Managed database services on AWS: choose the right one! Explore the options for HA, DR, Scalability, and more

29 January 2025 - 10 min. read

Amazon RDS | *Aurora Serverless* | *Database* | *Disaster Recovery (DR)* | *High Availability (HA)*

“When you ain´t got nothin’, you got nothin’ to lose”

Bob Dylan, Like a Rolling Stone

If you think about this quote, you will find that we always have something: our skills, expertise, and, obviously, data. While no one can take away your skills and expertise, data can disappear in many ways, and it is a pain in the neck to find the right solution to minimize the plethora of issues related to data storage persistence, availability, and scalability.

As you may know, AWS offers different kinds and flavors of managed database services; choosing the right one that fulfills our needs without increasing the bill for unnecessary features is not easy.

Today, we will explain Amazon’s many declinations for database services in the Cloud to give you an overview of what and when to choose a service.

Engines: Amazon RDS vs Amazon Aurora

The first confusing factor when approaching managed database services on AWS is the difference between Amazon RDS (for MsSql, MySQL, PostgreSQL…, etc.) and Amazon Aurora (MySQL or PostgreSQL).

RDS for… (choose your favorite engine)

When you choose to deploy an RDS instance, behind the scenes, AWS will deploy and configure a compute instance containing your engine of choice with the corresponding supported version.

You can select the compute and storage size, with an option to automatically scale the storage until it reaches your chosen upper limit. Operating system and minor engine patches are applied during a maintenance window.

You will have to choose when to upgrade for major engine updates. Be aware: if you fall behind the currently supported version matrix, you will incur additional charges because you will enter the "Extended support phase."

Caveats:

Here's a practical example: if you chose to deploy an Amazon RDS for MySQL instance in October 2022, you received all minor upgrades and entered extended support on February 29, 2024.

Be also aware that if you reach the upper limit for the storage allocated (even with autoscaling), you will need to resize the storage and experience downtime. Also, you can't easily scale down storage, so a blue/green deployment will be required for this kind of operation.

Amazon RDS Custom

RDS Custom may be for you if you want more control over the underlying operating system because you have legacy applications and a Microsoft SQL Server or Oracle database. We will not cover its architecture in-depth, but it is an option for this specific need. We recommend using managed services because you can reduce the operational effort and focus on business. You can find more information on the shared responsibility model for Amazon RDS Custom here.

Amazon Aurora

Amazon Aurora is a managed service that decouples the database engine and the underlying storage layer. By adapting this technique, you can worry less about additional aspects, such as storage replicas and scaling. As for the "traditional" RDS, you have to choose the computational size, and you will pay for what you use.

If you have ever deployed an Amazon Aurora RDS instance, you may have noted that the AWS Console doesn't ask for a size. This is because additional features have been added, so storage autoscaling is granted. Storage is also automatically replicated across three

Availability Zones, with each Availability Zone persisting two copies, tolerating the loss of up to two copies of data.

In the case of an instance failure, Amazon Aurora RDS will try to self-heal itself by restarting the relevant services.

Caveats:

Be aware that a single instance will try to self-heal, but downtime will occur. Additionally, Aurora engine versions are slightly different and don't follow the release lifecycle of the standard engines.

If you use a particular set of features (such as MyISAM for MySQL), check if they are supported.

Aurora Serverless

Aurora serverless pushes the decoupling further by containerizing the computational part to ease scaling.

When you deploy Aurora in serverless mode, instead of choosing the bare CPU and Memory size, you have to select your minimum and maximum size in terms of Aurora Capacity Units (ACUs).

An ACU measures resources in terms of RAM, CPU, and Networking; each ACU gives you 2 GB of RAM and a proportional CPU value. Working with ACUs allows you to scale "on-the-fly" the computational power assigned to your database.

You can specify the minimum and maximum values (from 0.5 to 256 ACUs) and even automatically pause your database, so no charges will incur during inactive periods (think about a development cluster or a workload used only during office hours).

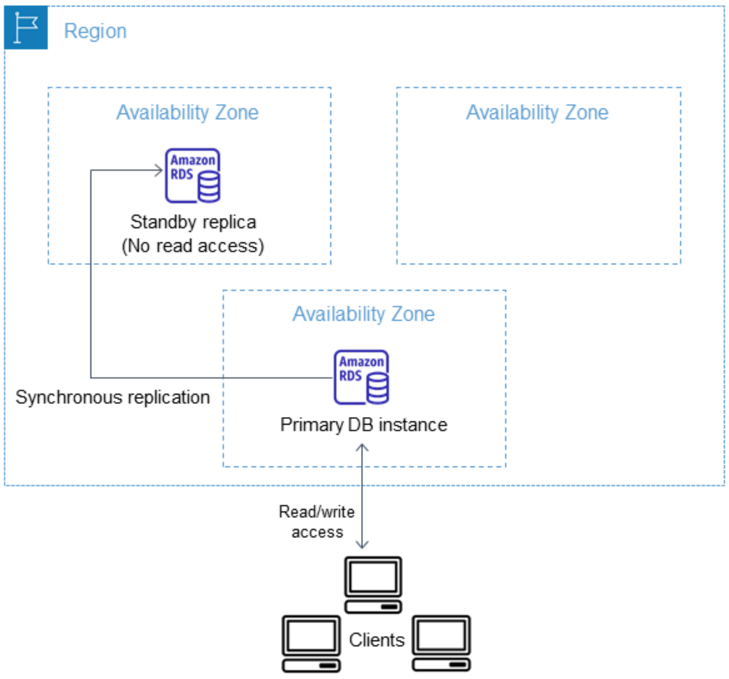
Caveats:

Be aware that, like "traditional" RDS instances, the number of simultaneous connections is determined by the size of the RAM memory assigned to the cluster. When you configure a serverless instance, the number of simultaneous connections is determined by the upper limit of ACUs. If you need more connections, you must change the maximum number of ACUs and reboot the instance to apply changes.

High Availability: Multi-AZ, read-replicas, let's clarify!

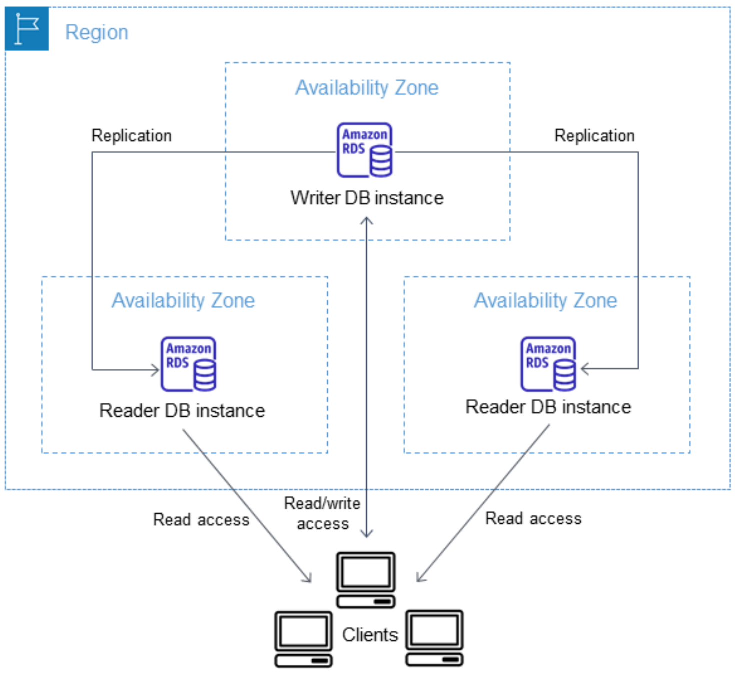
Even if the documentation clearly explains the features, we found that many people are always confused about Multi-AZ RDS Instances, Multi-AZ cluster deployments, and Aurora read replicas. Let's tackle this issue!

First, a **Multi-AZ Amazon RDS Instance** is a synchronous copy from a primary instance to a standby replica. **You can't use the standby instance to read data or offload reads**; it is only used as a failover option in case of issues or maintenance. In case of failover, the recovery occurs in 60 - 120 seconds.



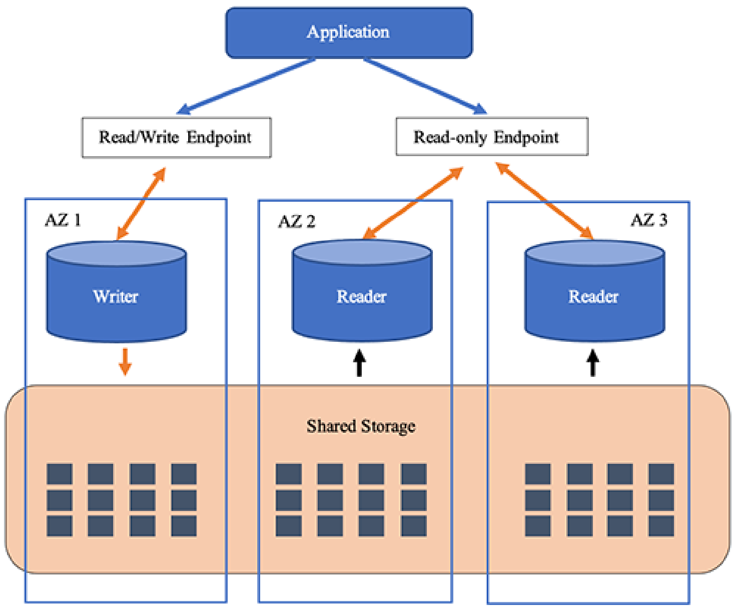
Source: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZSingleStandby.html

You can create **Multi-AZ Amazon RDS Clusters**. In this case, **replication is semi-synchronous** using the native database engine replication, and you can have **up to two accessible read-only instances** to offload reads. However, with this configuration, there could be a replication lag, but the write latency is lower than in a Multi-AZ instance. Failovers are typically completed in under 35 seconds.



Source: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/multi-az-db-clusters-concepts.html#multi-az-db-clusters-concepts-overview

As we said before, data is automatically replicated across three availability zones when you are using the **Aurora** engine. You can add up to 15 read replicas to scale reads. In case of a failure, failover occurs in under 30 seconds). Be aware that when using a multi-az Aurora Cluster, your application has to use a cluster endpoint managed and updated by Aurora.



Source: https://aws.amazon.com/blogs/database/improve-application-availability-on-amazon-aurora/

Tips and tricks to decrease failover time

Amazon RDS Proxy can decrease failover time in every case because it manages connection pooling between applications and your database architecture of choice.

In addition, you can reduce the number of maximum database connections needed, thus alleviating spikes in serverless applications.

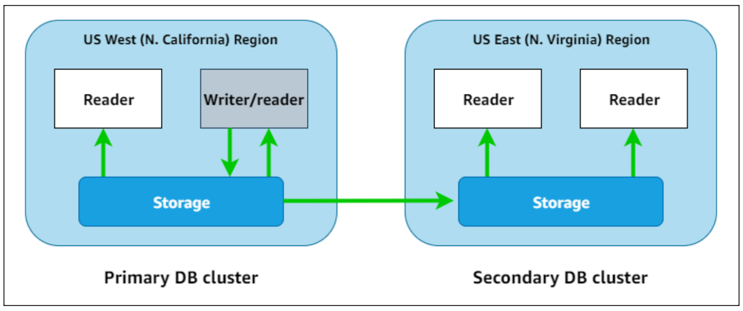
Be aware that Amazon RDS Proxy is a service that can impact your billing!

Global availability and scalability: Amazon Aurora Global Database vs. Amazon Aurora Postgres Limitless vs. Amazon Aurora Distributed SQL

Let's now discover Aurora's distributed flavors. These implementations focus on different aspects, but they may sound the same.

Amazon Aurora Global Database

Amazon Aurora Global Database focuses on **resilience**: it takes advantage of the independence between storage and compute layers to replicate data in a different region. If you configure a Global Database instance, you'll end up with a primary cluster (containing a reader and a writer instance) and a secondary cluster containing only read replicas. **Write forwarding** is supported, so if you perform a write using the secondary cluster endpoint, Aurora will take care of that by forwarding it to the primary DB cluster. If your application is latency-sensitive, be aware that when long distances are involved, the speed of light is a physical bottleneck that can't be avoided at the time of writing this article!



https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-global-database.html

Switchover, Managed Failover, and Manual Failover: the differences

When it comes to **resiliency**, multiple mechanisms allow tests, schedule maintenance, and, of course, a safety net in case problems happen.

If you perform a **switchover**, everything will be under control, and data will be entirely replicated to the secondary cluster. Hence, the Recovery Point Objective (RPO) is zero without experiencing data loss. This operation helps execute planned maintenance, recovery tests, regional rotation, or failing back from a disaster scenario to the original region.

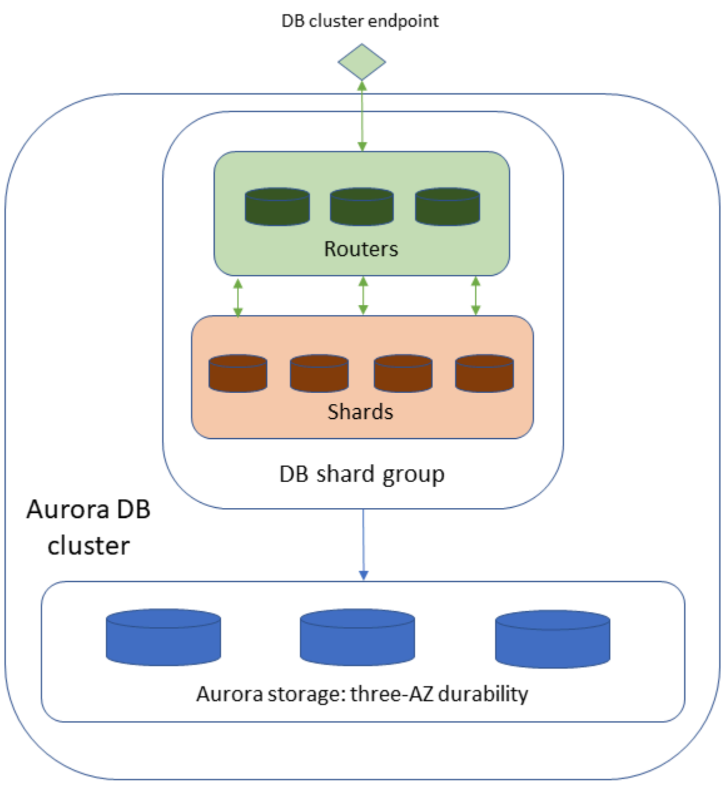
The second option is a **managed failover**. In a region or service-level outage, a managed failover will help maintain business continuity by switching the primary and secondary clusters without waiting for data synchronization. Data loss will happen, but no changes will be made to the replication topology (except for the cluster role switching).

A **manual failover** is a critical operation; you must be sure and understand its implications. It will break the existing cluster replication by making a standalone cluster and promoting a read replica to a writer. If you need to fail back, you will have to add a secondary region and rebuild the replication topology; no automatic failback options are available.
Bonus tip: if your RTO is not near zero, you can shrink your bill by taking advantage of the **headless configuration**: deploy a cluster in a secondary region and terminate the newly created Aurora instance. Data replication will not stop, and if you start a new instance in the secondary cluster, it will spin up without issues.

Amazon Aurora PostgreSQL Limitless Database

As the name suggests, this implementation is currently available only for PostgreSQL engine compatibility. It focuses on **scalability** by using sharding to distribute data between nodes, allowing you to scale horizontally for millions of write transactions and petabytes of data.

Its architecture slightly differs from the others because when you shard data, you need to route queries to the correct nodes that can query data. Router nodes can receive queries and send them to the proper shard owner.



Source: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/limitless-architecture.html

This scalability doesn't come without a price on the operational effort: you need to know how to partition data correctly, and since data is partitioned, you don't want a "hot shard" hit by more queries than the others. You will also need to manage shard nodes and track their load using metrics in CloudWatch.

To accommodate the load, you can change the capacity of shard nodes, split shards into more shard groups (but be aware that you cannot merge shards), and add routers.

Caveats

As the name says, this database implementation does not support MySQL compatibility. In addition, Amazon Aurora PostgreSQL Limitless isn't available in all regions, and you can't modify shard keys (including their value in table rows); you need to delete and re-create them. You find the complete list of limitations here.

Lastly, check for DDL and DML limitations before architecting your workload.

Amazon Aurora Distributed SQL

With the recent announcement, Amazon Aurora Distributed SQL (DSQL) focuses on **resilience and scalability**. It's a highly available multi-region active-active cluster that automatically scales and manages infrastructure.

While it combines the features of the Global Clusters and Limitless flavors, it has many limitations at the time of writing this article. It is currently in preview only in selected regions. It will become more feature-rich in the future, and for some use cases, it could be enough to grant your application scalability and resiliency without additional operational effort.

Aurora Distributed SQL limitations

At the time of writing, only PostgreSQL is available, and there are limitations on objects and operations. You can have a single database per cluster without views, triggers, temporary tables, or sequences. Foreign keys aren't supported, and you cannot perform a TRUNCATE, VACUUM, or create functions using plpgsql (or any other language than standard SQL).

While it supports ACID operations, transactions cannot have mixed DDL and DML operations, and they contain at most one DDL statement.

TL;DR

- Amazon RDS and Amazon Aurora have different architectures and flavors. You can choose between a more "traditional" approach by using Amazon RDS (including RDS custom for corner scenarios) or leveraging the storage and compute decoupling offered by Amazon Aurora.
- In terms of high availability, Amazon RDS offers Multi-AZ for synchronous replication to a standby instance and Multi-AZ Clusters with up to two readable standbys. Aurora automatically replicates data across three Availability Zones and allows up to 15 read replicas.
- Aurora Global Database replicates data across regions for global availability and scalability. Aurora PostgreSQL Limitless uses sharding for horizontal scaling, and Aurora Distributed SQL combines multi-region active-active clusters with automatic scaling.
- There are limitations for global architectures and horizontal scaling; check the article for a more comprehensive list!

What kind of architecture are you using for your database? Are there any special needs you need to address? Let us know in the comments!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!
