Mi Gao

migao@bu.edu

Term project  
The experience of Cloud-based database

MET CS 779

MET CS 779 Term Project

Table of Contents

[1. Introduction 2](#_Toc48720745)

[2. PostgreSQL replication based on Standby 2](#_Toc48720746)

[1) The master database configuration 4](#_Toc48720747)

[2) The slave database configuration 5](#_Toc48720748)

[3) Testing results 6](#_Toc48720749)

[3. Amazon RDS 8](#_Toc48720750)

[1) Deploy the database 9](#_Toc48720751)

[2) Read Replica 17](#_Toc48720752)

[4. Amazon Aurora 23](#_Toc48720753)

[5. Conclusion 25](#_Toc48720754)

[5. Revision History 26](#_Toc48720755)

[Bibliography 27](#_Toc48720756)

# Introduction

Assuming the business has developed and grown very fast in the past few years. As the example of Netflix database ERD (Figure 1.1) had shown, the amount of data and business became more and more complex and diverse. In order to be close to end users and to improve disaster tolerance, the company may deploy databases worldwide such as China, the United States, and Europe. And keep the data consistent which can make one-write and multiple-reads. The cloud-based database will eventually become necessary. Because of this consideration, I will conduct research on the features of cloud-based database about PostgreSQL and AWS RDS service based on the Netflix database provided in this semester. The main topic would be to focus on its concept of replication.

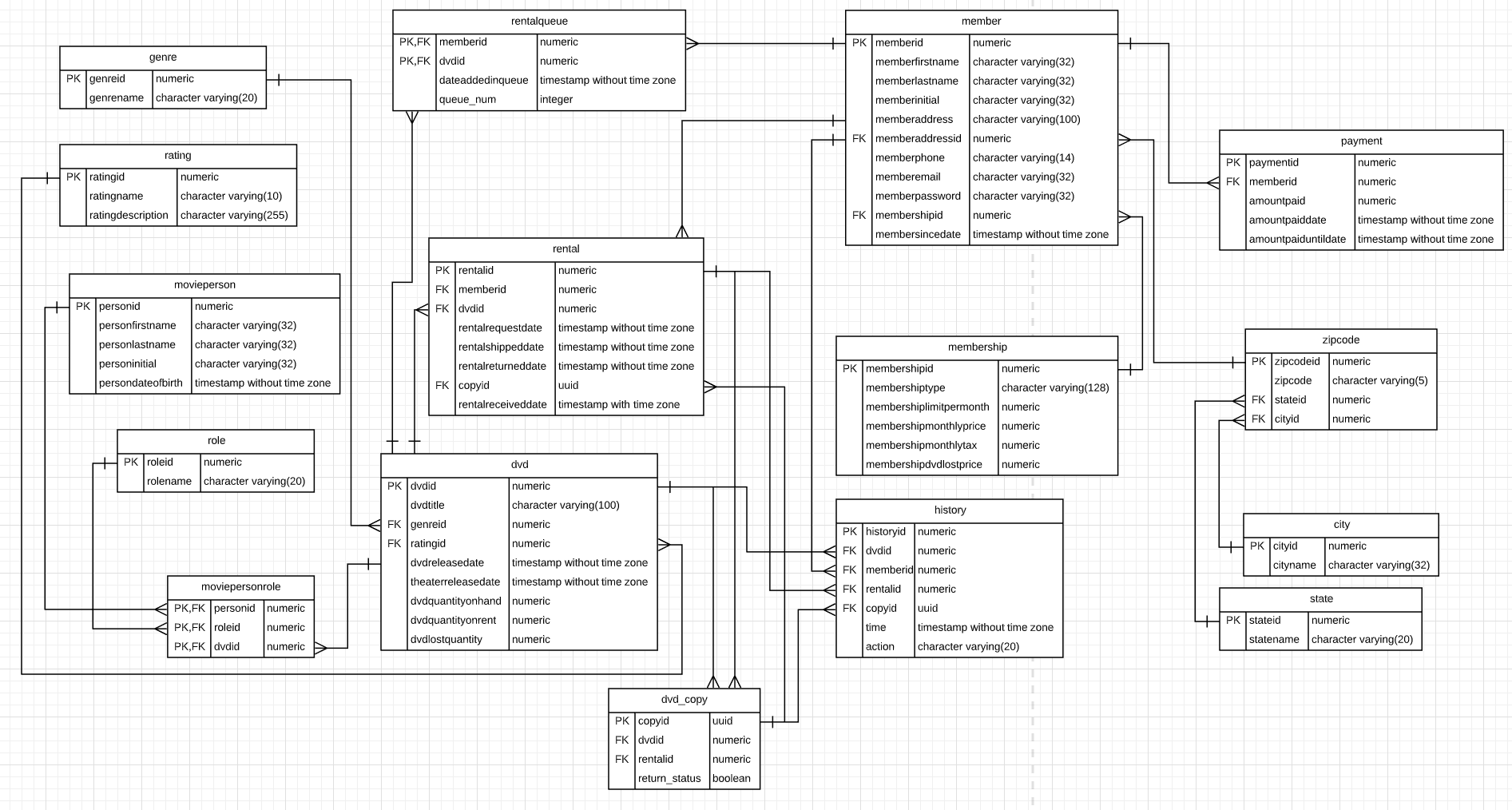


Figure 1.1

# PostgreSQL replication based on Standby

First of all, the replication is one of the most important and attractive features for cloud-based database. Manual implementation of the replication of the database was complex to implement in the past before those famous network company did not launch their cloud services yet.

PostgreSQL has launched a log shipping similar to Oracle's active dataguard and MySQL relay log after version 9.0. With this feature, we can achieve master-slave replication of PostgreSQL. The basic principle was that one master database provides read and write usually, and then synchronizes the data to another slave database. The slave database continuously applies the data received from the master database. The slave database does not provide write services, but only provides external read services. The server that provides full read and write functions in PostgreSQL is called primary database or master database, and the slave database server that can provide read services while receiving synchronization data from the primary database is called hot standby server.

PostgreSQL maintains a WAL log file in the pg\_xlog subdirectory of the data directory. This file is used to record each change of the database file. This log file mechanism provides a database hot backup scheme, that is, when the database is used When the file system is backed up, the corresponding WAL log is also backed up. Even if the data blocks backed up are inconsistent, the WAL log can be replayed to push the backup content to a consistent state. This is also known as Point-in-Time Recovery, or PITR for short. There are two ways to transfer WAL logs to another server, which:

1. WAL log archive (base-file)
2. Streaming replication (PostgreSQL, 2020)

The first type was to copy the WAL log file to the standby database after writing a WAL log. In short, the remote backup is realized through the cp command, so that the standby database usually lags behind the main database by one WAL log file. The second type which called streaming replication was a new method of transferring WAL logs provided after PostgreSQL 9.x. Its advantage is that as soon as the master library generates a log, it will be immediately transferred to the standby library, which is lower than the first synchronization delay, so we will definitely choose streaming replication.

Before the actual operation, there is one more point that needs to be explained is the most critical step in the establishment of the standby-the basic backup of the master is generated in the standby. After postgresql9.1, a very convenient tool-pg\_basebackup is provided. The detailed introduction and parameter description can be viewed on the official website (Documentation, 2020).

Learned much about how to manually set up the replication after taking the look at tutorial article at Alibaba Cloud Community (科技小能手, 2017). Based on the cost and necessary requirements for creating PostgreSQL asynchronous streaming master-slave replication, I chose to use free trial of virtual machine from Alibaba Cloud (Alibaba Cloud, 2020) although it charged me because of my mistakes of additional settings. There is the environment of the rental virtual machine for the example testing:

System: CentOS release 6.6.

PostgreSQL 9.6.6.

SELINUX=disabled

Iptables closed

Ips:

Master database: 192.168.221.161

Slave database: 192.168.221.160

## The master database configuration

* 1. Add a synchronized username and password in the main database:

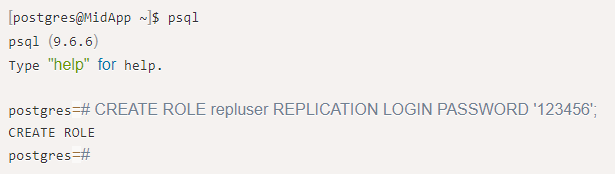


Figure 2.1

* 1. Modify “/home/postgres/pgsql/data/pg\_hba.conf” and add the last line:



Figure 2.2

Notice that this line of configuration means that the user repluser is allowed to initiate a streaming replication connection to this database in the form of md5 encryption from the host 192.168.221.160.

* 1. Configure the following parameters in the main configuration file:

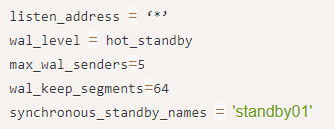


Figure 2.3

The first parameter indicates to monitor all IP; the second parameter indicates to start hot standby; the third parameter indicates how many concurrent standby databases the main database can have, here is set to 5; the fourth parameter indicates the size of a WAL log file, The default is 16MB; the fifth parameter specifies the name of the standby for synchronous replication.

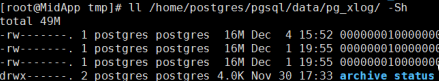


Figure 2.4

* 1. Restart the main library for the configuration to take effect. If an error is reported during startup, you can go to the log for troubleshooting.

## The slave database configuration

First of all, it is necessary to ensure that the environment before the synchronization between the main library and the slave library is consistent, so as to facilitate synchronization.

* 1. Use the pg\_basebackup generator tool which had been high lighted on the bash to generate a basic backup on the slave library. The command is as follows. If you see 100%, the backup is successful:



Figure 2.5

Add below information:

*standby\_mode = on*

*primary\_conninfo = 'application\_name=standby01 user=repluser password=123456 host=192.168.221.161 port=5432 sslmode=disable sslcompression=1'*

* 1. Modify the main configuration file where at “/home/postgres/pgsql/data/postgresql.conf” of the slave library for changing hot\_standby to enable state:

*hot\_standby = on*

* 1. Now the slave database can be started:

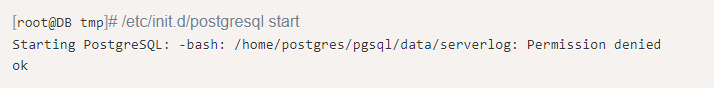


Figure 2.6

The error exception showed “Permission denied” at first time because the command to generate the backup above was executed by the root user, causing the attribute of the PG home directory to become root. Therefore, the permissions need to be reset.



Figure 2.7

To start again normally, and check the process was also successful.

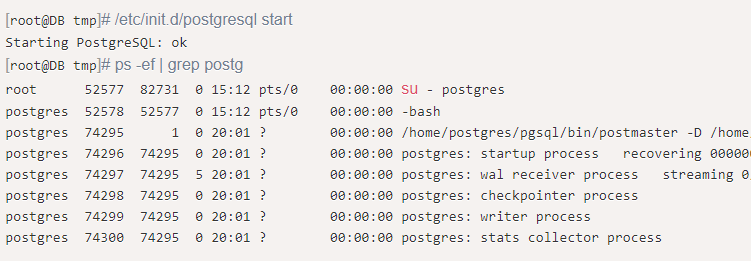


Figure 2.8

## Testing results

* 1. In the master database, query through select usename, application\_name, client\_addr, state from pg\_stat\_replication:



Figure 2.9

The output can be saw that the repluser on 192.168.221.160 is synchronizing the data of the main library through streaming replication.

* 1. To create a table for verification.
     1. Building a table on the master database and insert data:

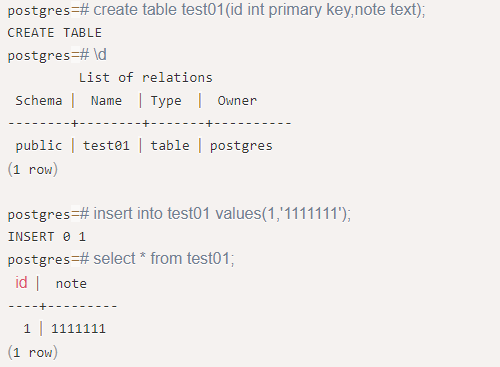


Figure 2.10

* + 1. Checking in slave database:

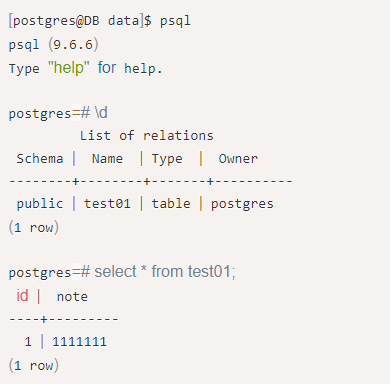


Figure 2.11

* + 1. Trying to insert data to see:



Figure 2.12

As people can see, the slave database can view the data synchronized from the master library, but it cannot write data.

# Amazon RDS

Although to manually develop and deploy the replication feature of the business databases was possible, as people knew, this used to be a complex and difficult deployment task; in addition to this, there was also operations and maintenance tasks that follow. Hosting services such as AWS RDS has cross region read replica can be greatly used in this situation. Thus, the experience of AWS RDS from mine will be shown below.

## Deploy the database

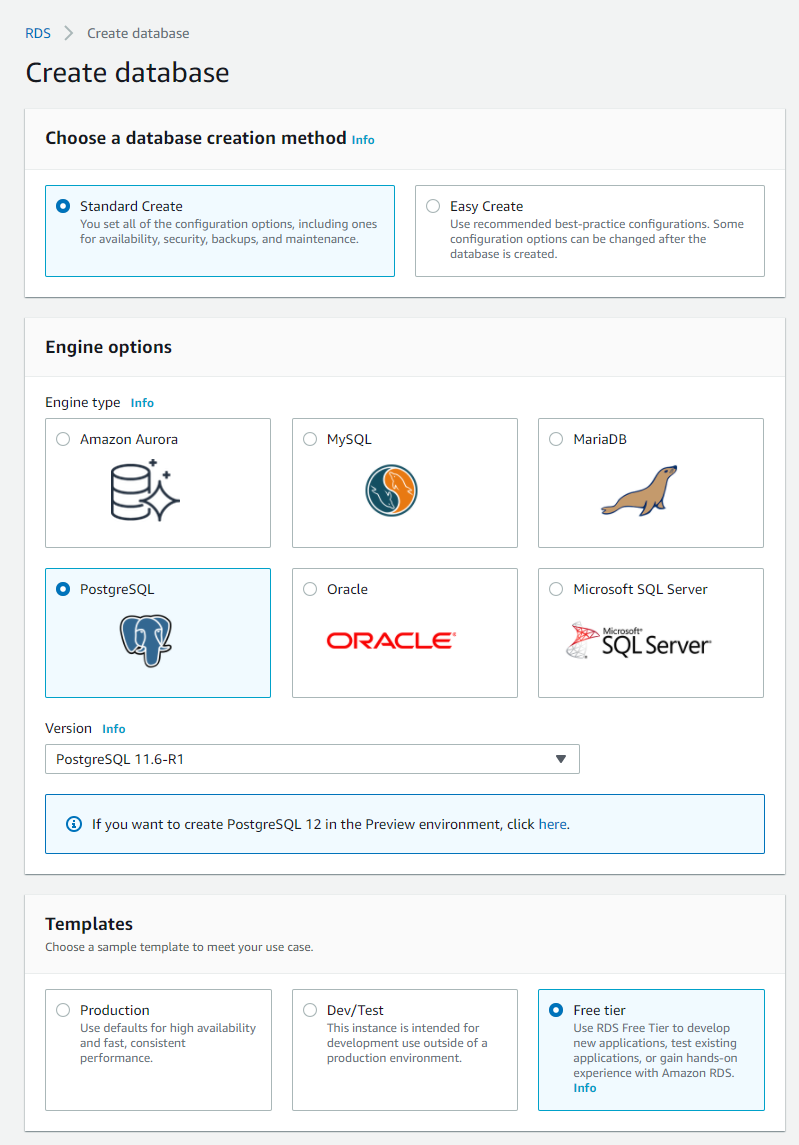


Figure 3.1

First, the location of the instance had been set in North Virginia, US East. The engine type was set as PostgreSQL. Then based on the ability of the cost and need from my own, I decided to use free tier setting of AWS RDS. For better performance, the production version would be better. Also, Amazon Aurora was a great option which was also compatible for PostgreSQL and had best performance so far in AWS service; it will be describe briefly in next section. After all set, I used pgAdmin to connect with the instance.

Before connecting with the AWS RDS instance, there still one thing had to be done. In EC2 Security Groups section, editing inbound rules type have to be turned to: all traffic and my IP. Also, turning the public accessibility in the database setting. Otherwise, the GUI tool from outside of AWS will not be able to connect to the instance of AWS RDS for security issues.

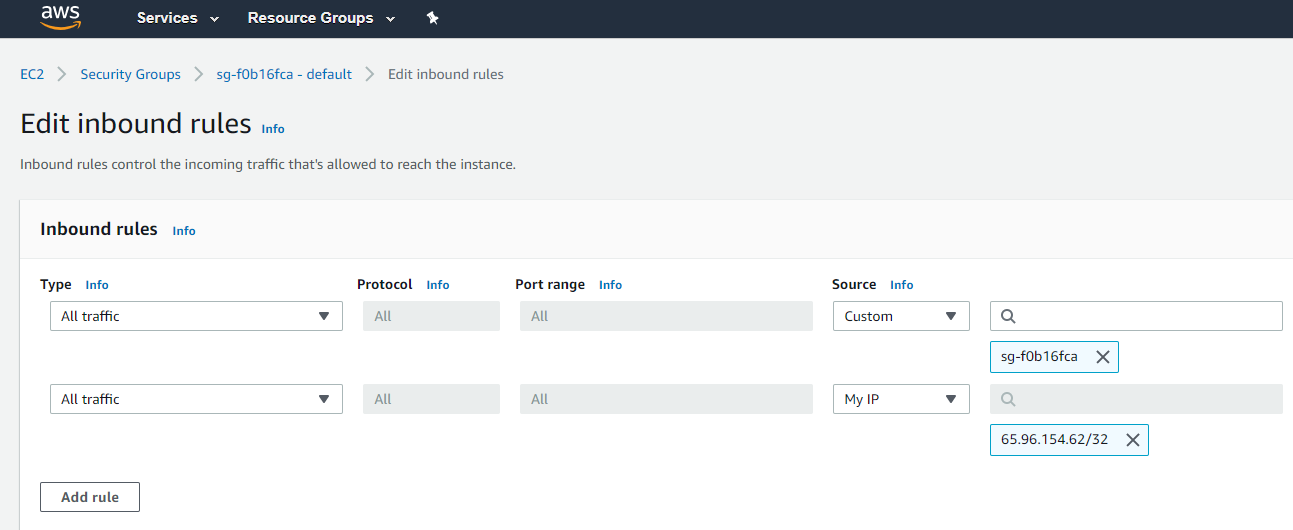


Figure 3.2

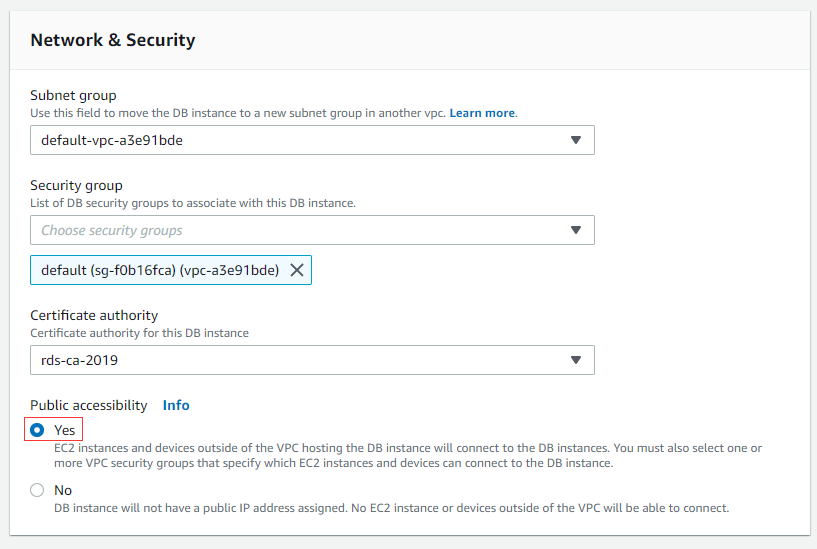


Figure 3.3



Figure 3.4

You may also need to change the port number if you had PostgreSQL server in local machine already.

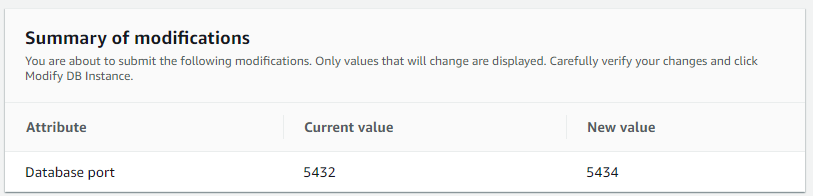


Figure 3.5

AWS RDS supported multiple kinds of database graphical user interface management tools; I would like to choose pgAdmin4 as the choice of this section because it was also recommended by AWS. Let’s start from create the new server. In pgAdmin4, creating a new server by using the information which was provided on AWS RDS console.



Figure 3.6

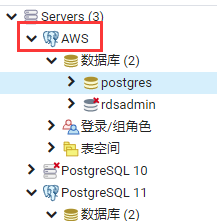


Figure 3.7

Then I restored the Netflix database by using the backup from my local machine. There were the views of the statistics of rows amount in both databases which local machine and AWS RDS by using the function I created.

*CREATE OR REPLACE FUNCTION public.count\_rows(schema text,tablename text)*

*RETURNS integer*

*LANGUAGE 'plpgsql'*

*VOLATILE*

*PARALLEL UNSAFE*

*COST 100*

*AS $BODY$declare*

*result integer;*

*query varchar;*

*begin*

*query := 'SELECT count(1) FROM ' || schema || '.' || tablename;*

*execute query into result;*

*return result;*

*end;*

*$BODY$;*

*CREATE OR REPLACE VIEW public.vw\_total\_rows*

*AS*

*SELECT tables.table\_schema,*

*tables.table\_name,*

*count\_rows(tables.table\_schema::text, tables.table\_name::text) AS count\_rows*

*FROM information\_schema.tables*

*WHERE (tables.table\_schema::text <> ALL (ARRAY['pg\_catalog'::character varying, 'information\_schema'::character varying]::text[])) AND tables.table\_type::text = 'BASE TABLE'::text*

*ORDER BY (count\_rows(tables.table\_schema::text, tables.table\_name::text)) DESC;*

*;*

Thus, the process was successful based on the outputs below.



Figure 3.8



Figure 3.9

Also, all the event will be record as the logs in AWS RDS console for checking.

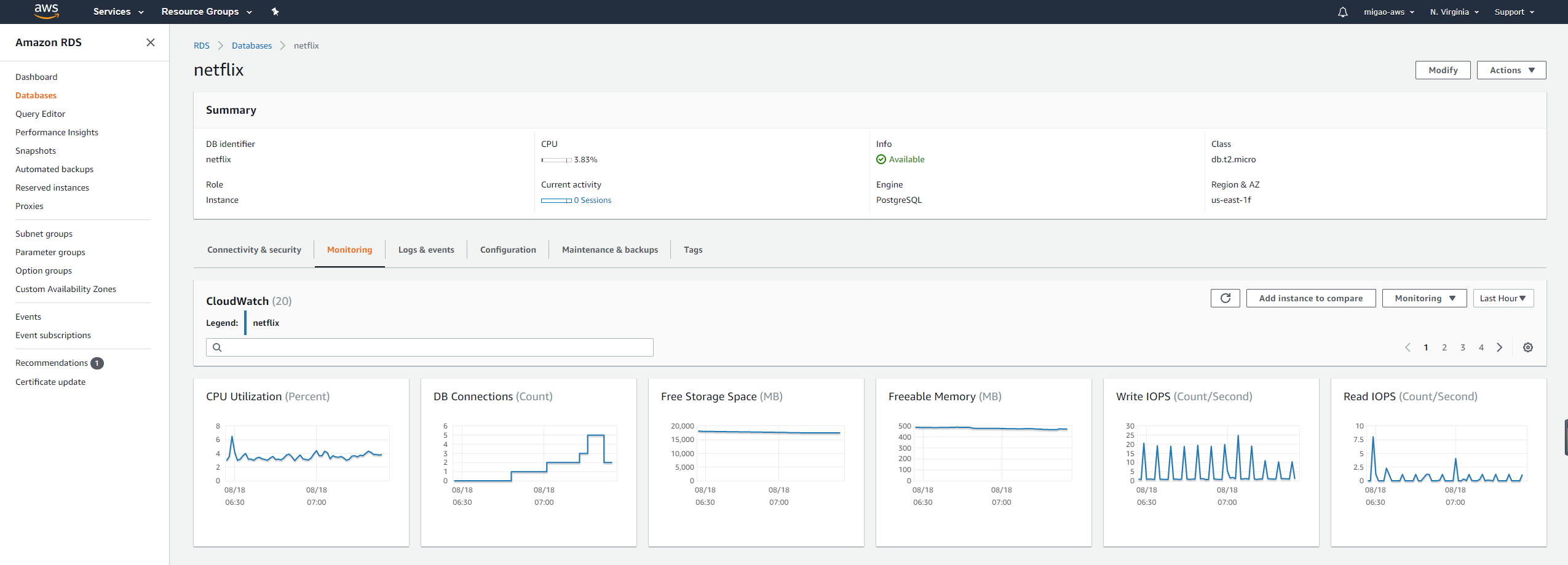


Figure 3.10

## Read Replica

Amazon RDS uses the built-in replication functionality of the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server DB engines to create a particular type of DB instance called a read replica from a source DB instance. The instance of source DB is the primary instance of DB. Updates made to the primary DB instance are copied to the read replica asynchronously. By routing read queries from your applications into the read replica, you can reduce the load on your primary DB instance. Using read replicas, you can scale out elastically for read-heavy data base workloads beyond the capacity constraints of a single DB case (Amazon RDS Read Replicas, 2020).

The first thing that was to define an existing DB instance as the source when building a read replica. Instead Amazon RDS takes a snapshot of the source instance and generates a snapshot-only read-only instance. Instead Amazon RDS uses the asynchronous DB engine replication approach to update the read replica if the primary DB instance changes. The read replica operates as an instance of DB which only allows read-only connections. Applications connect to a replica of read as they do to any instance of DB. In source DB for Amazon RDS replicates all databases.

For creating the read replica, the process can be started from the console. For Actions, choose Create read replica.

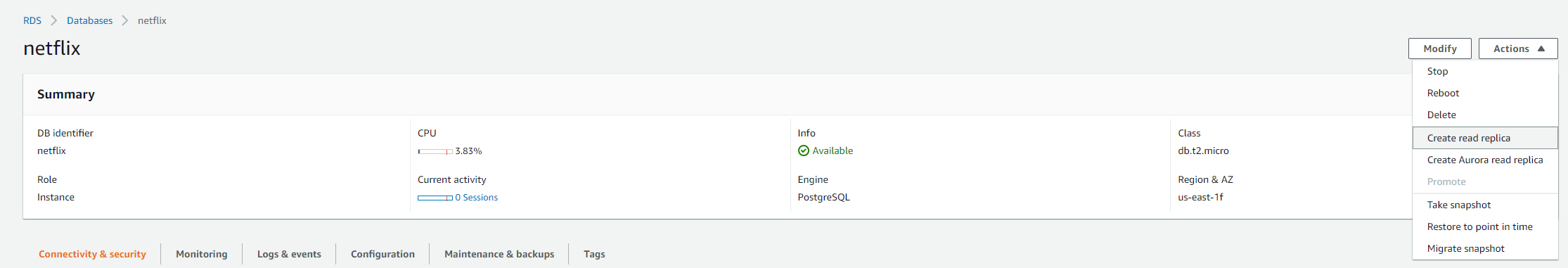


Figure 3.11

For DB instance identifier, enter a name for the read replica. Adjust other settings as needed.

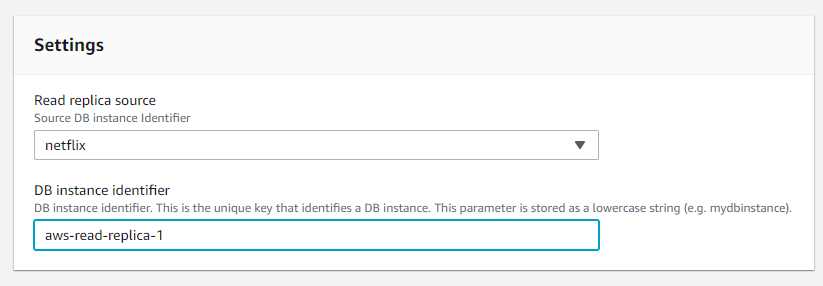


Figure 3.12

After creating the replication, it had to wait few minutes. Once the action completed, everything will be same as what had been done above when applying the master database instance. Also the console will show the lag of each read replication from the master database.

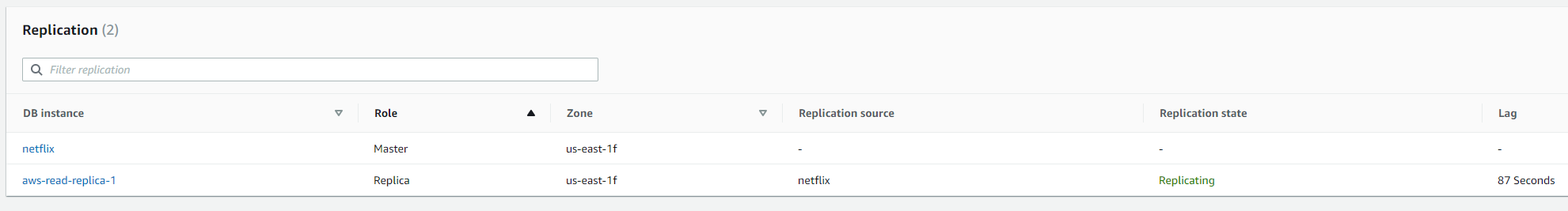


Figure 3.13

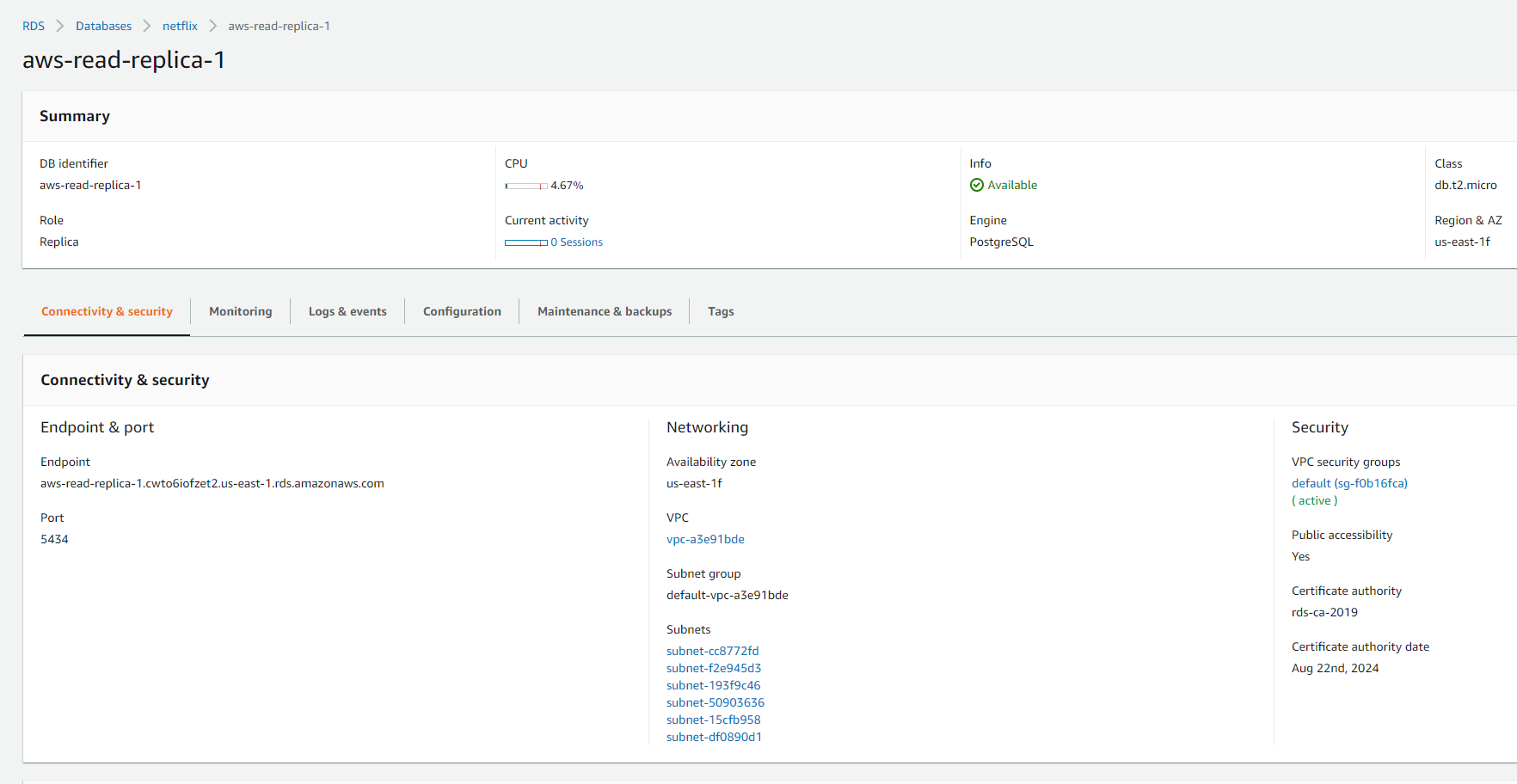


Figure 3.14

Next step was to use the information from the console to connect the replica database. It will show that everthing will be replicated as same as the master database. The member table had been chosen randomly as the test sample. As the replication of the master database, both of the databases will has the same data in that table.

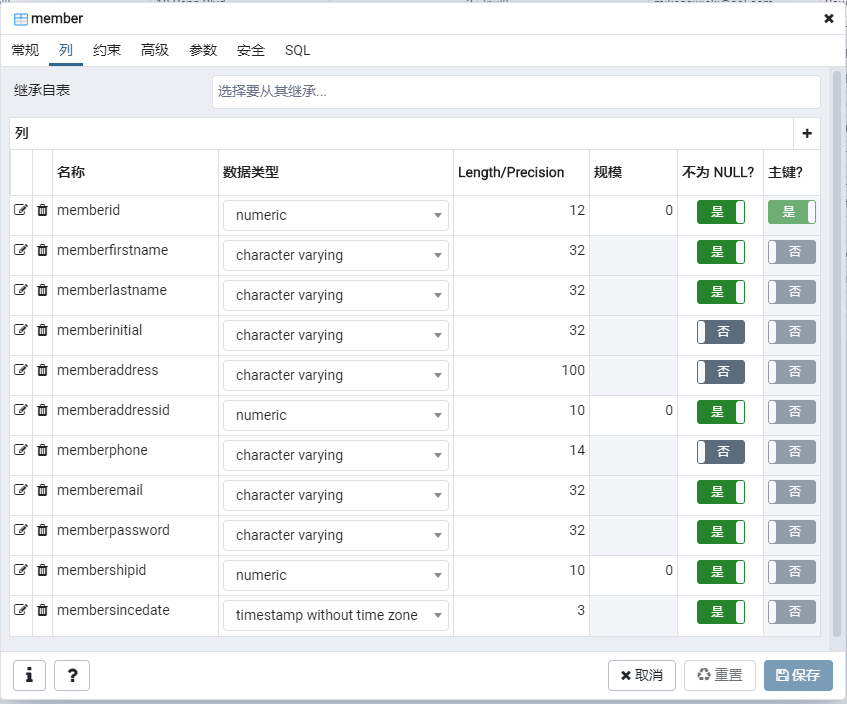


Figure 3.15

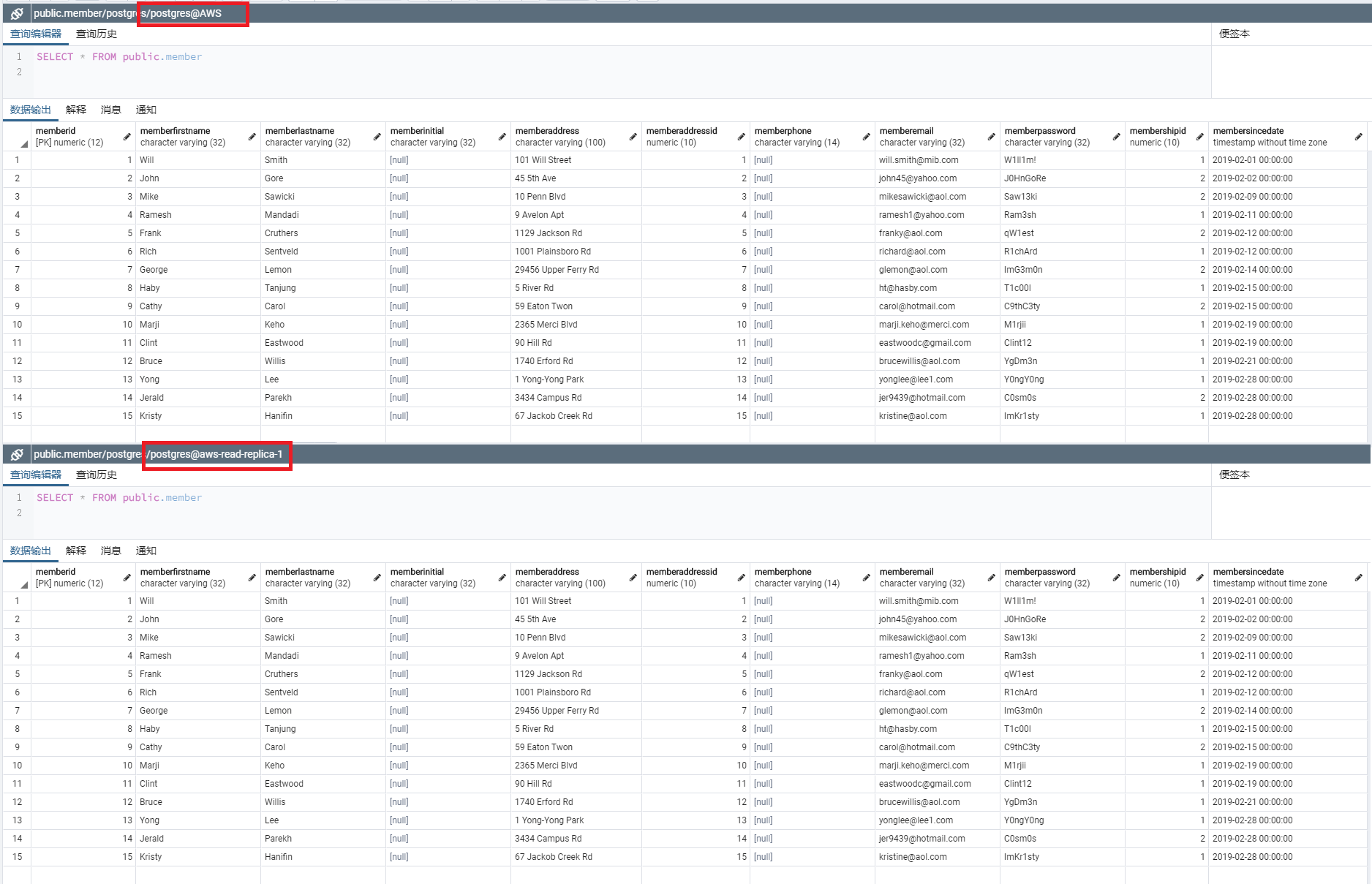


Figure 3.16

And if trying to modify the data from the read replica database at this moment, the error report presented in Figure 3.17 will alert that this database was read-only no matter what actions were taken without the select query which was reading.

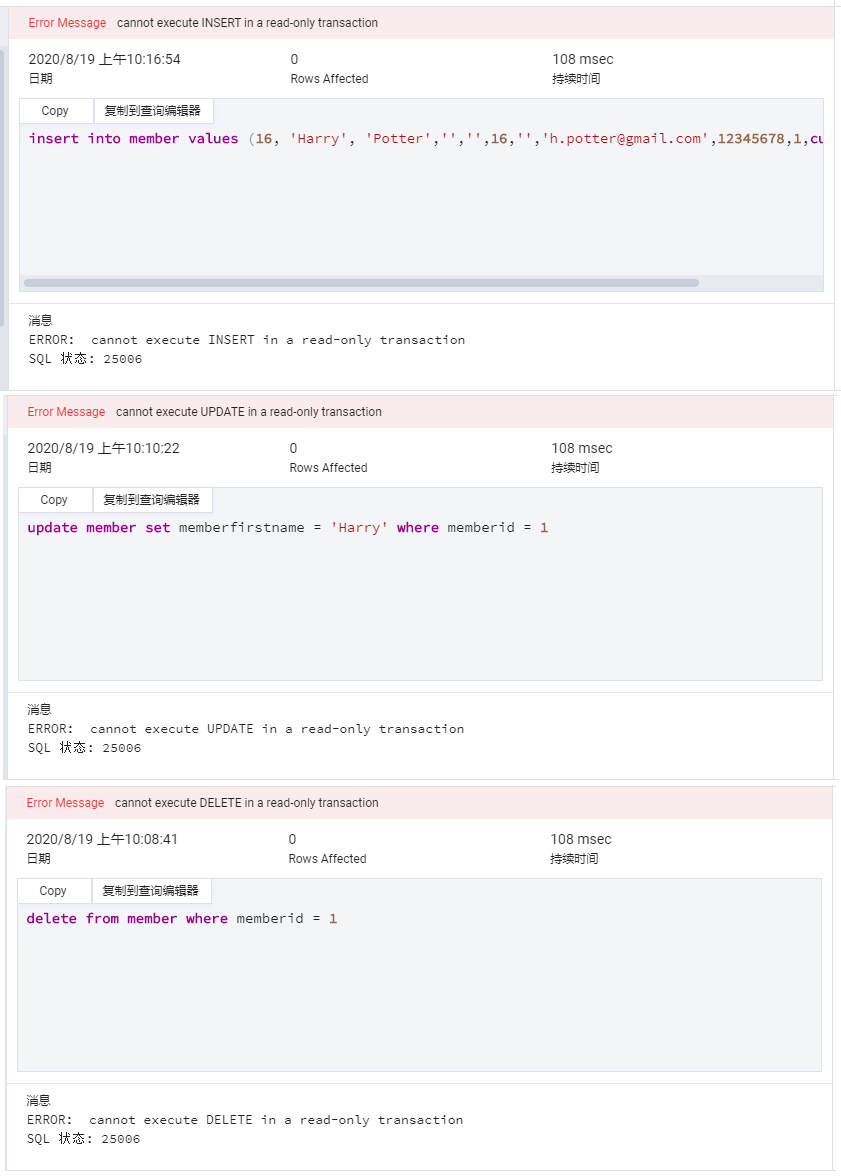


Figure 3.17

Next, trying to make some insert action for testing in the related table from master database. There will show the following steps and outputs from INSERT to UPDATE, then DELETE. However, one custom parameter group had to be created for recording the log about DML; the value of log\_statement() as mod means “logs all ddl statements, plus data-modifying statements such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM (Documentation: 11: 19.8.&nbsp;Error Reporting and Logging 2020).” The test will only do about DML this time; to set the value as all will make it record additional logs about DDL if the requirement was needed. Then, modifying the database option to use the custom parameter group before reboot the database instance for applying the change.

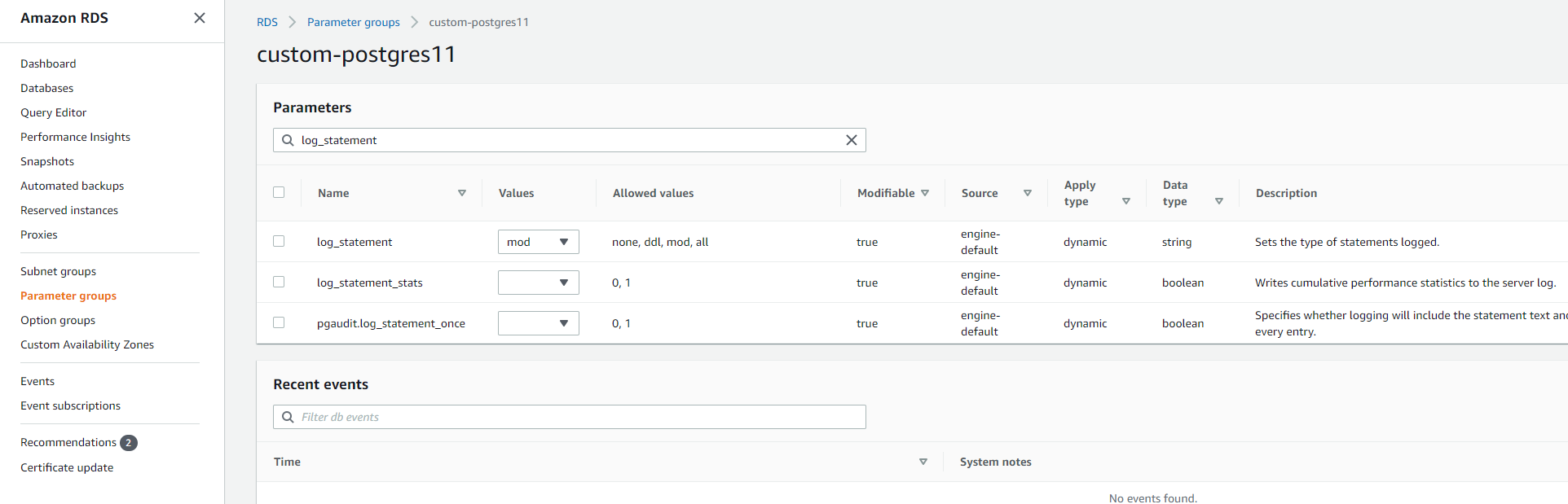


Figure 3.17

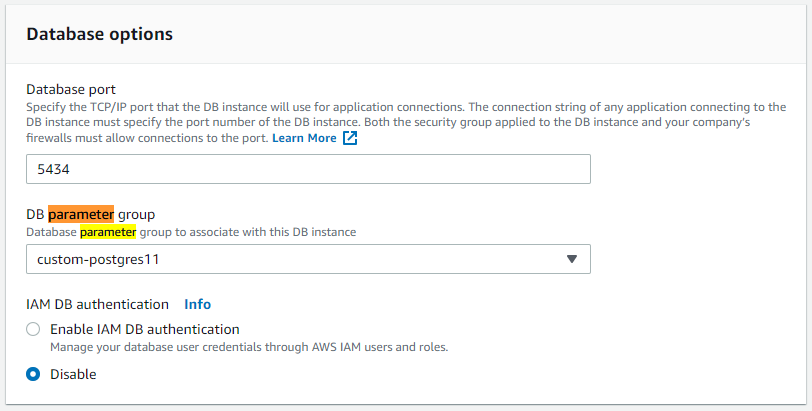


Figure 3.18

*insert into member values (16, 'Harry', 'Potter','','',1,'','h.potter@gmail.com',12345678,1,current\_timestamp);*

*update member set memberfirstname = 'Chris' where memberid = 16;*

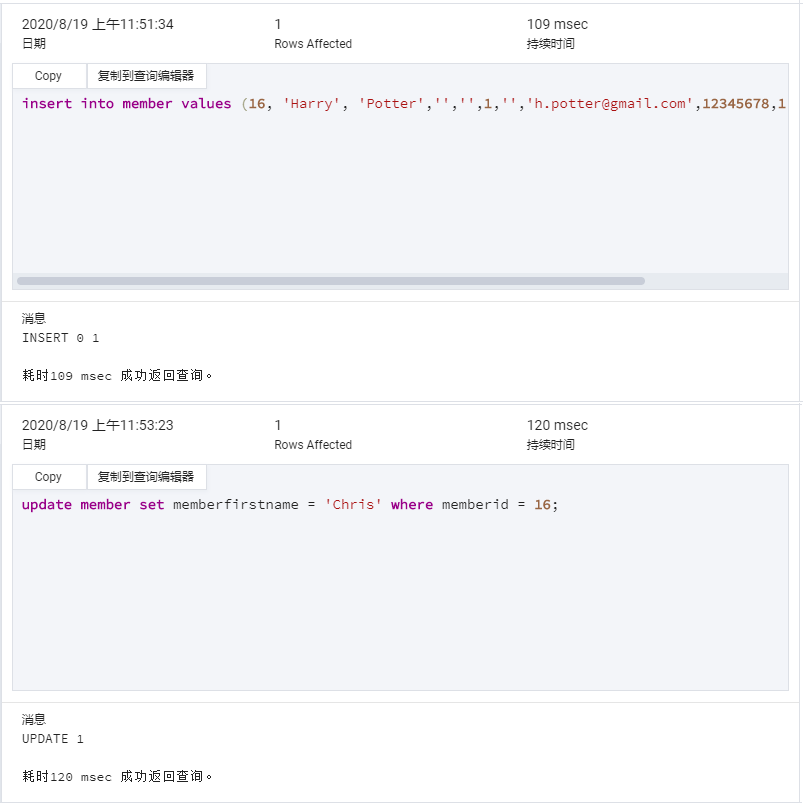


Figure 3.17



Figure 3.18

The reports will show on the log of both databases (Figure 3.19). The top log was the log of master database. It had shown what the log\_statement(mod) would present:

2020-08-19 15:51:35 UTC:65.96.154.62(56976):postgres@postgres:[13334]:LOG: statement: insert into member values (16, 'Harry', 'Potter','','',1,'','h.potter@gmail.com',12345678,1,current\_timestamp);  
2020-08-19 15:52:58 UTC::@:[10968]:LOG: checkpoint starting: time  
2020-08-19 15:52:58 UTC::@:[10968]:LOG: checkpoint complete: wrote 7 buffers (0.0%); 0 WAL file(s) added, 0 removed, 1 recycled; write=0.883 s, sync=0.002 s, total=0.898 s; sync files=7, longest=0.002 s, average=0.000 s; distance=12 kB, estimate=12 kB  
2020-08-19 15:53:24 UTC:65.96.154.62(56976):postgres@postgres:[13334]:LOG: statement: update member set memberfirstname = 'Chris' where memberid = 16;

Compared with the time show on the log of pgAdmin4, the lag can be known was very low. And in bottom of the figure was the log of read replica. It had shown the summary per five-minute period and the last completed transaction timestamp.

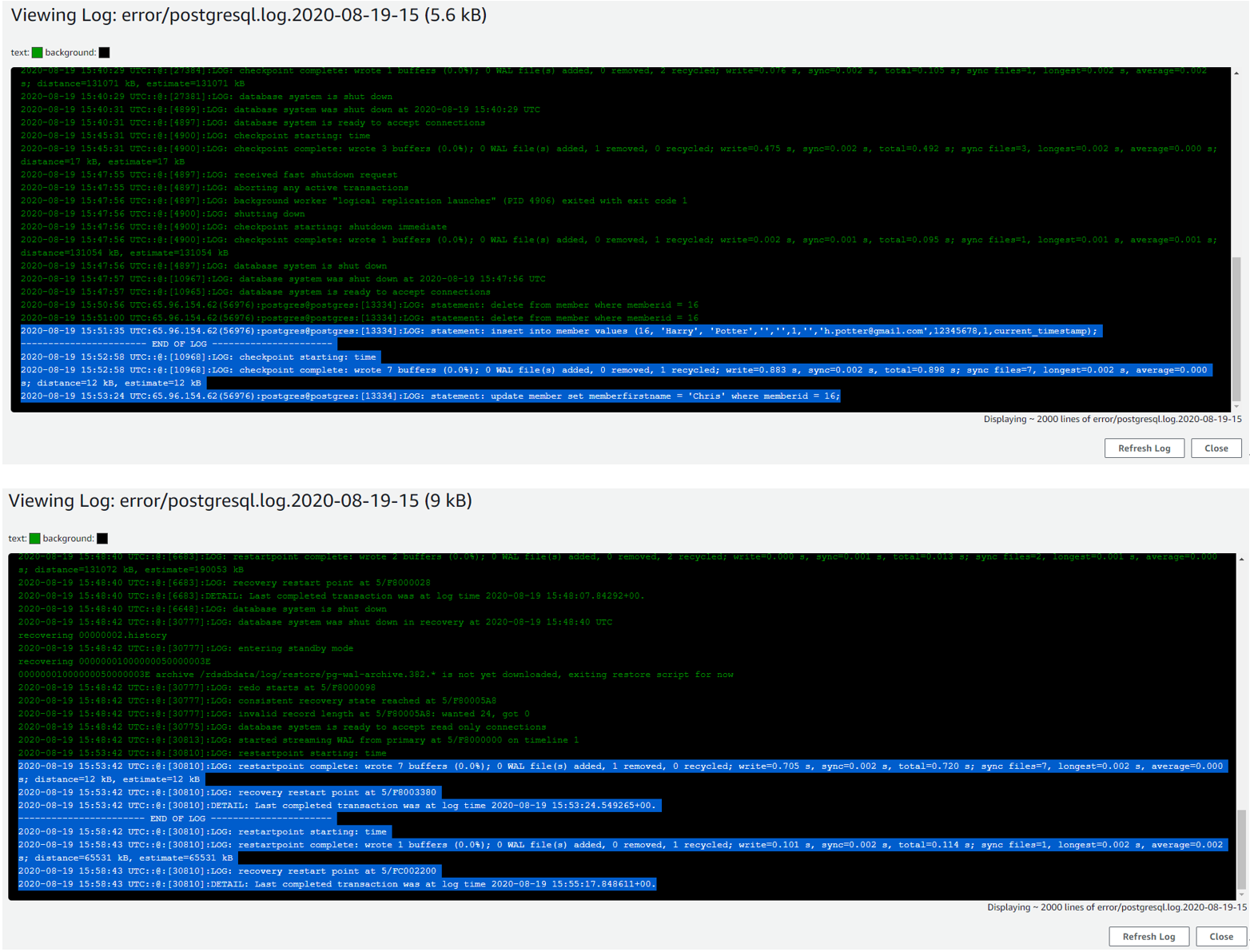


Figure 3.19

At the end of the test, this row will be deleted following the action showed in Figure 3.18 successfully.



Figure 3.18

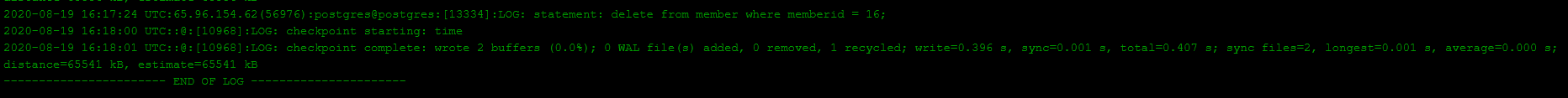


Figure 3.19

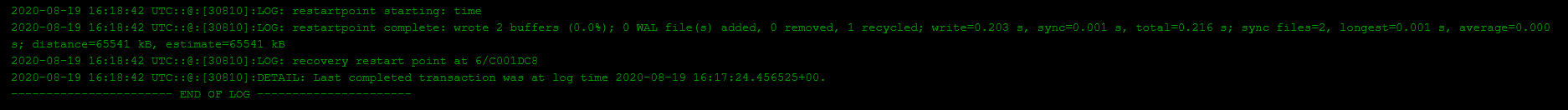


Figure 3.20

Figure 3.19 and 3.20 showed the logs of master database and read replica; the timestamps clear presented transactions for each one as well. Therefore, there stated that the lag of AWS RDS database and the replicaiton was very slight and low that can be done within milliseconds.

Moreover, creating a read replica in a different AWS region by using cross region read replica can effectively improve the data reading speed and disperse the pressure of a single database. Also, this would make it easier to migrate from a data center in one AWS Region to a data center in another AWS Region. The following diagram illustrates how Amazon RDS for PostgreSQL performs replication between a source and replica in different Regions show at the article, Best practices for Amazon RDS for PostgreSQL cross-Region read replicas (Singh, 2020).

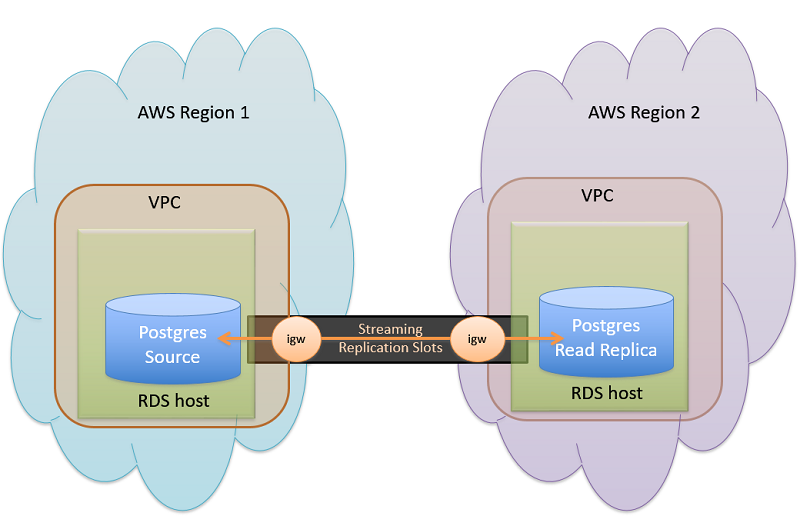


Figure 3.21

For the operation, the only action needs to be additional from general read replica is choose a value for Destination region and Destination DB subnet group in the Network & Security section.

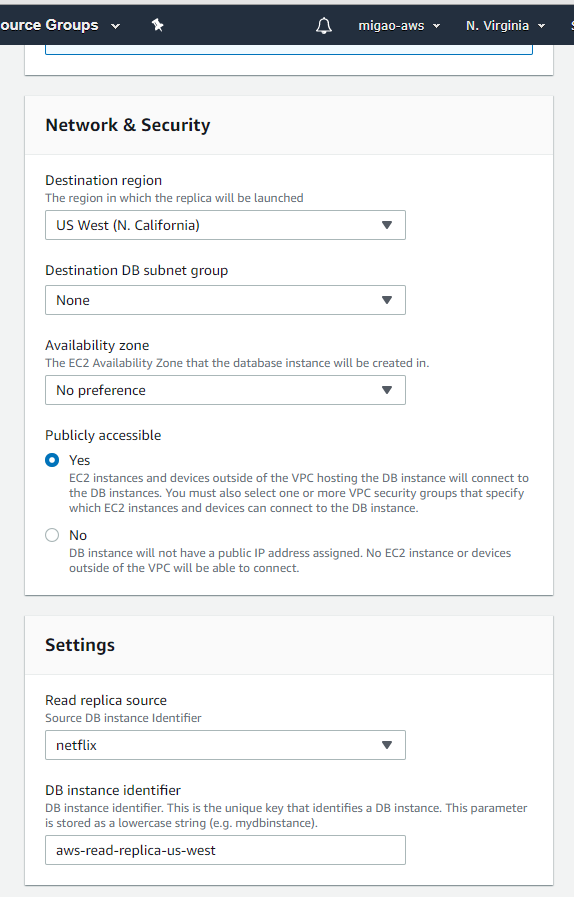


Figure 3.22

Therefore, AWS RDS was a great choice for the enterprise which was planning to grow or make industrial upgrading. Powerful service functions and convenient operation have reduced the company's operating costs and risks.

# Amazon Aurora

However, the amount of user’s input has gradually increased and become very dense; and the network latency requirements of read-only nodes are kept as low as possible. AWS RDS may no longer had the ability to handle this kind of problem. Thus, Amazon Aurora would be the better option in this situation.

Amazon Aurora was still a kind of relationship database which was fully compatible with MySQL and PostgreSQL. Amazon Aurora Global Database is designed for globally distributed applications, allowing multiple AWS regions to cover a single Amazon Aurora database. It replicates the data without impacting on the output of the database, allows quick local readings with low latency in each area, and provides recovery from regional outages from disasters. Important workloads with a global presence, such as banking, travel, or gaming applications, have stringent criteria in terms of availability and will need to accommodate a regional outage. This historically needed tougher tradeoffs between performance, availability, cost, and data integrity. Global Database uses storage-based replication with a typical latency of less than a second, using a dedicated infrastructure that leaves your database fully available for workloads in the application. In the unlikely event of a regional failure or outage, it is possible to enable one of the secondary regions to read and write capabilities in less than one minute (Amazon Aurora Global Database, 2020). Generally, there had three main advantages of Amazon Aurora.

First, Amazon Aurora’s network latency of replication was lower than another database. Although AWS RDS for PostgreSQL was using physical replication which has an expensive cost and efficient performance may not meet such problem usually, some other database management system might have specific situation need to be considered. The example worked with AWS RDS for MySQL (2020) will be shown below which stated by Lin Lv, AWS database expert architect.

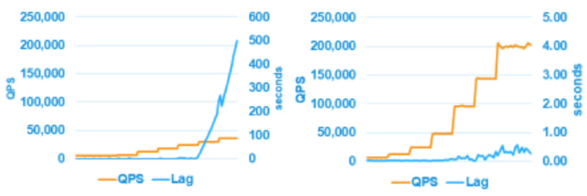


Figure 4.1

There was the figure of comparison between logical replication (Figure 4.1 left side) and physical replication (Figure 4.1 right side). As people knew, the replicaiton based on Binglog which is kind of logical replication of AWS RDS for MySQL was closely related to the inpout load. When the business has a high frequency and a large number of inputs, the lag will inevitably increase. Sometimes, the lag may be very exaggerated such as tens of seconds or minutes, resulting in unavailability of business. However, Amazon Aurora's log-based physical replication method is very stable. Even when the business faces heavy input, it can still provide only sub-second lag.

Second, this log-based replication method has relatively little impact on the input performance of the master database which in main region. This will hardly cause a significant decline in input capacity due to replication.

Moreover, disaster preparedness in Amazon Aurora had been improved as much as possible. When something goes wrong, it only takes one minute to raise the auxiliary region. There was an example from Lin Lv (2020).

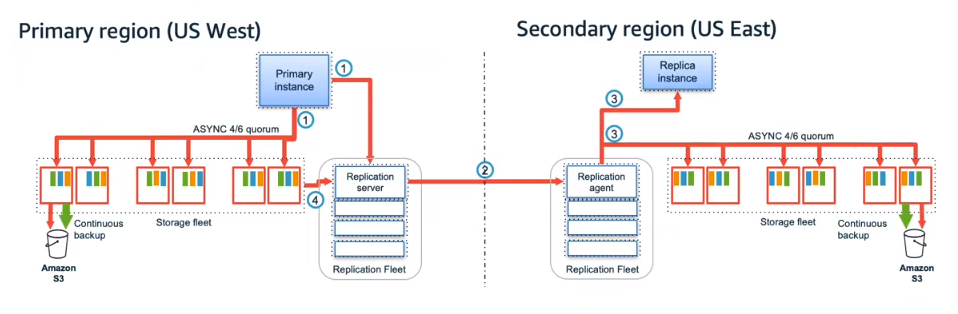


Figure 4.2

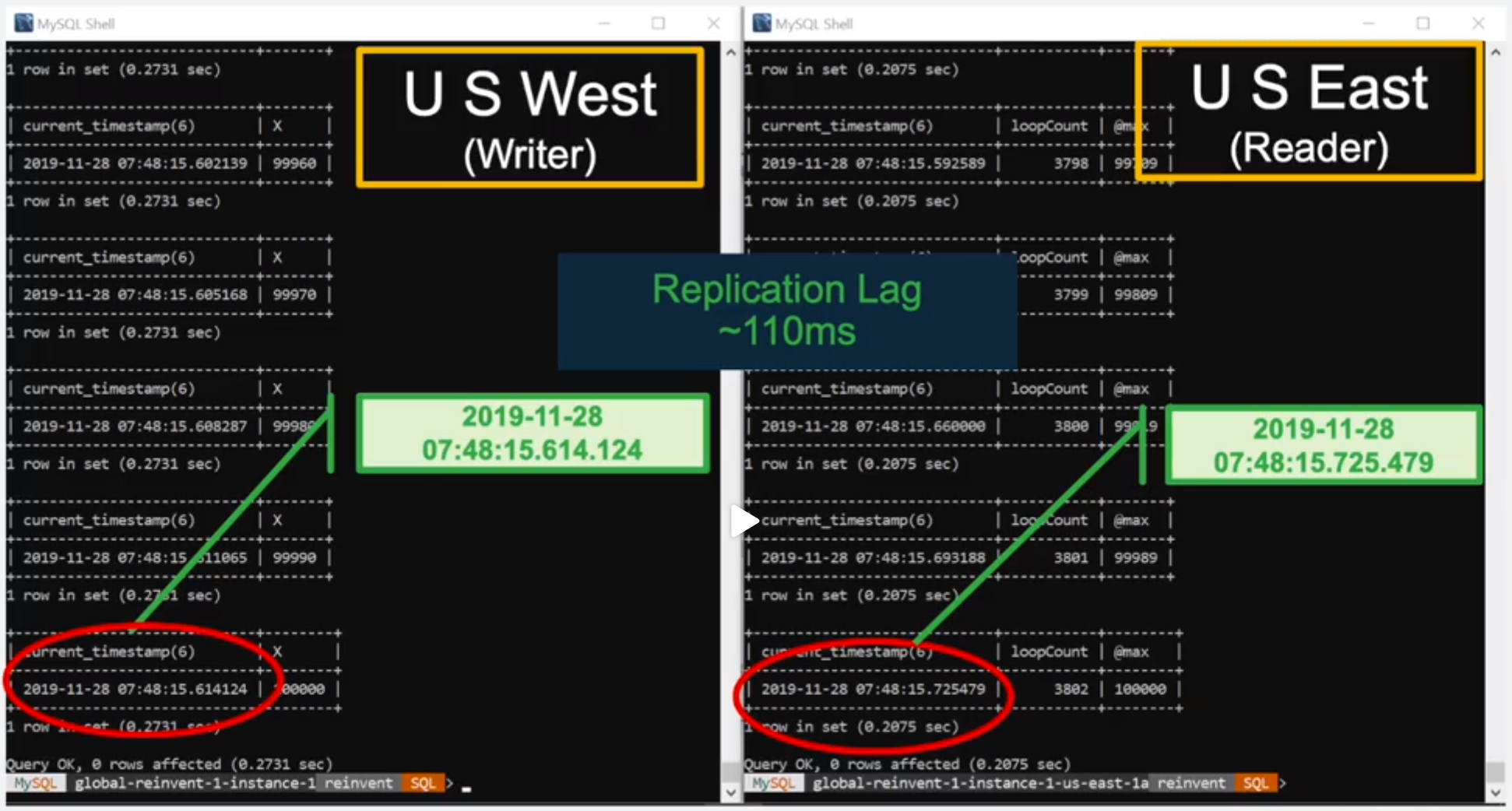


Figure 4.3

In the test, records were continuously inserted in the main region, and a timestamp was left every time it is inserted (Figure 2); then, the auxiliary region was continuously queried and a timestamp was also left. When this record is copied from the US West to the US East, the two timestamps can be compared; the delay is about 110 milliseconds (Figure 4.3).

# Conclusion

Replication is one of the amazing features of cloud-based database. It implemented data synchronization across machines and regions, and physically reduced the time and other costs of data interaction. AWS RDS is only one of the Cloud service suppliers in this big network market; other company such as Google, Microsoft, Alibaba, and other great ones were also providing their own cloud services. Furthermore, Partitioning is also a feature which included in the cloud-based database together with the replication, and also, they were parts of the distributed database. Distributed databases are a set of databases split across various locations that communicate and deliver services over a network. The benefits of it such as easily scale-out of the entire production and set up replication to maintain your data's integrity and high availability, simple failover. However, some cons such as hard to maintain and set up made me extremely hard to complete the works in a limited short period. This would be some regrets at this moment; I would like to try to implement this part in the next step of this project.

# 5. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Version** | **Description** |
| Mi Gao | 08/01/20 | 1.0 | Initial Document Creation |
| Mi Gao | 08/03/20 | 1.1 | Built the Structure |
| Mi Gao | 08/10/20 | 1.5 | Added Contents |
| Mi Gao | 08/12/20 | 1.6 | Fixed Citations |
| Mi Gao | 08/18/20 | 2.1 | Added Amazon Aurora section |
| Mi Gao | 08/19/20 | 2.2 | Added additional tests for AWS RDS |

# Bibliography

AWS. (n.d.). (2020). DMS: Diagnostic Modeling System. Retrieved August 12, 2020, from https://docs.aws.amazon.com/dms/

PostgreSQL. (2020, July 6). Streaming Replication. Retrieved August 12, 2020, from <https://wiki.postgresql.org/wiki/Streaming_Replication>

科技小能手. (2017, November 12). Centos6.5下postgresql9.4.3同步流复制的standby安装与配置-阿里云开发者社区. Retrieved from https://developer.aliyun.com/article/482836

Alibaba Cloud. (n.d.). Retrieved August 12, 2020, from https://us.alibabacloud.com/

PostgreSQL (2020). Documentation: 12: Pg\_basebackup. Retrieved August 12, 2020, from https://www.postgresql.org/docs/current/app-pgbasebackup.html

AWS. (n.d.). Amazon RDS Read Replicas. Retrieved August 12, 2020, from https://aws.amazon.com/cn/rds/features/read-replicas/

AWS (n.d.). Working with Read Replicas. Retrieved August 12, 2020, from https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER\_ReadRepl.html#USER\_ReadRepl.Create

PostgreSQL (2020). Documentation: 11: 19.8.&nbsp;Error Reporting and Logging. Retrieved from https://www.postgresql.org/docs/11/runtime-config-logging.html

Singh, V. (2020, May 26). Best practices for Amazon RDS for PostgreSQL cross-Region read replicas [Web log post]. Retrieved from https://aws.amazon.com/blogs/database/best-practices-for-amazon-rds-for-postgresql-cross-region-read-replicas/

AWS. (2020). Amazon Aurora Global Database. Retrieved August 18, 2020, from https://aws.amazon.com/rds/aurora/global-database/?nc1=h\_ls

Lv, L. (2020, July 14). Topic 4: The End of the World Is Neighbors-The Same Library in the World-Lu Lin-0629. Retrieved from https://weibo.com/tv/show/1034:4526649744752647?from=old\_pc\_videoshow