



COSC 416: Topics in Databases (DBaaS)

TOPIC 6: RDS CLUSTERS AND REPLICATION



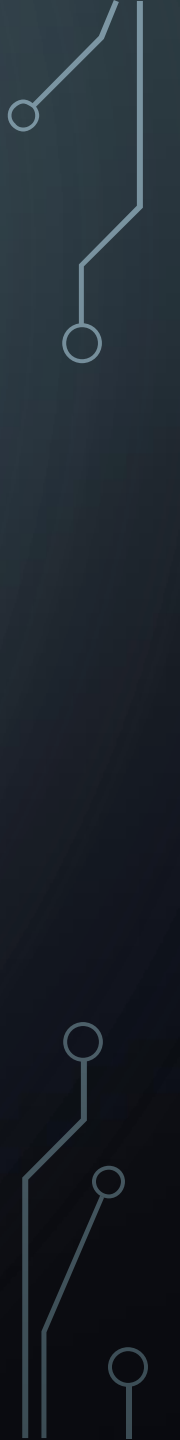
SCHEDULE

1. Advanced Replication

1. Multi-AZ deployment

2. Promoting replicas

2. Amazon Aurora



The background is a dark blue gradient. In the corners, there are white line-art illustrations of circuit boards or neural networks, with lines connecting to small circles.

ADVANCED REPLICATION


RECAP – READ REPLICAS

- We've already talked a bit about read replicas in the Topic 4 notes
- So we can manually create read-only copies of our database, and we can use these read-only replicas to help spread the query load
- But read-only replicas aren't the only way replicas can be used in RDS

UNDERSTANDING AVAILABILITY ZONES

- First, we need to understand the idea of an *availability zone* in AWS, often simply called an AZ

DB identifier	Role	Engine	Region & AZ
database-1	Instance	MySQL Community	us-east-2a



- Remember that single letter after the region? That's the availability zone of the instance

SO WHAT IS AN AZ?

- An availability zone is a subcategory of region
- A single region may comprise of multiple datacenters, with each being it's own AZ within the region
- You may pick an AZ, or have one assigned automatically by Amazon

WHY IS THIS IMPORTANT?

- Consider that through catastrophic failure (say, localized flooding, or a fire), a datacenter may end up offline
- This would result in all instances in that availability zone being unreachable
- But what if we had replicas across availability zones for just this purpose?

MULTI-AZ DEPLOYMENT

Multi-AZ deployment

Specifies if the DB instance should have a standby deployed in another availability zone.

☐ Yes

☒ No

- Amazon offers this ability through what it calls “*Multi-AZ deployment*”
- This option can be selected when creating a database instance, or when modifying a pre-existing instance

WHAT MULTI-AZ DEPLOYMENT DOES

- When you create a multi-AZ deployment, Amazon creates a replica instance in a different availability zone
- This standby replica is kept up to date with the original primary database
- If the primary is detected as going offline, Amazon will perform an *automatic failover* – the standby automatically takes over for the failed primary

STANDBY REPLICATION

- Because of their intended function, multi-AZ standby replicas use a slightly different replication technique than read-only replicas
- While read-only replicas use asynchronous replication, standby replicas use *synchronous* replication
- Changes to the primary are pushed to the standby copy in-sync, so the standby is always fresh – a perfect replica of the primary
- This way, no data is lost due to changes that might not have been pushed to the standby yet.

AUTOMATIC FAILOVER

- When a failure occurs, or an AZ undergoes planned maintenance, RDS will automatically switch to using the standby instance
- It will automatically change the DNS records to point to the standby instance, so you maintain the same public endpoint
- However, since the DNS change occurs, you still have to re-establish your connection (otherwise you'll be using the old DNS information for the primary)
- Amazon suggests 60-120 seconds for the failover to complete, but they state it *could* be longer

FORCING THE FAILOVER

- We can also manually force a failover event by rebooting the instance using the AWS CLI, and the *--force-failover* flag

```
C:\Users\KoBoLd>aws2 rds reboot-db-instance --db-instance-identifier database-1 --force-failover
{
  "DBInstance": {
    "DBInstanceIdentifier": "database-1",
    "DBInstanceClass": "db.t2.micro",
    "Engine": "mysql",
    "DBInstanceStatus": "rebooting",
```

A PERFECT SOLUTION?

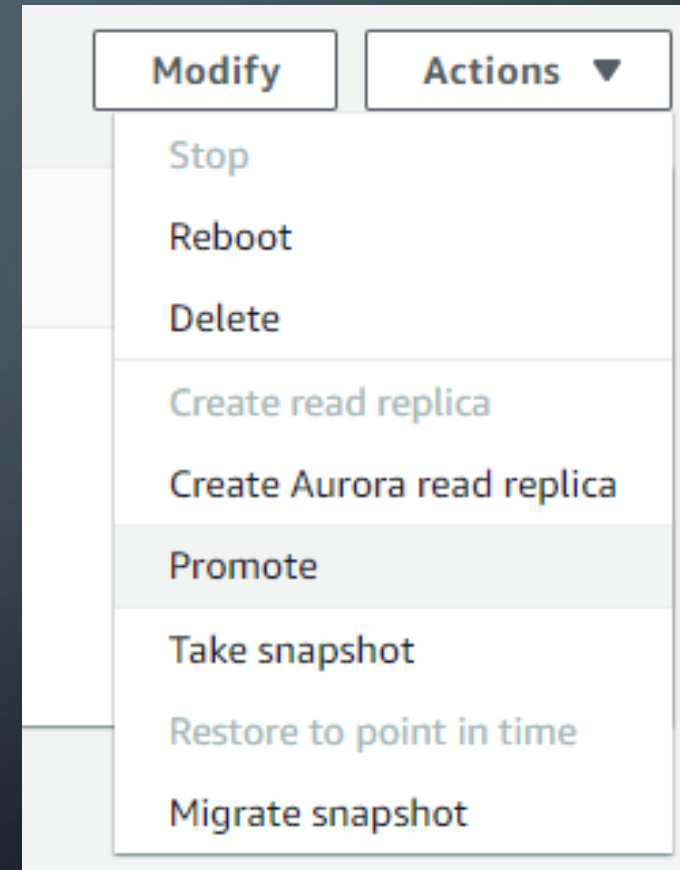
- While this is a pretty slick system, there are some things to keep in mind
- First of all, there's the cost – we are essentially keeping a duplicate instance, which Amazon is going to charge us for
- It can also hurt your database performance – synchronous replication means writes are going to be copied to the replica immediately, which eats up resources and forces the database to wait for the replica to be synced up

MULTI-AZ TAKEAWAY

- If you are providing a database service where downtime needs to be kept to an absolute minimum, multi-AZ deployment is a nice insurance policy
- The system is automated and hassle-free; you don't have to configure anything besides enabling the multi-AZ deployment option
- Downside: hurts performance and costs money
- Summary: Use when reliability and durability outweigh performance and cost considerations

PROMOTING READ REPLICAS

- Something we didn't talk about when we discussed read replicas was about promoting a replica
- This can be easily done through the drop-down actions menu



THE PROMOTION PROCESS

- When a replica is promoted, we have to follow a process to ensure our promoted replica is fresh
 - First, the origin instance is put into a read-only state
 - We then wait for all previous updates to the origin to be replicated onto the replica
 - Finally, we can perform the actual promotion

CHANGING THE READ_ONLY PARAMETER

- In the RDS GUI, you can view your database parameters under the “Parameter groups” page
- Keep in mind these parameters are applied to *all instances using that group*

Parameters

Q read_only

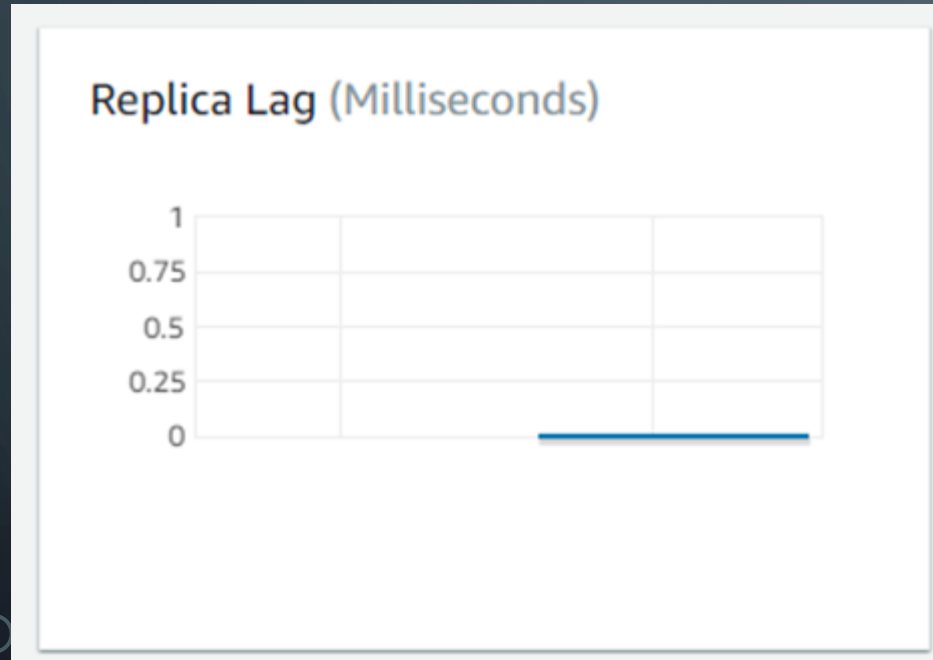
<input type="checkbox"/>	Name ▼	Values	Allowed values
<input type="checkbox"/>	innodb_read_only		0, 1
<input type="checkbox"/>	read_only	0 ▼	0, 1, {TruelfReplica}
<input type="checkbox"/>	super_read_only	0	0, 1
		1	
		{TruelfReplica}	

Recent events

CHANGING THE READ_ONLY PARAMETER

- You can edit the existing parameter group, and then search for the *read_only* parameter
- It has 3 values – 0 (false), 1 (true), and {TrueIfReplica}, which means only replicas will be read-only
- By setting this value to one, we can force our origin instance into read-only mode

MONITORING REPLICATION



- After we've set the origin instance to be read-only, we can keep track of when the replication is complete using the Replica Lag metric, which is available under the monitoring tab for the replica instance
- When replica lag is 0, the replica has caught up with the origin instance

PROMOTING THE REPLICA

- Once the replica has caught up, we can perform the actual promotion using the promotion action
- Once promoted, replication no longer occurs and the replica can function as a standalone database instance, separate from its origin instance
- Don't forget to set the `read_only` value back for your origin instance!

A FINAL INTERESTING ASIDE

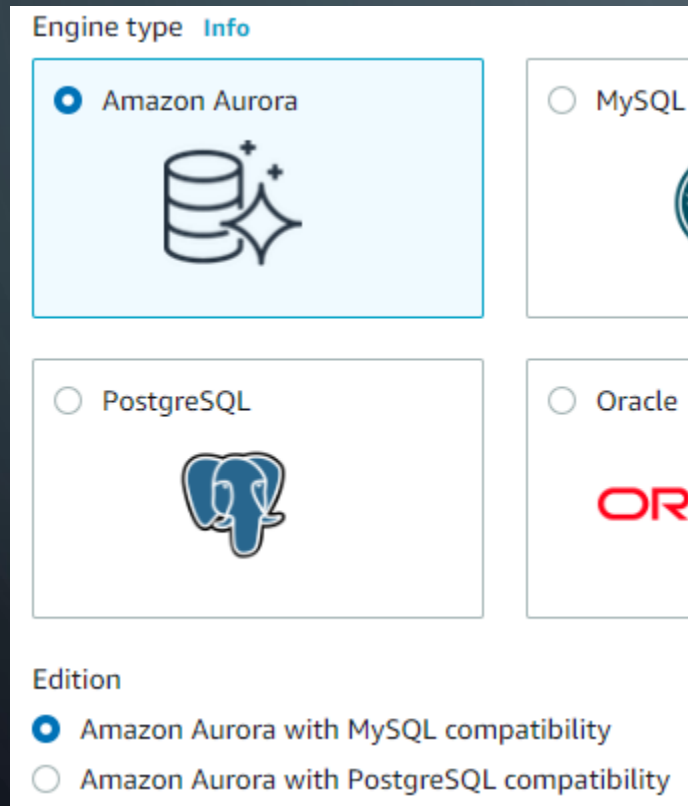
- One last interesting feature of read replicas is that they continue through an automatic failover event
- When an automatic failover occurs, the read-replica will begin asynchronously replicating from the standby instance, instead of the (offline) origin instance
- This means your read-replicas will stay online and still receive updates if the DB changes

RDS AURORA CLUSTERS

AMAZON AURORA

- Aurora is Amazon's in-house relational database engine
- Can be configured on creation to be compatible with either MySQL or PostgreSQL (including various versions of each)
- The primary purpose of Aurora is to facilitate database clusters with the RDS service

AMAZON AURORA



- Creating an Aurora database is straightforward, using the same interface as creating a MySQL instance
- Some additional options are available specifically for Aurora
- Unfortunately, doesn't support the t2.micro size, only t2.small

TYPES OF CLUSTERS

- Aurora with MySQL compatibility supports a number of different cluster configurations:
 - One writer, multiple readers
 - Multiple writers
 - Serverless
- Which type of cluster you create will depend on what your database workload is, and how you intend to use the cluster

ONE WRITER, MULTIPLE READERS

- A one writer, multiple readers cluster is essentially a single origin database instance and then asynchronously updated replicas
- The cluster provides two endpoints, one for writing and one for reading
- The reading endpoint use one of the Aurora read-replicas at random (providing simple load balancing)

MULTIPLE WRITERS

- In a multiple writer cluster, each instance is capable of performing writes to the database – there are no read-only nodes
- Changes on one writer are replicated to all other writers
- *Continual availability* – if a writer instance fails, there is no need for automatic failover – other writers automatically take up the slack

THE WRITE-CONFLICT PROBLEM

- A problem with the multiple-writer cluster is the possibility for a write-conflict
- The database is stored as fixed, 16KB data pages
- If two writers attempt to write to the same data page at once, this will result in an error, and the write will need to be attempted again when the data page is no longer in use
- Best to try to prevent operations on the same data from occurring at the same time across two different instances

AURORA SERVERLESS

- A serverless cluster with Aurora is similar to one writer/multiple readers, but designed for applications where usage may be variable
- Rather than defining an instance size (like t2.small), you can specify upper and lower bounds, and the cluster will expand or contract the computing resources used to match usage

WHEN DO AURORA CLUSTERS MAKE SENSE?

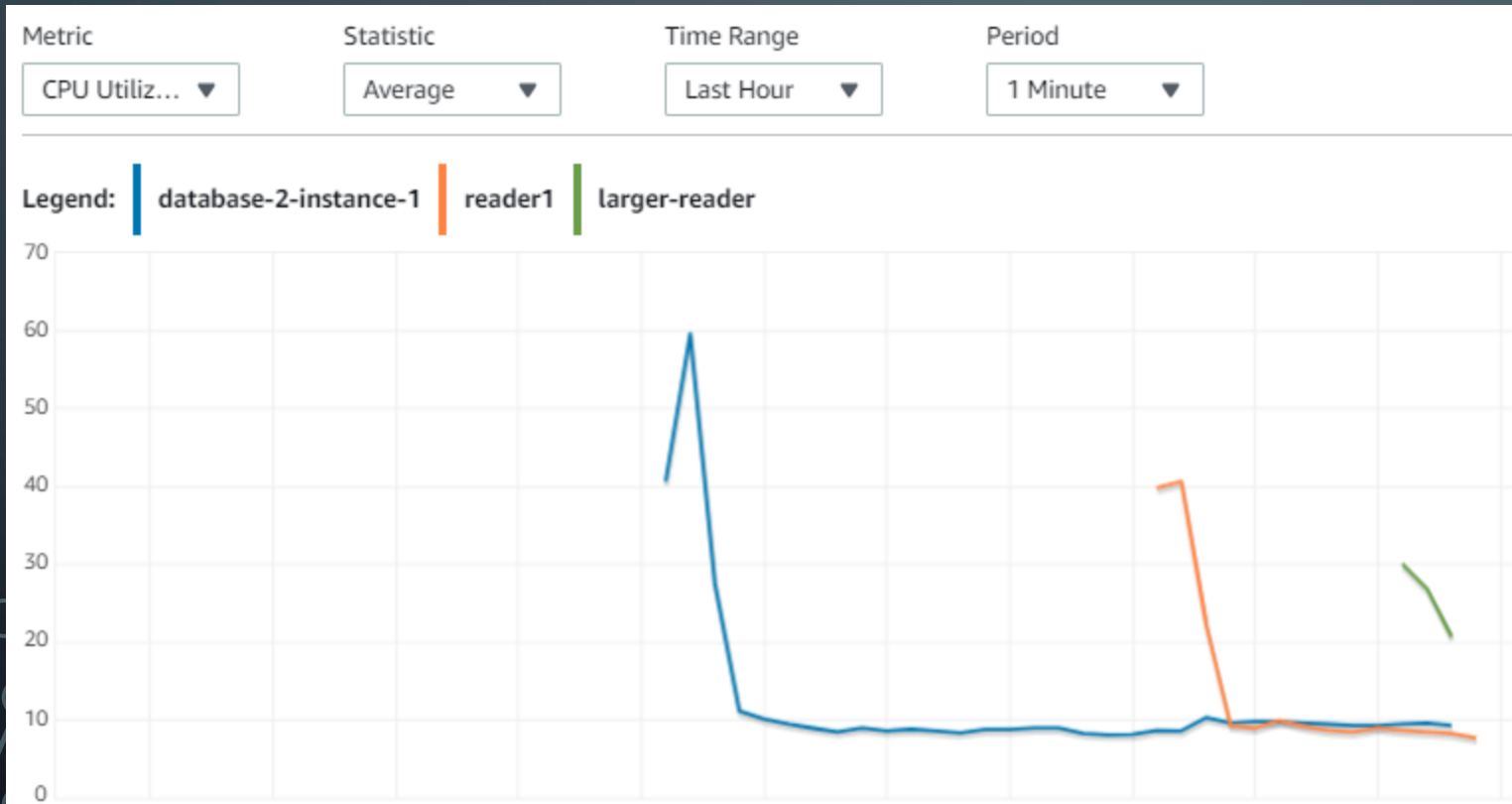
- Aurora clusters offer a fairly simple, managed cluster architecture that allows you to balance load across multiple database instances without too much work on your part
- Multiple-Writers pose some additional programming challenges to prevent write-deadlocks from occurring
- If you just want to ensure uptime and spread the workload across multiple instances, Aurora clusters are a good choice

CLUSTER INSTANCE SIZING & AZ

DB identifier ▲	Role ▼	Engine ▼	Region & AZ ▼	Size ▼
database-2	Regional	Aurora MySQL	us-east-2	3 instances
database-2-instance-1	Writer	Aurora MySQL	us-east-2a	db.t2.small
larger-reader	Reader	Aurora MySQL	us-east-2c	db.t2.medium
reader1	Reader	Aurora MySQL	us-east-2b	db.t2.small

- There is no requirement that clusters contain the same size instances – you can have a larger or smaller writer or reader instance, if needed (although Amazon will complain a bit)
- Notice also that Amazon has placed each node automatically into a different AZ → helps prevent outages if a single AZ goes down

MONITORING YOUR CLUSTER



- The monitoring dashboard for the cluster will allow you to visualize all the nodes in the cluster

AURORA TAKEAWAY

- Ultimately, Aurora is basically just MySQL (or PostgreSQL) with some pre-built options for handling replication and load balancing
- It's great if you just need something that works, and don't have specific requirements for your cluster beyond "it's a database"
- Cluster configuration is pretty simple and you can easily customize it to meet most workload demands, within reason

THE PRICE

- You're going to be paying the same amount for Aurora instances as you do for regular instances
- Since it's only available for instances larger than t2.micro, it's going to cost some money – the more nodes in your cluster, the more \$\$\$ per hour it's going to cost
- For your reference – a single t2.small works out to about \$0.03-\$0.04 cents per hour
- A cluster of 10 t2.small instances → roughly \$237 per month



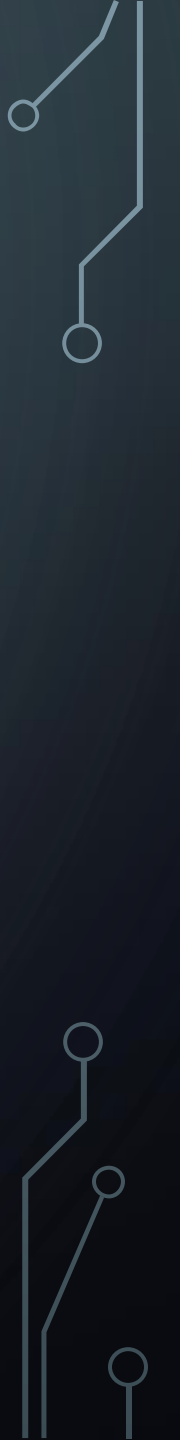
SCHEDULE

1. Advanced Replication

1. Multi-AZ deployment

2. Promoting replicas

2. Amazon Aurora



The image features a dark blue gradient background. In the corners, there are white line-art illustrations of circuit boards or neural networks, with lines and small circles representing components. The text "SO LONG, FOLKS!" is centered in a white, bold, sans-serif font.

SO LONG, FOLKS!