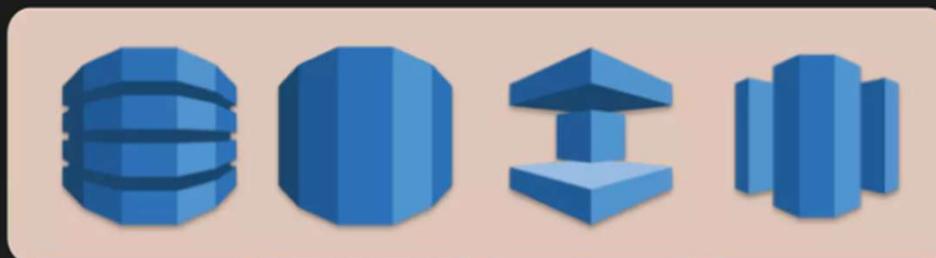


Amazon
ElastiCache



Amazon
Neptune

A managed service for each major DB type



Amazon
DynamoDB

*Document
and key-
value store*

Amazon
RDS

*SQL
database
engines*

Amazon
ElastiCache

*In-memory
cache*

Amazon
Redshift

*Data
warehouse*

Amazon RDS

- Amazon Relational Database Service (Amazon RDS) is a web service that makes it **easier to set up, operate, and scale** a relational database in the AWS Cloud.
- It provides **cost-efficient, resizable capacity** for an industry-standard relational database and manages common database administration tasks.
- **Managed Database service** provided by AWS

Amazon Relational DB (RDS) - Key Features

- **Simplifies** setup, operation, and scaling of relational databases in the cloud.
- Easy Deployment
- Cost-efficient and resizable capacity
- **Automates** tasks like **provisioning, setup, patching, and backups**
- Enhances **performance, availability, security, scalability and compatibility**
- Optimized for memory, performance, or I/O
- **Multi Database engines** : Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server.
- **AWS Database Migration Service (DMS)** to easily **migrate or replicate** your existing databases to Amazon RDS.

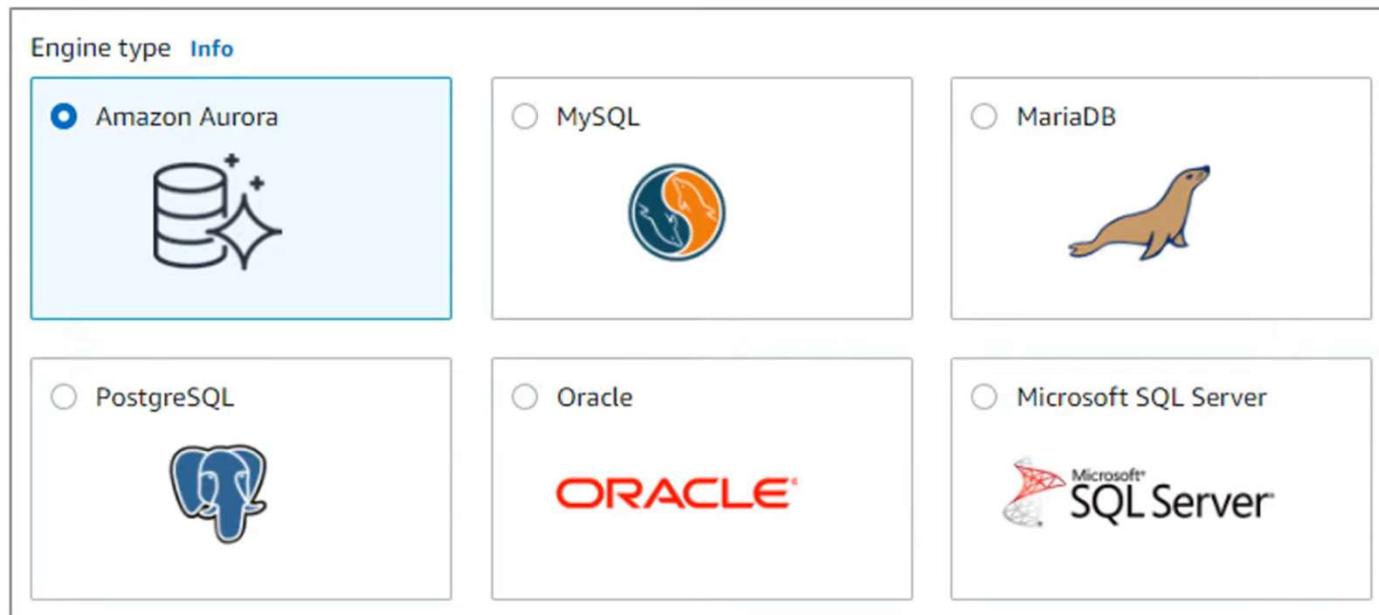
Feature	On-premises management	Amazon EC2 management	Amazon RDS management
Application optimization	Customer	Customer	Customer
Scaling	Customer	Customer	AWS
High availability	Customer	Customer	AWS
Database backups	Customer	Customer	AWS
Database software patching	Customer	Customer	AWS
Database software install	Customer	Customer	AWS
Operating system (OS) patching	Customer	Customer	AWS
OS installation	Customer	Customer	AWS
Server maintenance	Customer	AWS	AWS
Hardware lifecycle	Customer	AWS	AWS
Power, network, and cooling	Customer	AWS	AWS

RDS Properties and Uses

- **Access pattern** – Light analytics
- **Size** – Low TB
- **Performance** – Mid to high throughput, low latency
- **Business Use case** – transactional OLAP
- Works well for applications:
 - Complex data
 - Need to combine and join dataset
 - Enforced Syntax rules

RDS delivers high Performance

- **Instance** – On Demand / Reserved
- **Storage** – General SSD / Provisioned IOPS
- Supports 6 database engine types

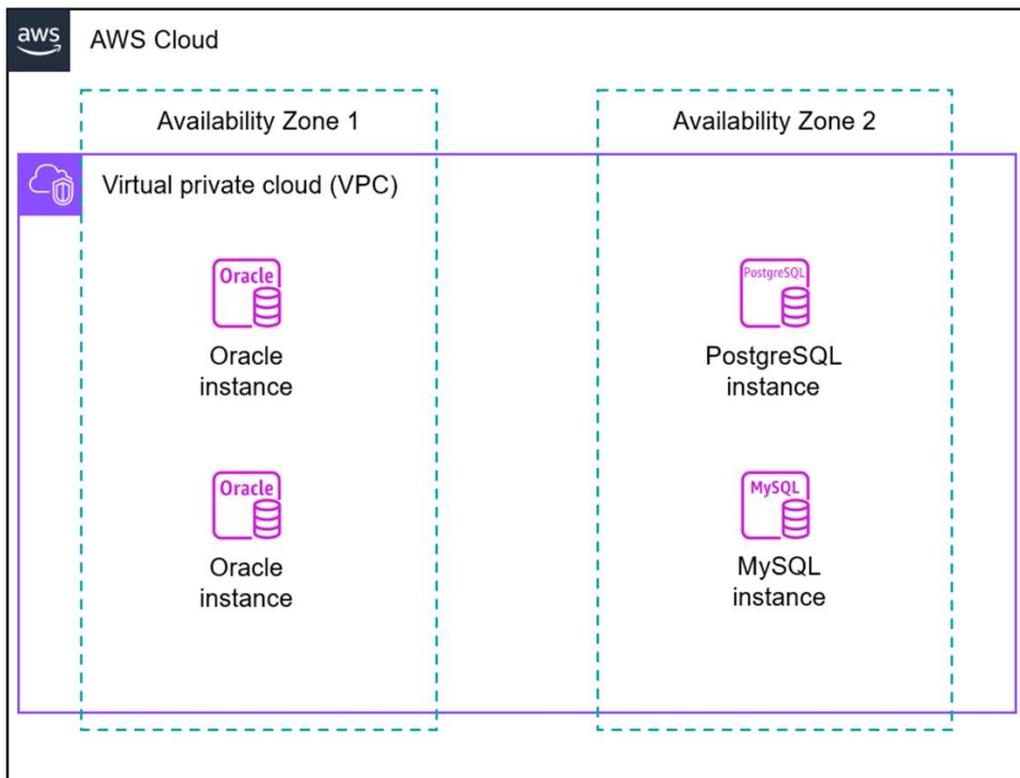


RDS shared responsibility model

- **Amazon RDS handles:**
 - Hosting software components and infrastructure for DB instances and clusters.
 - Automating administrative tasks like provisioning, setup, patching, and backups.
- **User responsible for:**
 - **Query tuning:** Optimizing SQL queries for better performance.
 - **Monitoring performance:** Using tools like **Amazon RDS Performance Insights** to identify slow or problematic queries.
- **Performance depends on:**
 - Database design
 - Data size and distribution
 - Application workload
 - Query patterns

RDS DB instances

- A **DB instance** is the core unit of Amazon RDS, representing an isolated database environment in the AWS Cloud.
- Each DB instance can host **one or more user-created databases**.
- DB instances are deployed within a **Virtual Private Cloud (VPC)** for secure networking.
- A VPC can span **multiple Availability Zones (AZs)**, and each AZ can contain **multiple DB instances**, enhancing availability and fault tolerance.
- **Create and modify** a DB instance by using the AWS Command Line Interface (AWS CLI), the Amazon RDS API, or the AWS Management Console.



DB instance class and storage

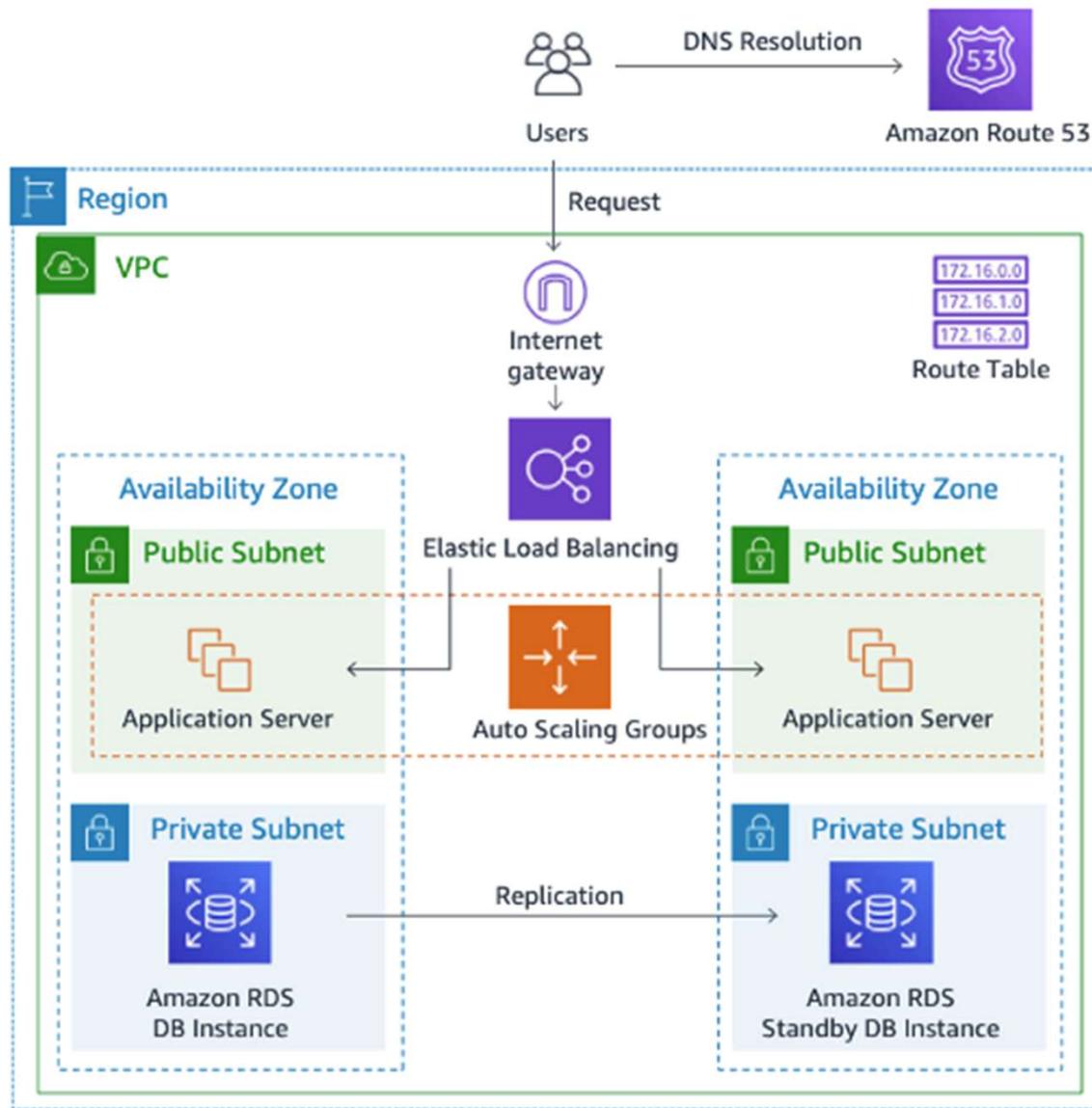
- A ***DB instance class*** determines the computation and memory capacity of a DB instance. Select the DB instance class that best meets your requirements.
 - General purpose – db.m*
 - Memory optimized – db.z*, db.x*, db.r*
 - Compute optimized – db.c*
 - Burstable performance – db.t*
- **DB instance storage** comes in the following types:
 - **General Purpose (SSD)** : This cost-effective storage type is ideal for a broad range of workloads running on medium-sized DB instances. General Purpose storage is best suited for development and testing environments.
 - **Provisioned IOPS (PIOPS)** : This storage type is designed to meet the needs of I/O-intensive workloads, particularly database workloads, that require low I/O latency and consistent I/O throughput. Provisioned IOPS storage is best suited for production environments.
 - **Magnetic** : Amazon RDS supports magnetic storage for backward compatibility. Recommended to use General Purpose SSD or Provisioned IOPS SSD for any new storage needs.

RDS Application Architecture

- Basic functionality of Amazon RDS is the same whether it's running in a VPC or not
- Amazon RDS uses Network Time Protocol (NTP) to synchronize the time on DB instances.

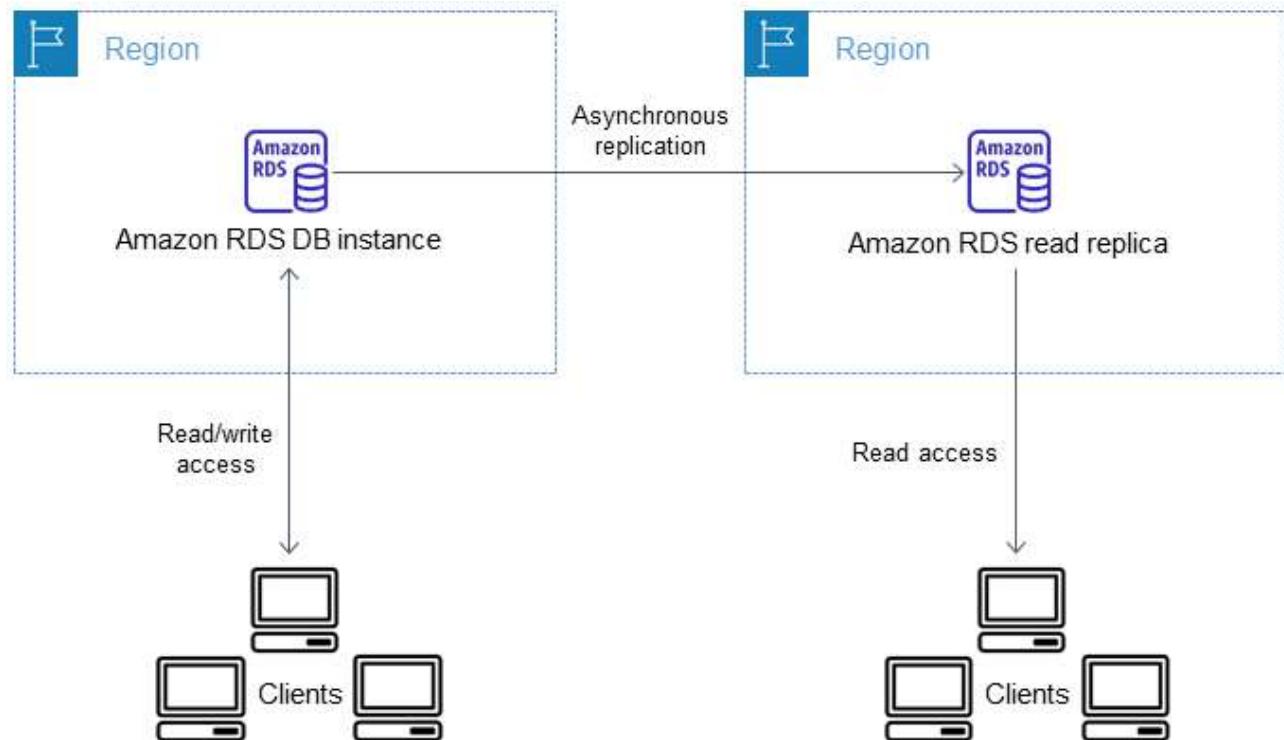
Primary components:

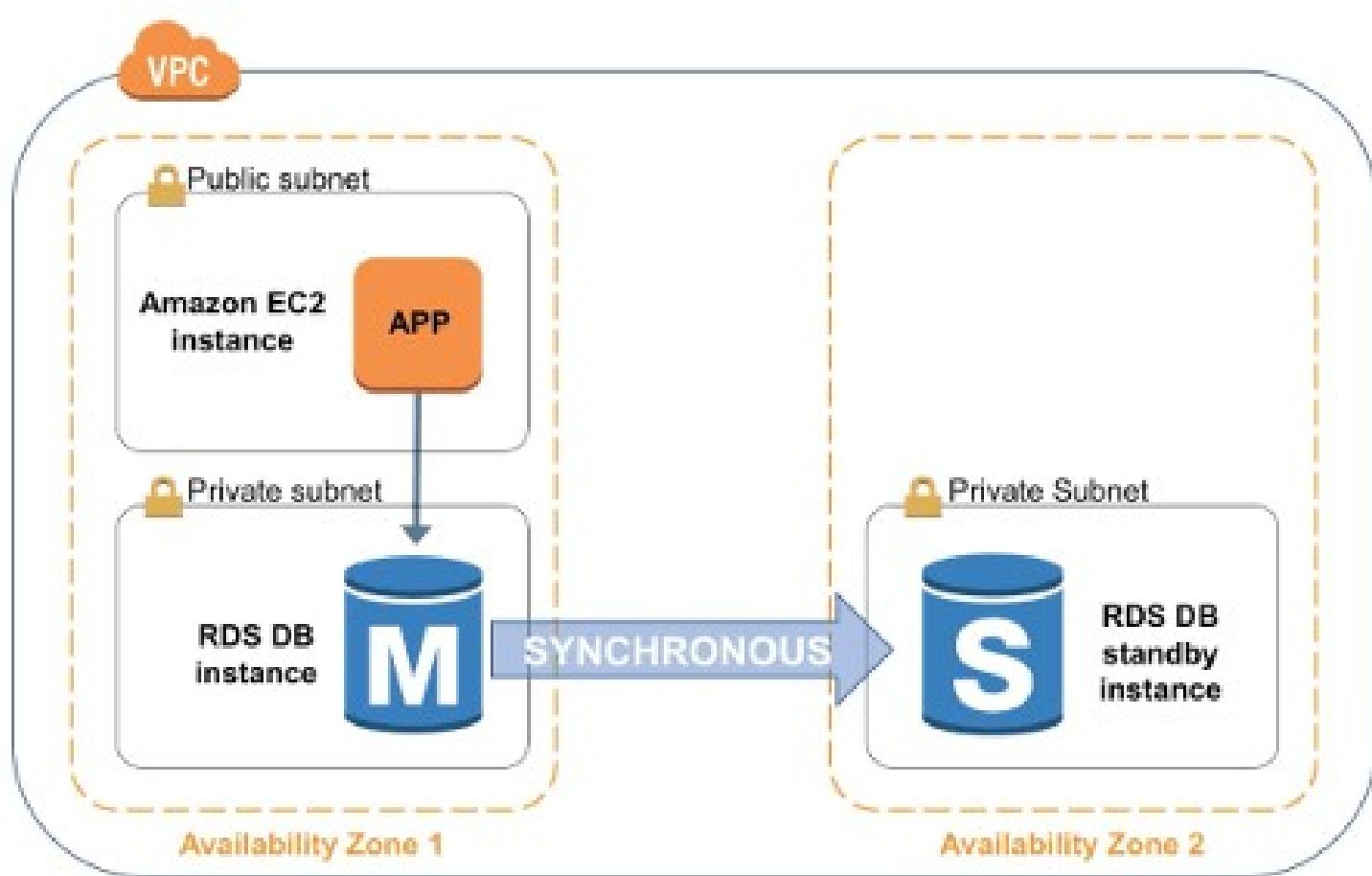
- **Elastic Load Balancing**
- **Application servers**
- **RDS DB instances**
 - **DB engines**
 - **DB instance classes**
 - **DB instance storage**
 - **DB instances in VPC**



Regions and Availability Zones

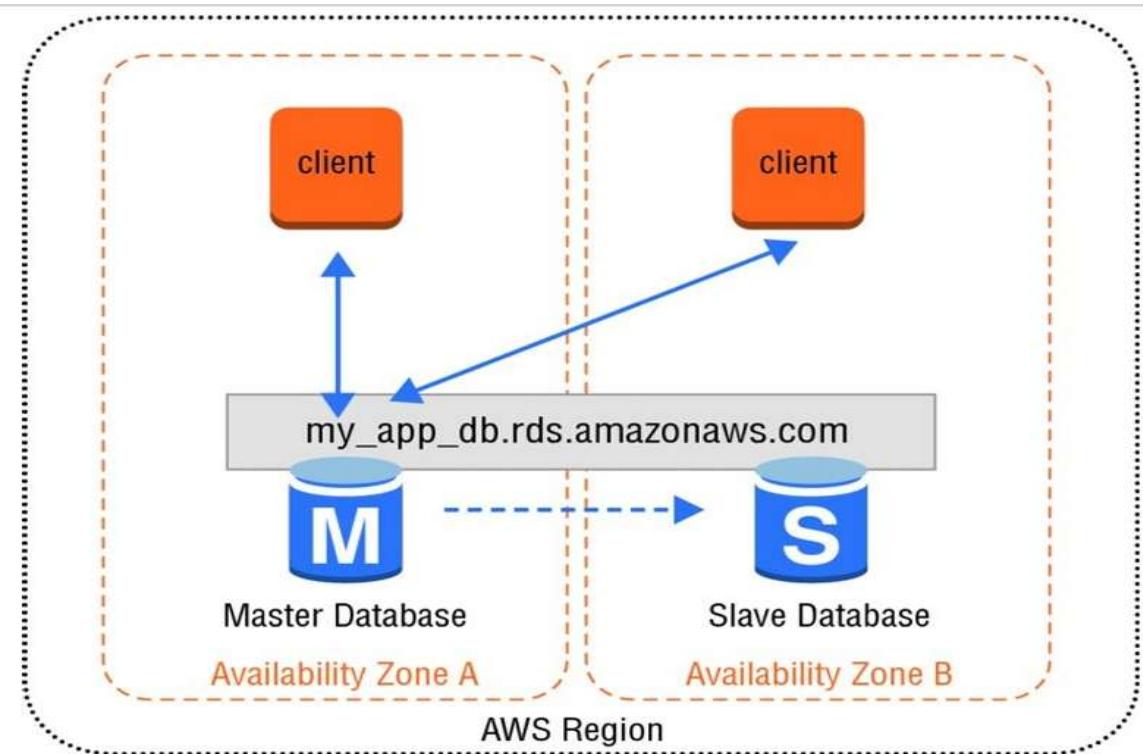
- Can create your DB instances in multiple Regions.
- RDS DB instance in one Region that replicates asynchronously to a standby DB instance in a different Region.
- If one Region becomes unavailable, the instance in the other Region is still available.





Multi-AZ Amazon RDS Architecture

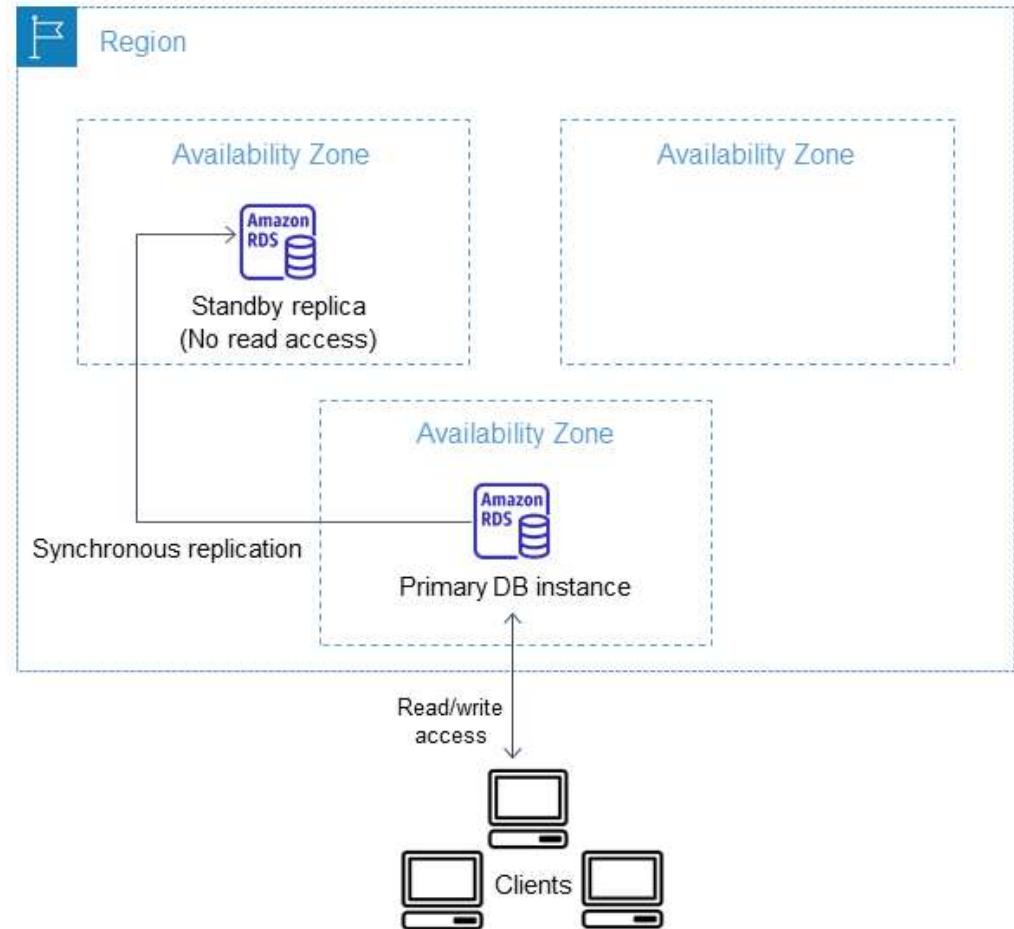
- Run your DB instances in more than one Azs.
- Automatically replicates the data from the master database or primary instance to the slave database
- Amazon RDS automatically performs a failover in the event of any of the following:
 - Loss of availability in primary
 - Availability Zone Loss of network connectivity to primary database
 - Compute unit failure on primary database
 - Storage failure on primary database

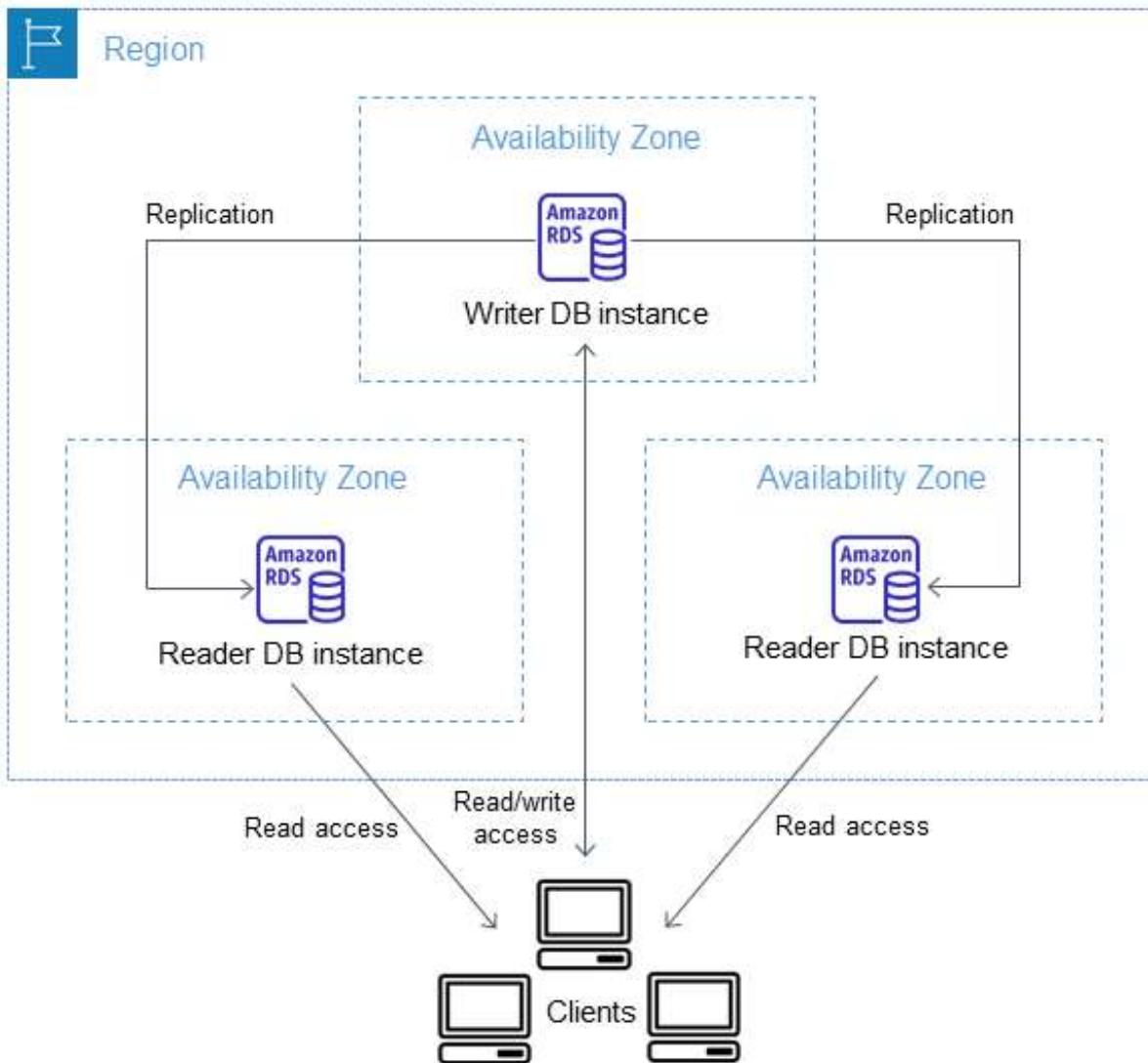


Multi-AZ DB Instance deployment

Advantages:

- Providing data redundancy and failover support
- Eliminating I/O freezes
- Minimizing latency spikes during system backups
- Serving read traffic on secondary DB instances (Multi-AZ DB clusters deployment only)

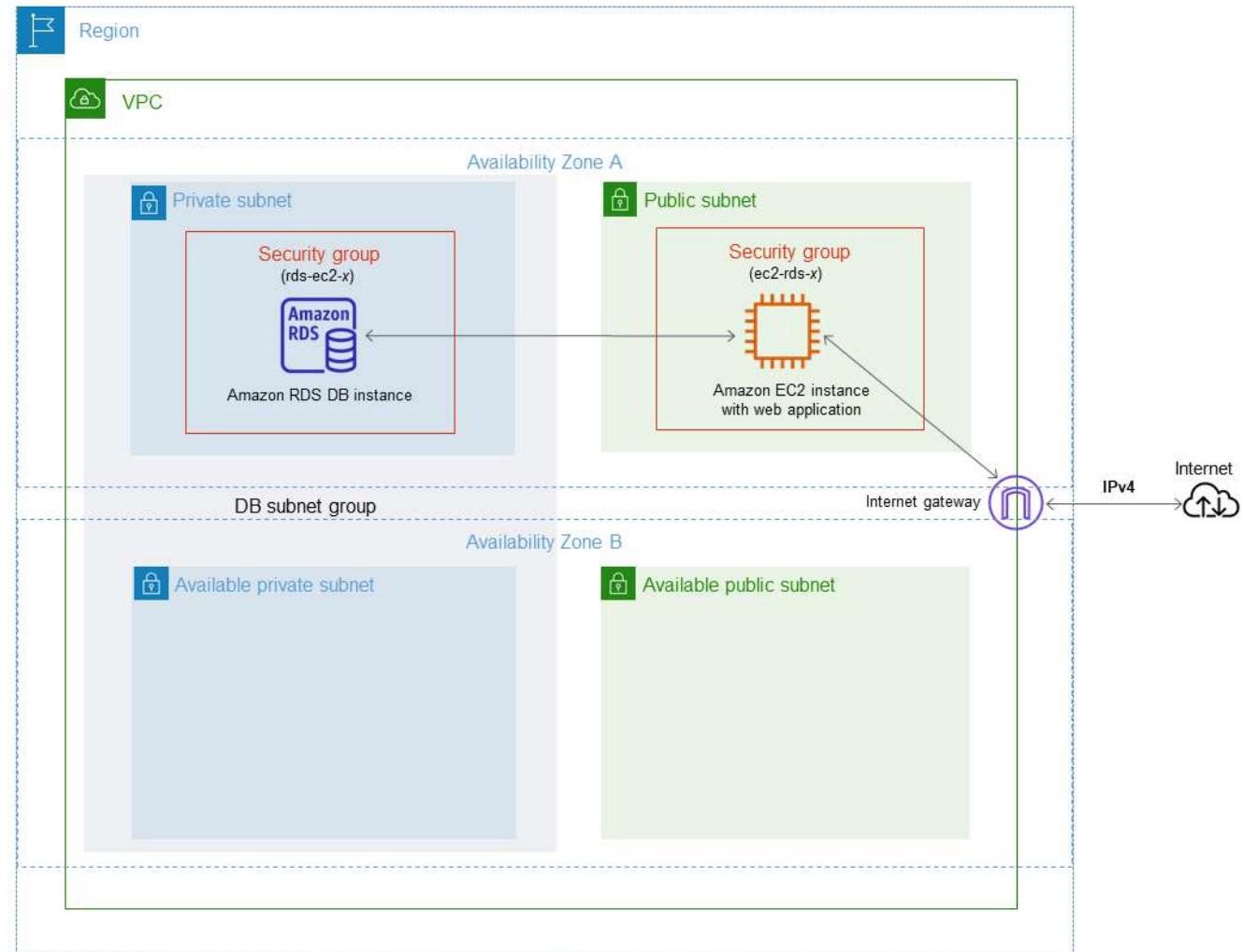


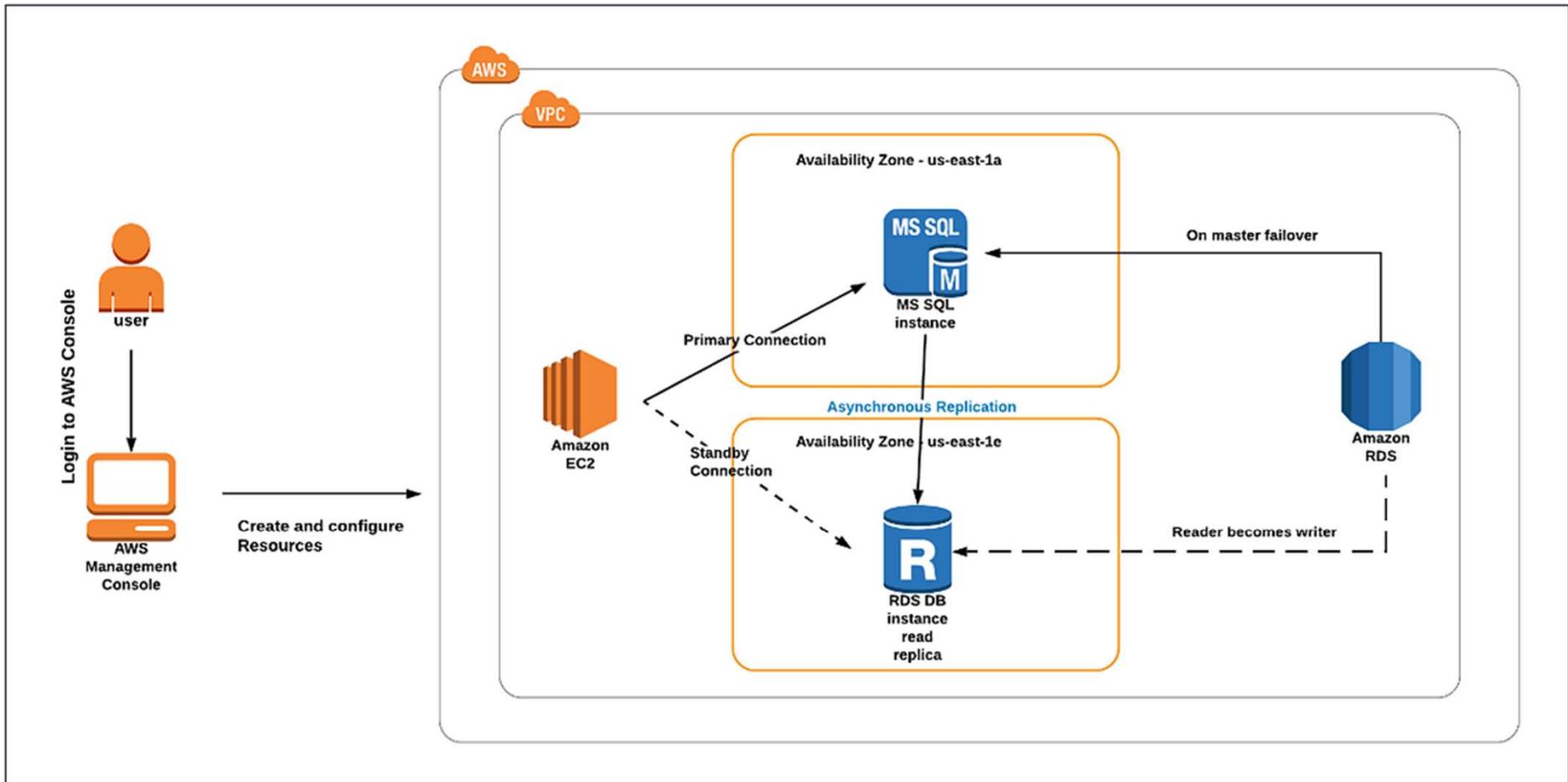


- **Multi-AZ deployments:** For disaster recovery only, not performance.
- **Standby instance:** Not available for queries.
- **Performance improvement:** Use **read replicas** or caching (e.g., Amazon ElastiCache).
- **Scaling Up (Vertical):** Add more compute, memory, or storage to a single DB instance.
Easy with Amazon RDS, supports storage growth (5 GB–6 TB), except SQL Server.
- **Scaling Out (Horizontal):** Used when vertical scaling isn't enough.
- **Partitioning/Sharding:** Splits database into multiple instances; needs app logic; limits cross-shard queries; common in NoSQL (e.g., DynamoDB, Cassandra).
- **Read Replicas:** Offload reads from primary DB, boost read-heavy workloads, support maintenance availability, and reporting. Supported for MySQL, PostgreSQL, MariaDB, Aurora.

Access control with Security Group

- *Security group* controls the access to a DB instance by allowing access to IP address ranges or Amazon EC2 instances that you specify





Use cases

Web and mobile applications	<ul style="list-style-type: none">✓ High throughput✓ Massive storage scalability✓ High availability
Ecommerce applications	<ul style="list-style-type: none">✓ Low-cost database✓ Data security✓ Fully managed solution
Mobile and online games	<ul style="list-style-type: none">✓ Rapidly grow capacity✓ Automatic scaling✓ Database monitoring

Amazon RDS- Summary

- Automated Backups
 - Retention period enabled default - upto 35 days
- Manual Snapshots
- Scale up or down with resizable instance types
- Read replicas for improved scalability
- Off load read traffic to read replicas
- Cache infront of RDS
- Ease of migration (cross-region)
- Multi AZ – synchronous replication within region
- Handles failover automatically
- **Multi AZ for safety, Replicas/ caching for speed**

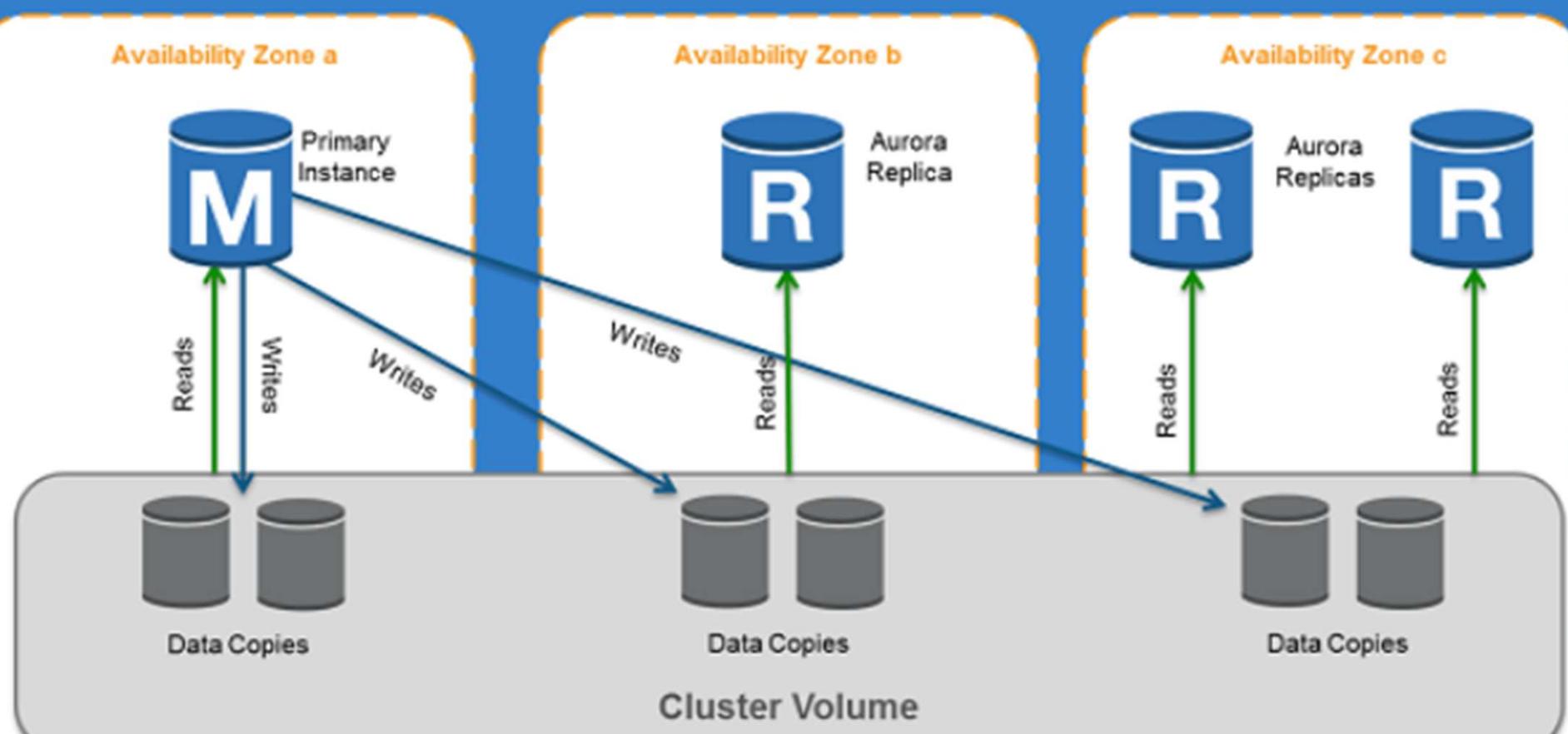
Amazon Aurora

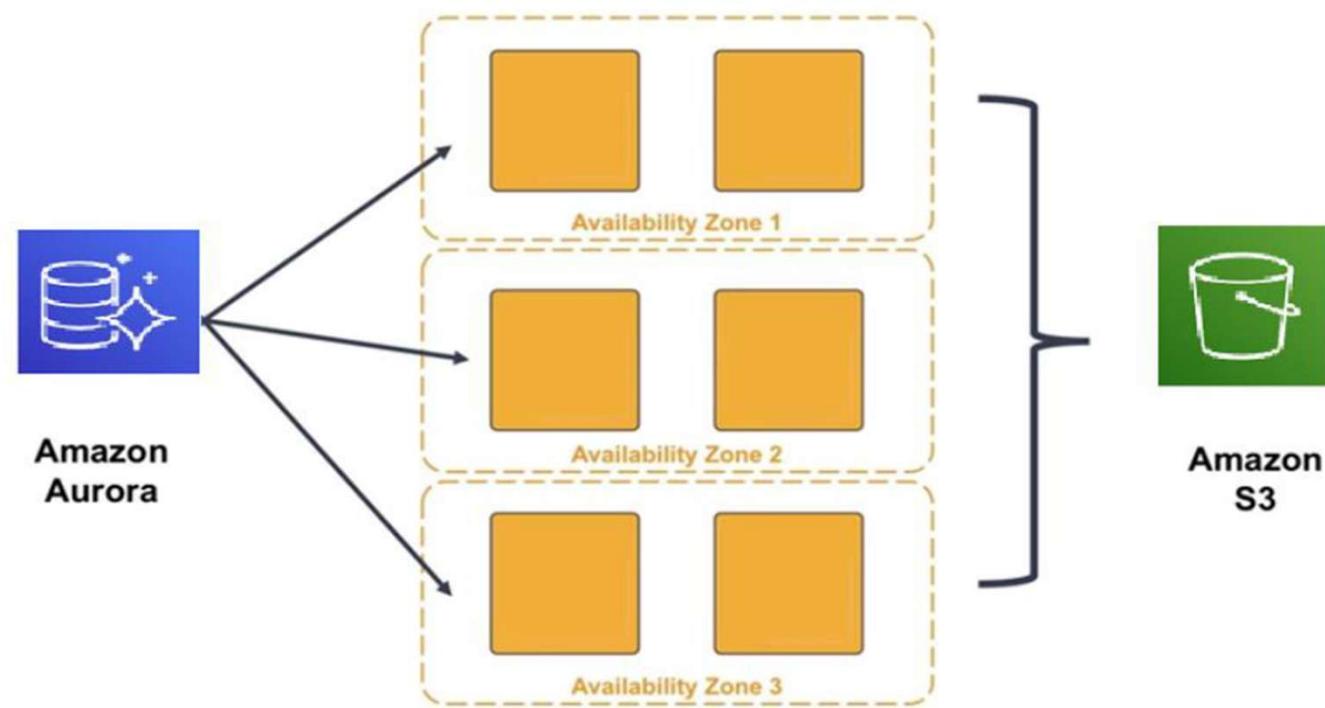
- Reinvented - Relational database for cloud
- 5 times better performance than RDS
- Runs in amazon VPC
- RDS handles admin tasks for Aurora
- Pay only for the storage you use
- Storage automatically grows increments of 10 GB to 64 TB.

Amazon Aurora

- Amazon Aurora (Aurora) is a fully managed relational database engine that's compatible with MySQL and PostgreSQL.
- The code, tools, and applications you use today with your existing MySQL and PostgreSQL databases can be used with Aurora. With some workloads, Aurora can deliver up to five times the **throughput of MySQL** and up to three times the throughput of PostgreSQL without requiring changes to most of your existing applications.
- Aurora includes a high-performance storage subsystem. Its **MySQL- and PostgreSQL-compatible** database engines are customized to take advantage of that fast distributed storage.
- The underlying storage grows automatically as needed. An **Aurora cluster volume can grow to a maximum size of 128 tebibytes (TiB)**. Aurora also automates and standardizes database clustering and replication, which are typically among the most challenging aspects of database configuration and administration.

Amazon Aurora DB Cluster





Features of Amazon Aurora

- 1. Availability and Durability:** AWS Aurora has a feature of fault-tolerant and self-mending storage built for the cloud. It offers an incredible accessibility of 99.99%. The storage in the cloud replicates the 6 copies of the information across 3 Availability Zones. The AWS Aurora backs up the data continuously due to the safety purpose and precaution from storage failure.
- 2. Performance and Scalability:** AWS Aurora provides 5 times the throughput of ordinary MySQL. This performance is comparable with enterprise databases, at 1/10th the cost. The user can scale database preparation up and down for smaller to larger instance varieties as per the user needs.
- 3. Fully Managed:** The Amazon Aurora is managed by Amazon Relational Database Service (RDS). The user no longer needs to stress over database management tasks, for example, hardware provisioning, software fixing, setup, configuration, or backups. Aurora consequently and consistently screens and backs up the database to Amazon S3, empowering granular point-in-time recuperation.
- 4. Security:** Amazon Aurora provides numerous degrees of security to the database to improve it among others. On an encoded Amazon Aurora occurrence, data within the underlying storage is encrypted. The administration is through AWS Key Management Service and encryption of information in transit using SSL. In addition, there are the automatic reinforcements, snapshots, and replica within the same cluster.

5. Migration Support: MySQL and PostgreSQL compatibility make Amazon Aurora a convincing target for database relocations to the cloud. If the users want to migrate from MySQL or PostgreSQL, can see migration documentation for a list of tools and options. To move from commercial database engines, the user can use the AWS Database Migration Service for a safe migration with minimal downtime.

6. Compatibility with MySQL and PostgreSQL: The Amazon Aurora database engine is perfectly compatible with existing MySQL and PostgreSQL open supply databases, and adds compatibility for new releases frequently. This means that the user can relocate MySQL or PostgreSQL databases to Aurora using standard MySQL or PostgreSQL import/export tools or previews. It also means the user using code, applications, drivers, and tools with existing databases can also use it with Amazon Aurora with little or no modification.

7. Cost: Amazon Aurora is designed to be cost-effective. You only pay for what you use, and you can scale your database up or down as needed. Additionally, Aurora provides cost-saving features such as automated storage optimization and the ability to pause or stop your database when it's not in use.

Summary - Aurora

- High Availability
- Automatic replication- replicated 6 ways across 3 AZs
- Global reach
- Compatible with MySQL and PostgreSQL
- 5 times high performance

Amazon DynamoDB

- AWS DynamoDB is a **Non-Relational, NoSQL database**.
- DynamoDB is a **serverless, highly available NoSQL database** that offers SSD-backed, low-latency performance with automatic scaling and replication.
- Zero infrastructure management, zero downtime maintenance, instant scaling to any application demand, and pay-per-request billing.
- No cold starts, no version upgrades, and no maintenance windows

	Relational (SQL)	Non-Relational												
Data Storage	Rows and columns	Key-value, document, graph												
Schemas	Fixed	Dynamic												
Querying	Uses SQL	Focuses on collection of documents												
Scalability	Vertical	Horizontal												
Example	<table border="1"> <thead> <tr> <th>ISBN</th> <th>Title</th> <th>Author</th> <th>Format</th> </tr> </thead> <tbody> <tr> <td>3111111223439</td> <td>Withering Depths</td> <td>Jackson, Mateo</td> <td>Paperback</td> </tr> <tr> <td>312222223439</td> <td>Wily Willy</td> <td>Wang, Xiulan</td> <td>Ebook</td> </tr> </tbody> </table>	ISBN	Title	Author	Format	3111111223439	Withering Depths	Jackson, Mateo	Paperback	312222223439	Wily Willy	Wang, Xiulan	Ebook	<pre>{ ISBN: 3111111223439, Title: "Withering Depths", Author: "Jackson, Mateo", Format: "Paperback" }</pre>
ISBN	Title	Author	Format											
3111111223439	Withering Depths	Jackson, Mateo	Paperback											
312222223439	Wily Willy	Wang, Xiulan	Ebook											

AWS RDS	AWS DynamoDB
Supports complex data	Supports simple structured data
Stores data in database tables	Stores data in documents
More expensive than AWS DynamoDB	Cheaper than AWS RDS
Slower than AWS DynamoDB	Faster than AWS RDS

Payment model

- **On-demand:** DynamoDB's on-demand model charges for each request made, and the price depends on the type of request, operation, table class, and AWS region.
- **Provisioned capacity:** This model reserves a certain level of performance for your database, which is paid for on an hourly basis.
- **Reserved capacity:** You can use reserved capacity to further reduce the cost of provisioned throughput. For example, a one-year upfront commitment for 100 write capacity units costs \$150.00, and a one-year upfront commitment for 100 read capacity units costs \$30.00.

DynamoDB

- **Performance at scale**

- It offers consistent, **single-digit millisecond** response times
- You can build applications with **virtually unlimited throughput**

- **Serverless**

- **No servers** to provision, patch, or manage
- **No software** to install, maintain, or operate

DynamoDB

- **Enterprise-ready**
 - It supports ACID transactions
 - It encrypts all data at rest by default
 - Multi-Region replication (global tables) is available
 - It provides fine-grained identity and access control
 - You can perform full backups of data with no performance degradation

Amazon DynamoDB Characteristics

Works well for applications that:

- Have simple high-volume data (high-TB range)
- Must scale quickly
- Don't need complex joins
- Require ultra-high throughput and low latency

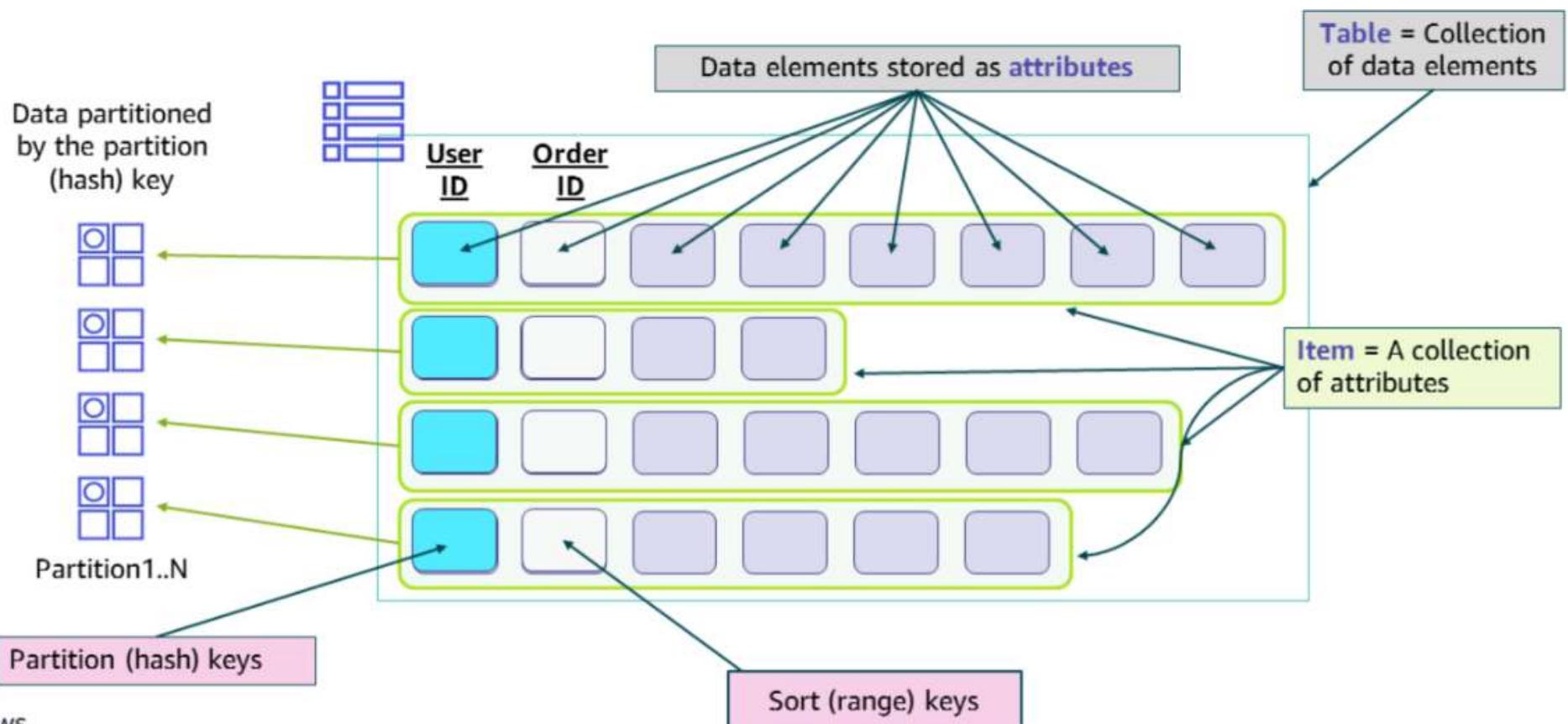
Key features

- NoSQL tables
- Items can have differing attributes
- In-memory caching
- Support for peaks of more than 20 million requests per second

Data Model

- DynamoDB does **not enforce strict schemas**
- One of the structure types of the non-relational database is **key-value pairs**.
- **Data key** represents an **item**, and **data value** represents that **item attributes**.
- **Tables hold items**.
- An item has a **variable number of attributes**. You can **remove any** attribute from any item at any time.
- **Items** are grouped into **partitions** using a **partition key**, optionally ordered within a partition using a **sort key**.
- Each attribute consists of **a key-value pair**. DynamoDB supports **key-value GET/PUT operations** that use a user-defined primary key (**partition key**, also known as **a hash key**)

Amazon DynamoDB data model



Primary Key

Partition Key Sort Key

Attribute

AccountId	CreationDate	OriginCountry	Details
1	2019-09-30	USA	{...JSON...}
2	2019-10-01	Canada	{...JSON...}
2	2019-10-02	USA	{...JSON...}
3	2019-10-03	Germany	{...JSON...}

Amazon DynamoDB data model

- The primary key is the **only required attribute** for items in a table and it uniquely identifies each item
- You specify the primary key when you **create a table**.
- In addition, DynamoDB provides **flexible querying** by allowing queries on **non-primary key attributes** using **Global Secondary Indexes** and **Local Secondary Indexes**
- **Global** → You can query across the entire table, not restricted to the base table's partition key (using alternative keys).
- **Local** → Queries are limited to items within a single partition (same partition with different sort orders).

DynamoDB Indexing Example: Orders Table with GSI and LSI

Base Table: Orders
Primary Key = CustomerID (PK) + OrderID (SK)

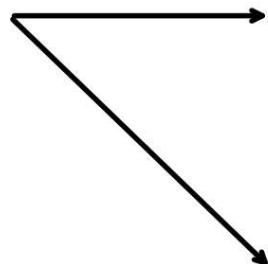
Example Query:
Find Order #123 for Customer A

Global Secondary Index (GSI)
Partition Key = OrderDate
Sort Key = Status

Example Query:
Find all orders placed on 2025-09-01

Local Secondary Index (LSI)
Partition Key = CustomerID (same)
Sort Key = OrderDate

Example Query:
Find all orders of Customer A sorted by OrderDate



Global Secondary Indexes (GSI)

- Partition and Sort Key: A GSI can have a **different partition key and sort key** than the primary key of the table. This means you can query your data using different attributes than the table's primary key.
- Global: The term "global" refers to the fact that GSIs allow you **to query data across all partitions of the table**, regardless of how the data is distributed.

Global Secondary Indexes (GSI)

- Data Projection: You can specify **which attributes from the base table are copied** (or "projected") into the index. This can be all attributes, specific attributes, or only the keys
- Consistency: Queries on a **GSI are eventually consistent by default** but can be made strongly consistent by specifying a parameter during the query operation
- For example, if your table's primary key is UserID but you also need to query by Email, you could create a GSI with Email as the partition key.

Local Secondary Indexes (LSI)

- Partition Key: An LSI uses the same partition key as the table's primary key but **allows for a different sort key**. This enables you to run queries on the same partition using a different sort key.
- Local: The term "local" means that the index is "local" to the partition that shares the same partition key as the base table. It **cannot query across partitions** like a GSI.

Local Secondary Indexes (LSI)

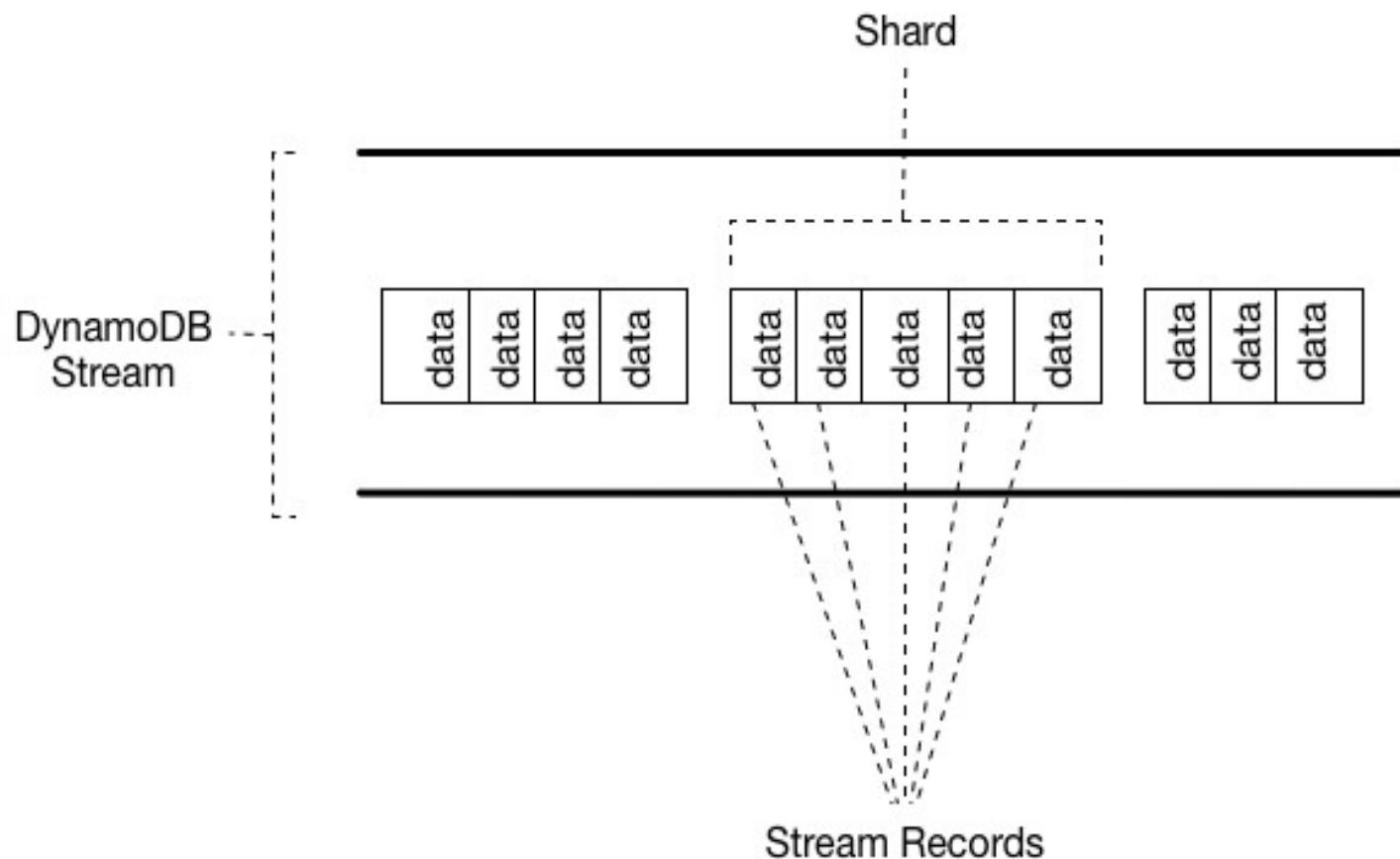
- Data Projection: Similar to GSI, you can **choose which attributes** to project into the LSI.
- Consistency: Queries on an LSI are always **strongly consistent**, which means you'll always see the most up-to-date data when you query an LSI.
- Use Case: LSIs are useful when you need to **query your data on a different attribute within the same partition**.
- For example, if your primary key is a combination of **UserID and Timestamp**, but you want to query based **on UserID and Category**, you could create an LSI with Category as the sort key.

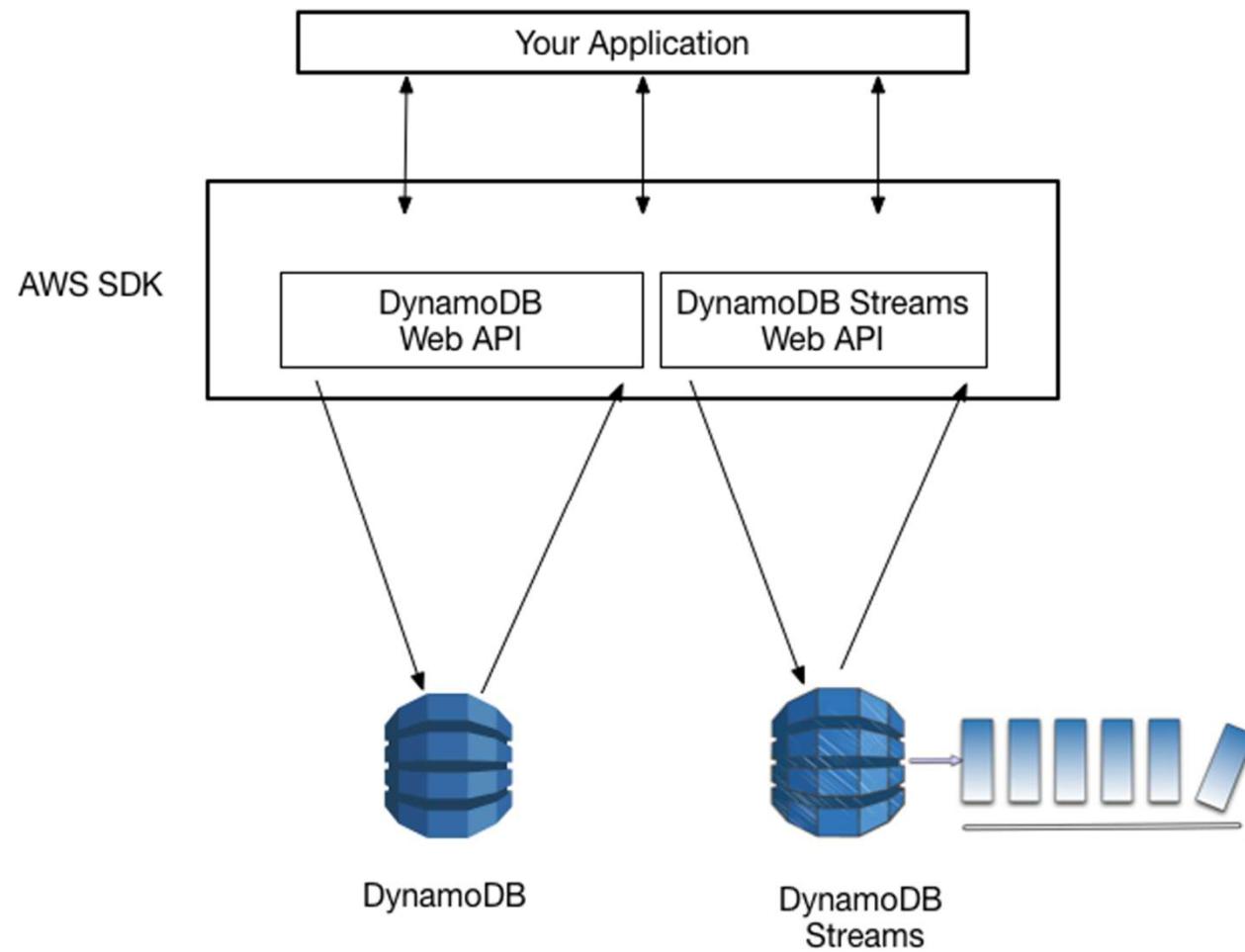
DynamoDB Streams

- DynamoDB Streams **captures a time-ordered sequence of item-level modifications** in any DynamoDB table and stores this information in a **log for up to 24 hours**.
- A ***DynamoDB stream*** is an ordered flow of information about changes to items in a DynamoDB table.
- A ***stream record*** contains information about a data modification to a single item in a DynamoDB table

Stream Records

- To read and process a stream, your application must connect to a DynamoDB Streams endpoint and issue API requests.
- A stream consists of ***stream records***
- **Each stream record is assigned a sequence number**, reflecting the order in which the record was published to the stream.
- Stream records are organized into **groups, or shards**
- The stream records within a shard are **removed automatically after 24 hours**.
- Shards are **ephemeral**
- Because shards have a lineage (parent and children), an application must always process a parent shard before it processes a child shard.
- This helps ensure that the stream records are also processed in the correct order.





Benefits of DynamoDB stream

- Each stream record appears **exactly once in the stream**.
- For each item that is modified in a DynamoDB table, the stream records **appear in the same sequence** as the actual modifications to the item.
- DynamoDB Streams writes stream records in near-real time so that you can build applications that consume these streams and take action based on the contents.

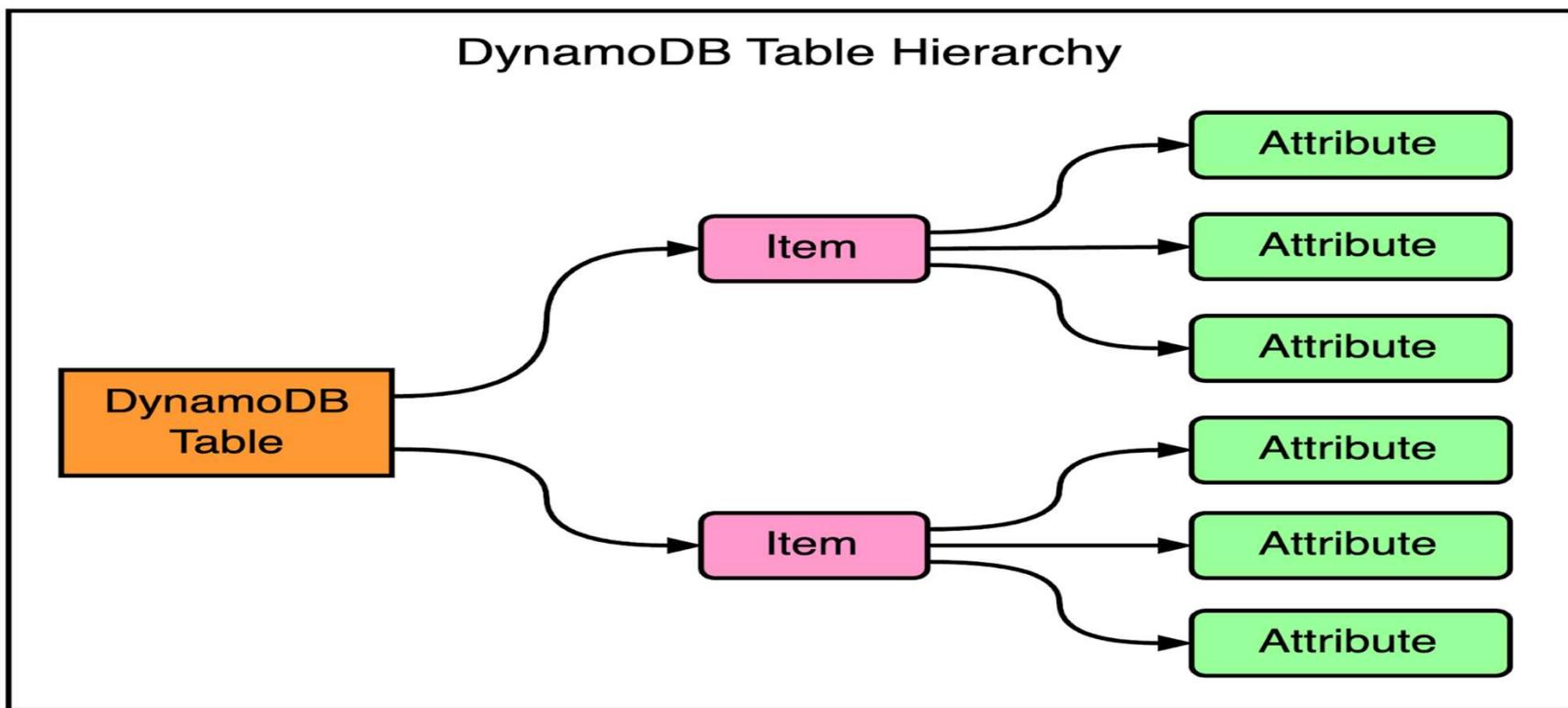
DynamoDB API Actions

- ListStreams
- DescribeStream
- GetSharedIterator
- GetRecords

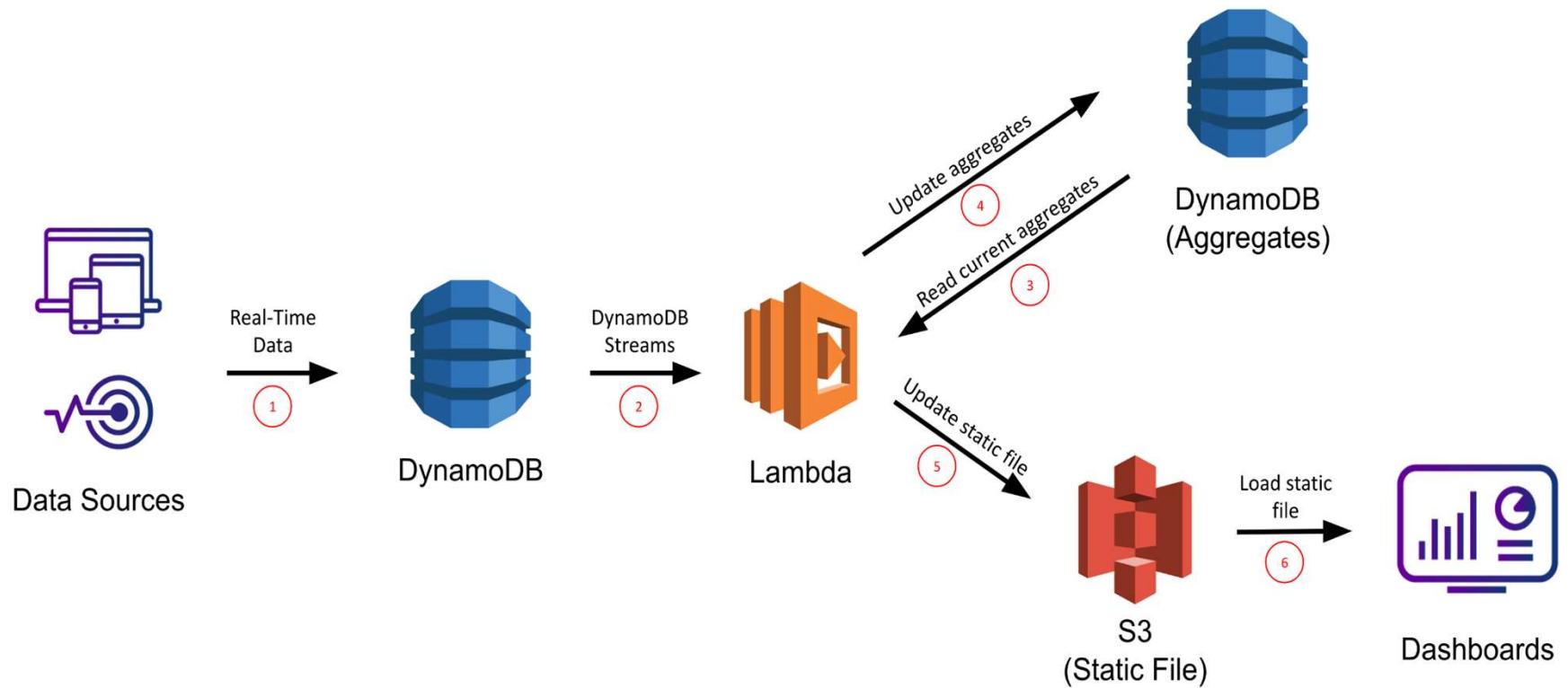
Data Retention Limit

- All data in DynamoDB Streams is subject to a **24-hour lifetime**.
- You can **retrieve and analyze** the last 24 hours of activity for any given table. However, data that is older than 24 hours is susceptible to trimming (removal) at any moment.
- If you **disable a stream** on a table, the data in the stream continues to be **readable for 24 hours**. After this time, the data expires and the stream records are automatically deleted.
- There is **no mechanism for manually deleting an existing stream**. You must **wait until the retention limit expires (24 hours)**, and all the stream records will be deleted.

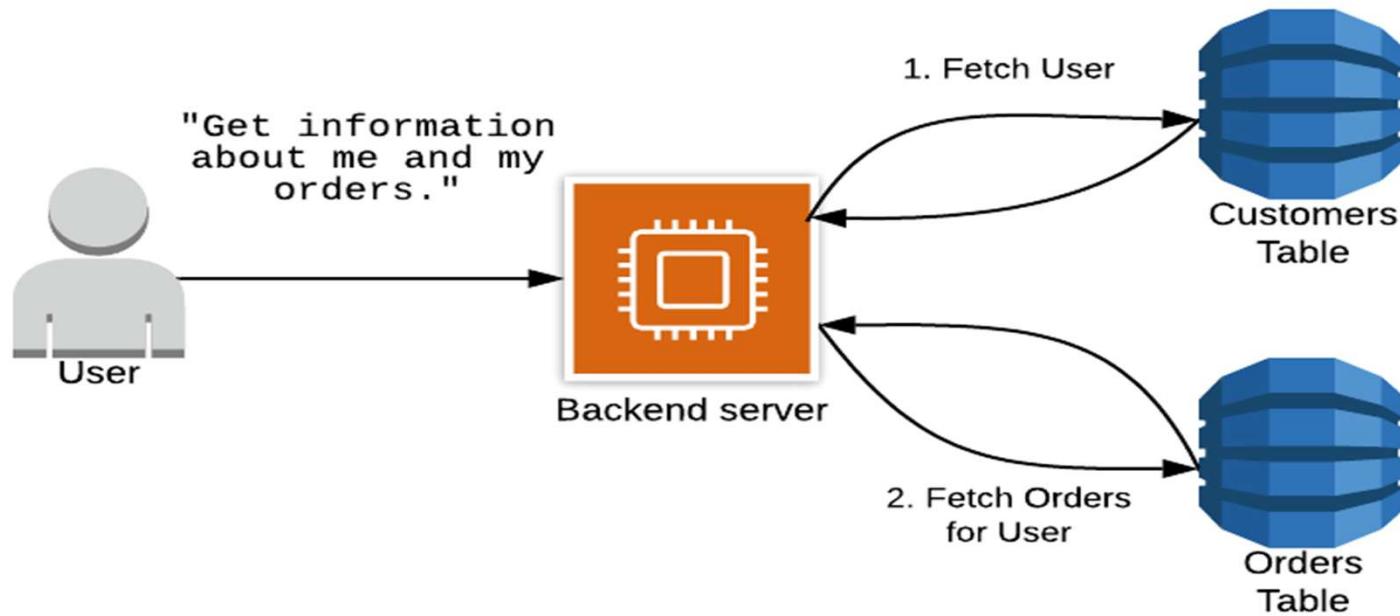
DynamoDB Table Hierarchy



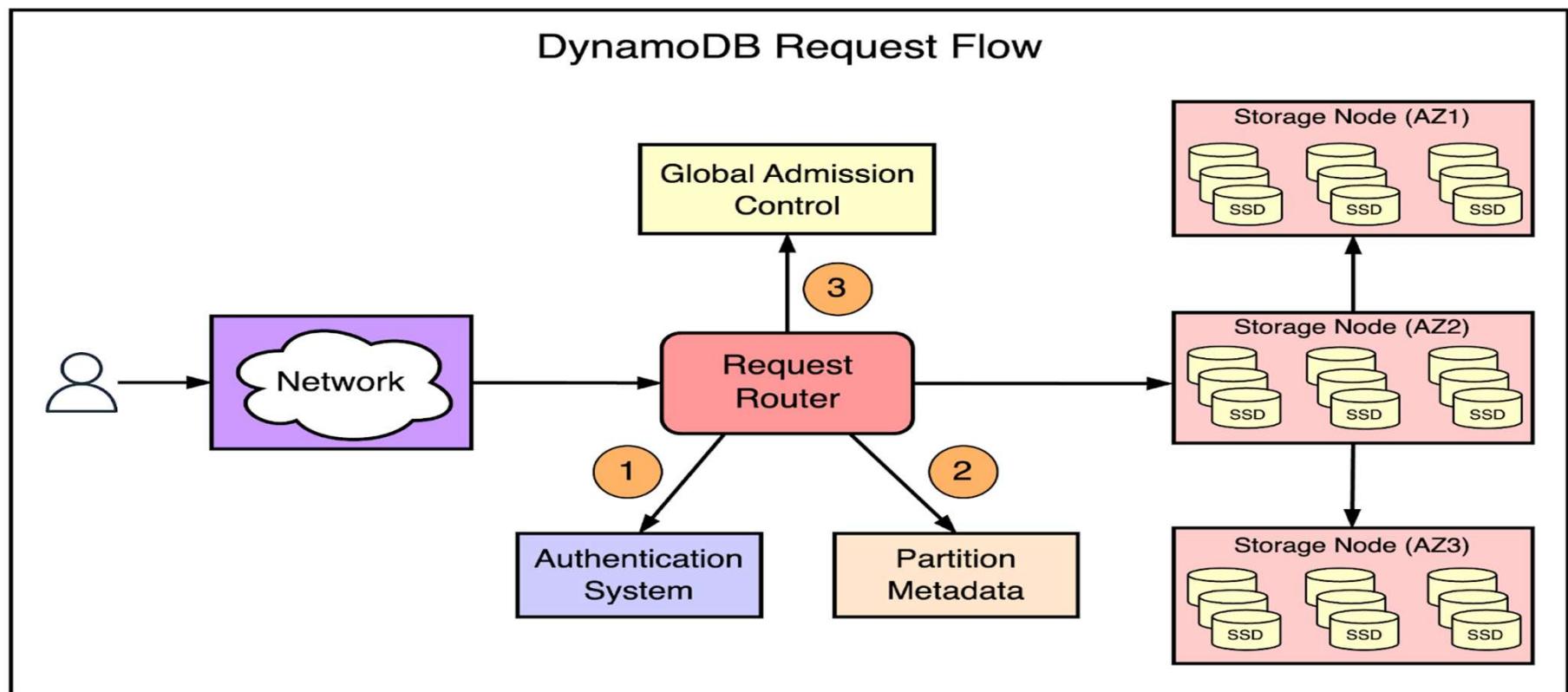
Analytics use case with DynamoDB



Usecase 2: Fetching order info from DynamoDB



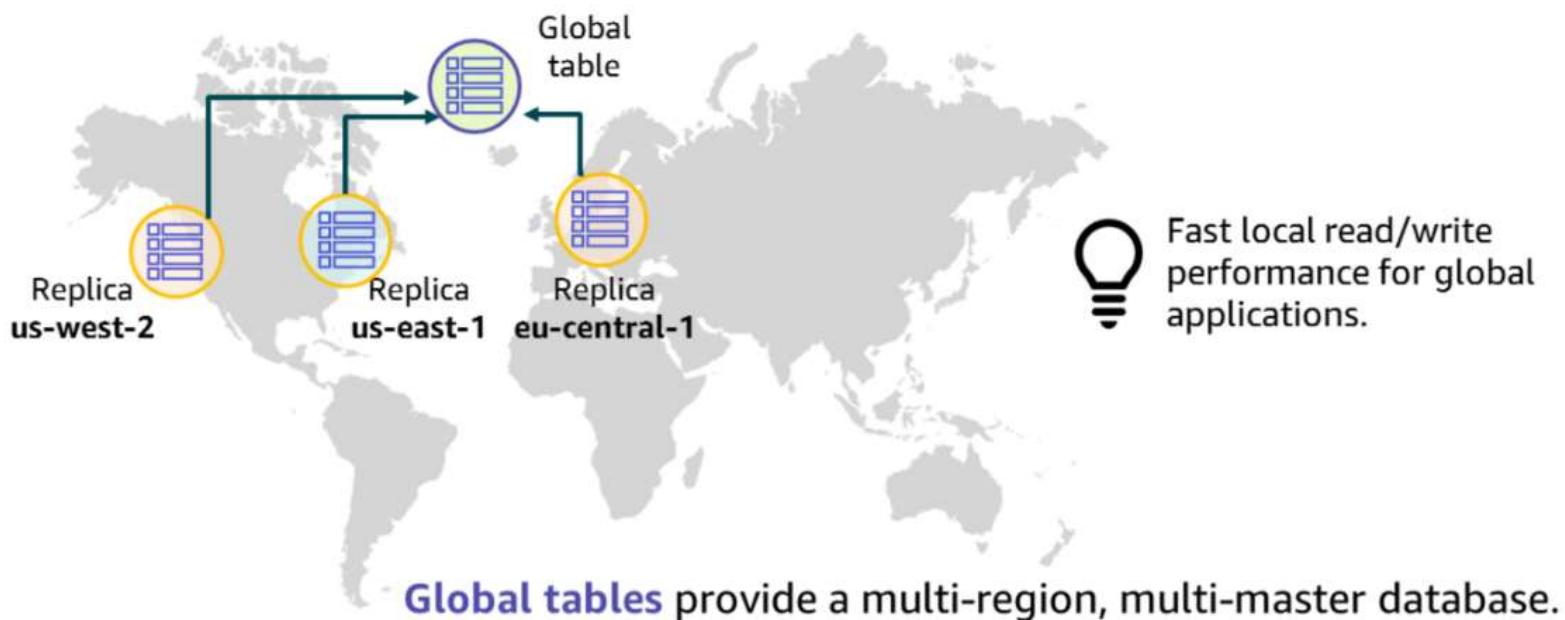
DynamoDB Request Flow



Design work Flow

- **Authenticate request** → IAM credentials check.
- **Find partition** → Identify the right partition holding the data.
- **Check admission control** → Ensure request doesn't exceed capacity.
- **Route to storage nodes** → Read/write from SSDs across 3 Availability Zones.
- Gives **scalability, low-latency, and fault tolerance**.

Amazon DynamoDB

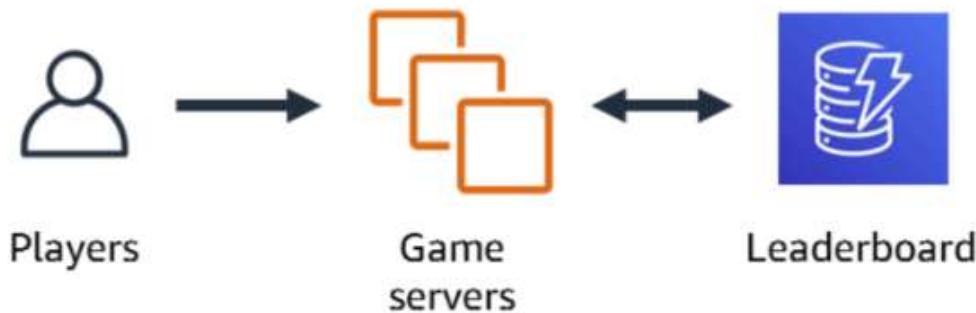


Use cases

- Scale gaming platforms - Handles **high-velocity writes and low-latency reads** across global regions
- Deliver seamless retail experiences - Scales to handle **peak shopping events** without downtime.
- Create media metadata stores - Supports **flexible queries** using Global Secondary Indexes (GSI) for different attributes like *actor* or *release year*.
- Develop software applications - Provides **fast session lookups** and scales automatically with demand
- Ideal for applications with known access patterns
- Cost-effective usage based payment model

Amazon DynamoDB Use Case

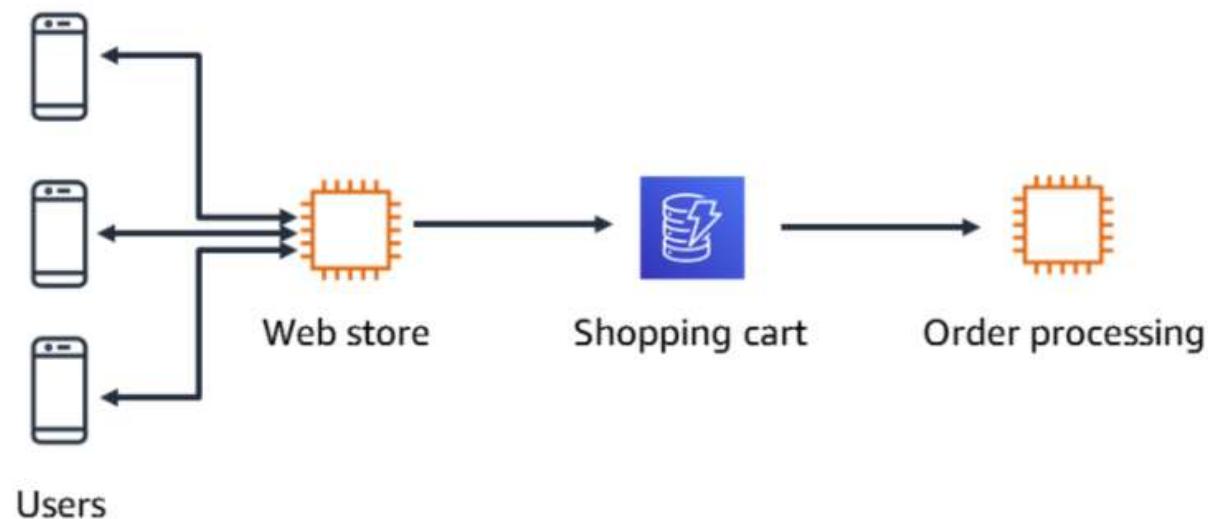
Leaderboards and Scoring



GameScores						
UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	PlayTime
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	PT 0H 1M 5S
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	PT 0H 1M 5S
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	PT 0H 1M 5S
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	PT 0H 1M 5S
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	PT 0H 1M 5S
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	PT 0H 1M 5S
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	PT 0H 1M 5S
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	PT 0H 1M 5S
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	PT 0H 1M 5S

Amazon DynamoDB Use Case

Temporary Data (Online Cart)

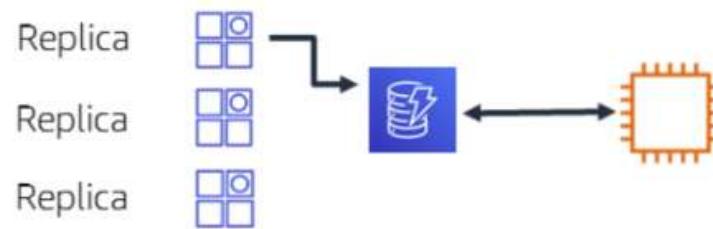


Read consistency Amazon DB

- **Eventually Reads**

- When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation.
- The response might include some stale data.
- If you repeat your read request after a short time, the response should return the latest data

Eventually consistent

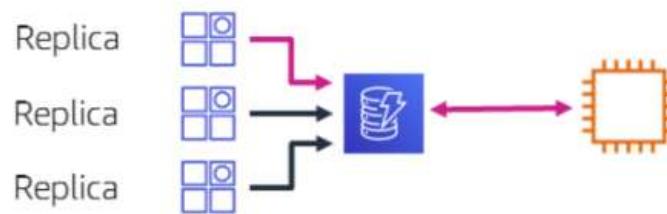


The **default** setting. All copies of data usually reach consistency within **1 second**.

Read consistency Amazon DB

- **Strongly consistent reads**
- When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data.
- The response reflects the updates from all previous write operations that were successful.
- A strongly consistent read might not be available if there is a network delay or outage

Strongly consistent



This feature is optional. Use for applications that require all reads to return a result that reflects all writes before the read.

Summary-Key Features - DynamoDB

- **Fully managed NoSQL service** → offloads admin tasks like setup, provisioning, patching, replication, and scaling.
- **High performance** → consistent, low-latency access using SSD storage (**Single digit millisecond performance**)
- **Automatic scaling** → data and traffic distributed across partitions; read/write capacity can be adjusted dynamically.
- **Reliability & availability** → automatic replication across multiple Availability Zones.
- **Monitoring** → integrates with Amazon CloudWatch for performance metrics

- **Flexible Data Model** → Tables with **primary key (partition key + optional sort key)**.
- **On-Demand & Provisioned Capacity** → Pay for throughput per request or pre-allocate.
- **Global Tables** → Multi-Region, active-active replication.
- **Backup & Restore** → Point-in-time recovery (PITR).
- **Streams** → Capture item-level changes for real-time apps, triggers with AWS Lambda.
- **Security** → Fine-grained IAM access, encryption at rest (KMS), TLS in transit
- **Developer-friendly** → unlimited items per table with predictable latency.

- **Free limits:**

write up to **25 writes/sec** (for 1 KB items) and read up to **50 reads/sec**
(eventually consistent reads of 4 KB items)

Reference

- <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/tutorial-ec2-rds-option1.html>
- <https://dev.to/aws-builders/installing-mysql-on-amazon-linux-2023-1512>