AWS RDS, Aurora, ElastiCache

AWS RDS

Relational database service. It is a managed database service and provide you with databases that uses SQL as the query language and AWS manages it for you.

Database that you can create is Postgres, MySQL, MariaDB, Oracle, Microsoft SQL Server, and AWS Aurora (AWS's proprietary database).

Why RDS over Database on EC2

Well RDS is managed for you so you don't have to deploy them yourself with an EC2 instance. There is automatic continuous backup and restoration capability.

Monitoring dashboard is provided, have read replicas for improved read performance. Multi AZ setup for disaster recovery, and maintenance windows for upgrades.

You can scale vertically by increasing the size and horizontally by increasing the number of instances. The storage is backed by EBS (gp2 or io1).

However, you can't SSH into the underlying EC2 instance can only interact with the database via a database client.

Storage auto scaling

When you first provision your RDS you will specify the initial capacity, and with this feature RDS will detect when you are running out of database storage and scales it automatically.

You will have to set a Maximum storage threshold (the maximum limit that you are willing storage auto scaling to allocate at max for you).

Then it will automatically modify the size of the storage if the amount of free space is < 10%, and the low-storage last at least 5 minutes, and 6 hours has passed since last modification.

This feature is good if you can't predict the amount of storage you will be taking up.

Read replicas

Read replicas is used for read scalability.

Your application will communicate with a RDS database instance and most of the time it is read requests, lots of them actually. Your database might be overwhelmed by those requests and therefore you can set up read replica, which is a copy of the main database instance but can only take read requests. They cannot take write operations since they are a replication of the original.

Read replicas can be setup in the same AZ, cross AZ, or even cross region.

The way that the replicas sync up the data is via **asynchronous synchronization** meaning that the reads will eventually become consistent. The replicas will catch up when there is free time, which means if your application reads from the replicas there is a possibility that it is reading old data, but it will eventually be synced up with the latest changes.

Replicas can be promoted to its own database giving it the capability of writing.

Use cases for read replicas

The good use cases for that is for example if you are going to run some analytics on the production database instances. If you are going to query directly into the production database it might overload it since the production application is using it as well.

What you should do is that you make a replica of the production database then your analytic application can query into the replica instead of the prod database that way the **production application will not be slowed down/affected.**

Network cost

Normally there is a network cost when data goes from one AZ to another, but are exceptions depending on the services for example ALB for cross zone load balancing.

For RDS read replicas within the same region, you don't have to pay cross AZ network fee. For example, you set up your main RDS database instance in us-east-1a, but you set up your read replica in us-east-1b, normally because the asynchronous synchronization require data to cross AZ, **in this case you don't have to pay the fee if it is in the same region.**

However, if your replica is in another region for example instead of us-east-1b it is eu-west-1b, then it will cost you network fee.

Multi AZ

This is for disaster recovery mainly.

You have your application that talks to the main database instance like usual, then you set up another RDS DB as a standby in another AZ and data is going to be replicated **SYNCHRONOUSLY**, meaning every write on the master database will be also pushed to the standby instance as well.

Then with multi AZ database set up you will get one DNS name which behind it has two instances of RDS database, one that's the master and the other as standby, **if the master database failed due to AZ failure, the DNS name will automatically switch over to the standby instance.**

This increases overall availability, and this is completely transparent to the application because they just need to talk to the DNS name that it is fronted with.

This is not for scaling since the standby instance is just for backup. **You can also set up your read replica as multi AZ for disaster recovery that is possible this is possible.**

From single AZ to multi AZ

To change it from single AZ to multi AZ there is no need to stop the database. Just need to modify it.

In the background, a snapshot of the original database will be taken and a new database instance will be restore from the snapshot. Then there will be synchronization established between the two database for multi AZ setup.

RDS Custom

This is only for managed oracle and Microsoft SQL server databases. it lets you access the underlying EC2 instances managed by AWS for the RDS.

This is so that you can configure settings, install patches, enable native features, and SSH into the EC2 instances that you normally don't get to see when you create an RDS instance.

You should disable automation mode when you are performing customization on the underlying EC2 instance and take snapshot just in case you wreck your database.

Amazon Aurora

AWS's proprietary database. But it is compatible with Postgres and MySQL, connecting to an Aurora DB you can query it like it is Postgres or MySQL database.

They claim it is cloud optimized, more performance compared to MySQL on RDS.

Aurora storage automatically grow in increments of 10GB, up to 128TB. So you don't even need to provision the initial disk space.

You can have up to 15 replicas compared to 5 for MySQL. Failover in Aurora is instantaneous and is highly available by default. But it cost more.

High availability and read scaling

Every time you write to the database Aurora will store 6 copies of your data across 3 AZ. Meaning every time you write to the database it will store 2 copy of the same data in one of each AZ. **The underlying storage are shared across the AZ!**

It needs 4 copies out of 6 needed for write operations and 3 copies out of 6 needed for reading operation.

Self healing with peer-to-peer replication so if corruption occurs in one database it will be automatically corrected.

Only master instance of Aurora instance takes writes, and automated failover for master in less than 30 seconds. You can have up to 15 replicas + master for your database.

It also support cross region replication.

Data replication is in millisecond which is extremely fast.

Aurora database cluster

Writer endpoint: Points to the master instance for writing operations. The client will talk to this endpoint in order to invoke write operations.

Reader endpoint: Like a load balancer because if you are going to auto scale how do you keep track of the instances of read replica that is part of the auto scaling? Easy it is done by the reader endpoint, it will automatically **point to all of read replicas and those that are added due to auto scaling and load balance read requests among all of them.**

Then the client will communicate with the reader endpoint in order to read from the database.

Aurora custom endpoints

This allows you to define a subset of Aurora instances into a custom endpoint. So with each Aurora instance there is a EC2 instance behind it, you can define a subset of them to be a larger instance to say handle analytic work on the database under a custom endpoint. Then normal reader endpoint under another custom endpoint.

If you define your own custom endpoint then the reader endpoint will not be used after, you would have to define your own reader endpoint to point to those instances that you would still like to use for reading.

Aurora serverless

Automate database instantiation and auto scaling based on actual usage. Like lambda functions.

Good for infrequent, intermittent or unpredictable workloads.

No capacity planning needed and is pay per second which can be more cost-effective.

You will set up a proxy fleet which is managed by Aurora and it will instantiate the Aurora RDS based on the incoming requests.

Aurora multi-master

In case you want immediate failover for writer node. Let every node does R/W vs. promoting a read replica to be the new master (this is default behavior).

Global Aurora

Aurora cross region read replicas: Allow cross region read replica which is useful for disaster recovery.

Aurora global database: You set one primary region for read/write, then up to 5 secondary read only region, each of those region can have up to 16 read replicas. **help you decrease latency for data access all over the world.**

Have recovery time objective of < 1 minute, how long it will take to recover by promoting another region.

Typical cross-region replication takes less than 1 second. Hint to use global Aurora

Aurora machine learning

Let you add machine learning based prediction to your application via SQL interface to do things like fraud detection, ads targeting, sentiment analysis, product recommendation all in Aurora.

Amazon SageMaker and Amazon Comprehend is supported.

RDS backup

- Automated backup: daily full backup of the database. Transaction logs are backed-up by RDS every 5 minutes, which means you can go back to 5 minutes ago. The backup are kept 1 - 35 days.
- Manual database snapshots: Manually triggered by user, but you can keep the backup for as long as you want

If you are going to stop a RDS database, which you will still pay for storage. To save cost, you do a snapshot and restore it which will cost way less. Then just restore the database from the snapshot once you are ready to use it again.

Aurora backup

- Automated backup: 1 to 35 days of retention you cannot disable it. Point-in-time recovery in that timeframe, meaning you can restore to data up to 35 days ago.
- Manual database snapshot: Same triggered by user, keep backup as long as you want

Restore options

Restoring RDS/Aurora backup or a snapshot creates a new database.

You can also migrate on-premise database to RDS database by making a backup of your on-premise database and put it in S3 bucket, then create a new RDS from it.

To migrate to Aurora cluster from on-premise database you would need to use service call Percona XtraBackup and store the backup file into S3 and create Aurora from it.

Aurora database cloning

Create a new Aurora database cluster from an existing one.

Basically let you clone an existing Aurora, this is faster than snapshotting it and restoring it.

The cloned database initially uses the shared volume but as writes are made then it will create new shared volume, this is copy on write optimization.

RDS & Aurora security

You can have at-rest encryption: require master database and replicas encryption using KMS

If master is not encrypted then read replicas can't be encrypted. Again to encrypt an unencrypted database you have to go through snapshot and restore as encrypted

You can also have in-flight encryption: It is ready by default, data that are in transit are encrypted.

IAM Authentication: You can use IAM roles to connect to your databases instead of username and password.

You can allow or block specific ports / IP using security groups.

No SSHing except RDS custom. These are the security options for RDS / Aurora

RDS proxy

A database proxy for RDS.

The proxy allow apps to pool and share DB connection established with the database. This allows you to improve database efficiency by reducing the number of open connections. Save CPU and

RAM resources.

Definitely needed for Lambda functions that need database access. This is because lambda function appear and disappear really quickly, if every lambda function open up a database connection and close it that quickly then it will overwhelm the RDS / Aurora database very fast, **this is why a proxy is needed to pool and share the connection.**

It is serverless, autoscaling, high available through multi AZ. Can reduce failover time by up 66%.

Supports RDS and Aurora, no code change is needed. It also enforce IAM authentication for database instead of username/password.

The proxy is only available in the VPC. Cannot be access publicly.

Amazon ElastiCache

RDS is a managed relational database.

ElastiCache is a managed Redis or Memcached. Caches are basically memory that are high performance and have low latency, you use them to store frequently accessed data so that you don't have to query the data in slower disks.

Help reduce load off of databases for read intensive workloads, if they exist in cache then no need to hit the disk which cost even more time. Also make your application stateless.

Using ElastiCache need heavy code changes!

Use ElastiCache as database Cache

Your application will first query the ElastiCache if it exist then is cache hit, no need to hit up RDS. If cache miss then you will query RDS and then store it in ElastiCache.

Help relief load off from RDS from same request over and over again.

Use ElastiCache as user session store

User will log into your application, your application store the session data into ElastiCache instead of database. Then you can retrieve session data from the ElastiCache making the user log in already

Redis vs Memcached

Both are in-memory data structure server. Both utilizes key/value store.

Redis allow multi AZ with auto-failover, can have read replicas to scale and is highly available. There is backup and restore feature. Redis is persistent by default.

Memcached however don't have replication thus not highly availability, you can't backup and restore, and is non persistent. Memcached is not persistent, if you restart the service you will lose your data!

ElastiCache security

Uses IAM authentication for Redis only. The rest just use username and password.

Redis AUTH: Set password/token when you create a Redis cluster, extra layer of security. Gain in-flight encryption.

Memcached supports SASL based authentication.

Pattern for ElastiCache

- Lazy loading: All the read data is cached, data can become stale in cache. Cache data that you don't have this is simple
- Write through: Add or update data in the cache when writing to a database, more complicated.
- Session store: Store temporary session data in a cache

Redis Use Case

Gaming leaderboards are computationally complex. Redis sorted set give you that ranking in uniquenesses and element ordering. Each time a new element is added, it will be reordered in real time.

So you can use Redis as a game leaderboard

Revision #3
Created 2023-02-12 18:41:16 UTC by Tamarine
Updated 2023-02-13 01:28:57 UTC by Tamarine