Providing High Availability for the cloud database system

Priyanka R Patel (0883743)
Master of Computer Science
Lakehead University
Thunder Bay, ON, Canada
ppatel30@lakeheadu.ca

Dr. Jinan Fiadhi
Computer Science
Lakehead University
Thunder Bay, Canada
jinan.fiaidhi@lakeheadu.ca

***Abstract*— Providing high availability means a system is remain highly available for the client in the phase of any type of failure. In the present time, the idea of high accessibility isn't just utilized in critical applications, however, it is likewise utilized for all sort of application. Since these applications require databases, the databases themselves must be made highly(exceptionally) accessible. To provide such type of service different techniques are used such as the hot stand by technique, SHADOW system [1], PostgreSQL [3] and Remus DB [2]. Remus and Remus DB utilize virtual machine relocation to a backup database which is totally transparent to the DBMS. This methodology brings about a ton of overhead, as the measure of data moved to the backup database is huge and this exchange is done regularly. Another framework Postgres-R works just with PostgreSQL databases.**

In this research paper we represent a system which is based on AWS cloud. In our system we used EC2 instance for web base application and RDS which provide database service. There are two database primary and secondary and both are synchronized. In case of any failure in the primary database the secondary will start and handle all the request of client. There is no downtime during the process, so there is no data loss.

Key word: - high availability, cloud, EC2 , RDS, MySQL

I. INTRODUCTION

In present time, there are numerous application which are being given to clients over the cloud such as banking, social media, video spilling, and many more. The main aim of those applications is to provide high availability (HA). Downtime results in heavy loss in business and client disappointment, that is why the uptime must be high. High accessibility of frameworks infers that the databases at their backend should likewise be very accessible.

To provide this facility we used AWS cloud. In our system we used EC2 instance for web base application and RDS instance for database service. There are two availability zone named as primary and standby(fig1) in RDS. By default, the primary database which is in MySql handles all the user request. The secondary database is synchronized with the primary one and also known as stand by database. So, changes made to the essential database engendered to backup database. At the time of primary database failure, the secondary (standby) database starts working and handles all the client's request. Here, we have used hibernate in web application to perform different operation, which provide the flexibility to the administrator. The administrator can change data base at any time with any database such as Mysql, Mogo DB, Oracle, etc. The hibernate is database independent and we can change the database by changing only few lines in code. We used Spring MVC to make website which is based on MVC (model, view, controller) provide best security service.

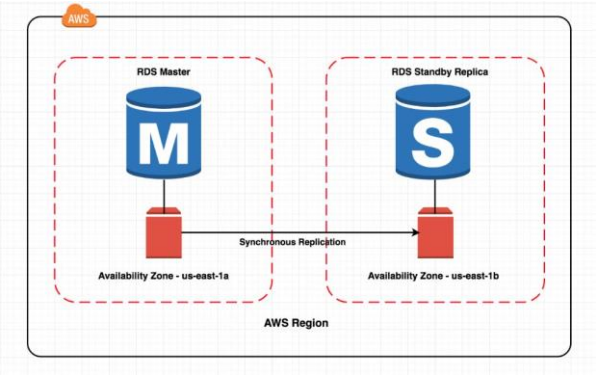

Figure 1 RDS Structure

II. RELATED WORK

Lot of work has been done to provide high availability to the system. The strategies utilized for doing this can likewise be utilized for databases. Here, I would like to discussed few technologies.

A. Traditional Technique to provide HA

Hot standby technique is also called as traditional technique [1] to provide high availability. In this technique there are two DBMS (1) Primary and (2) Secondary (fig 2).

In this technique the primary database handles the user request. It except user request and perform different type of operation in DB. The change which are made in a primary server is also propagated to back up server (standby). When the failure occurs at that time the back will start working as primary server. Though the system is highly available there is little downtime, while the primary DB handover the request to the standby.

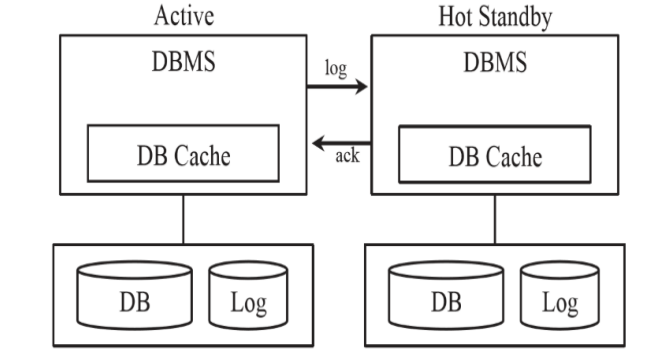

Figure 2 Hot stand by technique Architecture.

There are few challenges in this system which are mention below.

1 Complex to implement DBMS

⇒ It is difficult to implement database management system (DBMS) and it is also complex for administrative to manage the system

2 Active handovers

⇒ When the failure occurs at that time the primary server stops working and the backup server star working. The main problem is that there is no active hand over from primary server to backup server. Due to that there is few data loss.

3 Redirect client requests

⇒ It is difficult to redirect client request in this system. The user continually sends the request to primary DB and in that case, it is difficult to re-direct the request from primary to backup DBMS.

B. RemusDB

Here I would like to describe the research work [2] which is published by Published by Microsoft. In this research the they use the Virtual machine (VM) layer concept to provide High Availability. They implement active / standby replica in virtual machine layer. they pushed complexity out of Database. In this concept any DBMS can be made high availability with only little or no code change. In this system he uses the concept of RemusDB, because it is reliable and cost-effective. RemusDB is part of remus. Remus is part of the Xen hypervisor which is mainly used to provide high availability in general term. Here the main work of remus is to maintain a replica of running virtual machine. RemusDB is improvement in remus for database. In this system the time slots are divided in to epoch. There is checkpoint at the end of each epoch. When the failure occurs, the system start taking back up from last check point. The data after the last checkpoint is loss (we can see in the diagram 5, the data of Slot C is not more, means system can't tack backup of that). The Data buffered in the network until checkpoint come (at each check point the data are stored) .

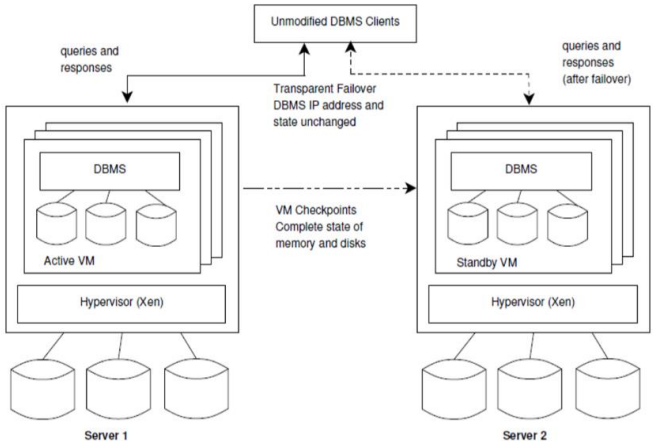

Figure 3 RemusDB System Architecture

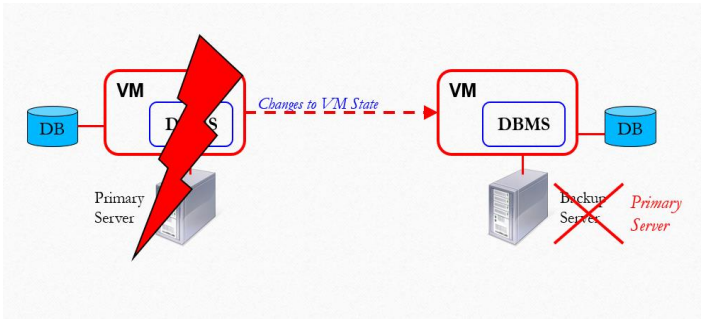

Figure 4 RemusDB Failure

There are two explanations behind the execution overhead caused by Remus and other comparative VM checkpointing frameworks.

1) Database frameworks make a substantial utilization of memory.

At every check point the whole memory state is transferred from primary to backup DB(database). The amount of data(information) is to large.

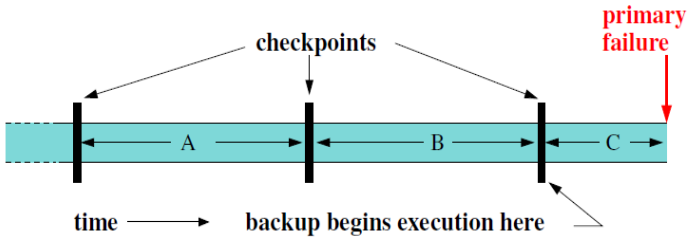

Figure 5 Execution time line

2) DB workload can be sensitive for network dormancy(latency).

C. SHADOW System

"Database High Availability Using SHADOW Systems". In this research [1] they use the concept of active standby technique in cloud environment. In SHADOW system they push the task of managing DB replication out of the DB system and pushed into the underlying storage service. They represent the result of a performance evaluation using a SHADOW prototype on Amezon's Cloud. It is enhance version of traditional hot standby technique.

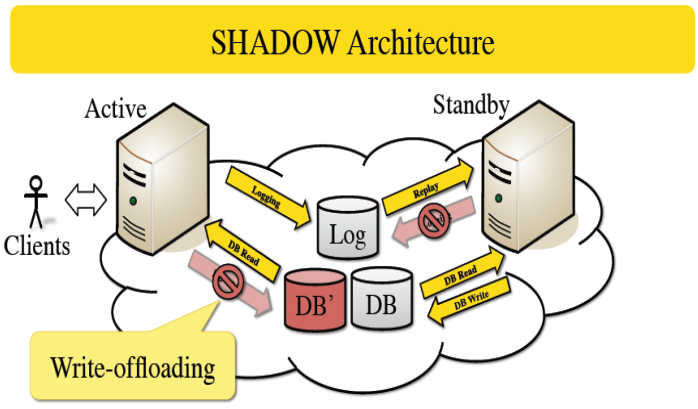

Figure 6 Shadow System Architecture

"When a SHADOW system starts operation, only the (unprotected) active DBMS is running and there is only one database. A secondary DBMS is created by taking a

snapshot of the primary's database, and that snapshot becomes the primary database managed by the secondary(standby) ". " In this system The Active DBMS perform transactional updates on its local cached copy of the DB. The transaction is committed once its log records are safe in the persistent storage system." Here when failure occur at that time the backup database is start working as primary and use log for different operation.

D. Postgres-R

Postgres- R [3] is an extended version of PostgreSQL database. It is designed in a such a way that it will provide different features such as scalability, reliability and flexibility. The basic use of this system is to provide service of load balancing and high accessibility to database. Due to it's different features large objects can be replicated. The system uses two phase locking for concurrency control. The system is working on the concept of read-one-write-all in which the transaction is divided in to four phases. Postgres-R works just with PostgreSQL databases.

1)Local phase
In this phase all the transaction is done and it is replicated at local replica
2)Send phase
In this phase the data are propagated to the replica.
3)Synchronization phase
serializability of the exchanges is protected
4)Write phase
In this phase one replicas perform writes and the respective replicas execute the transaction.

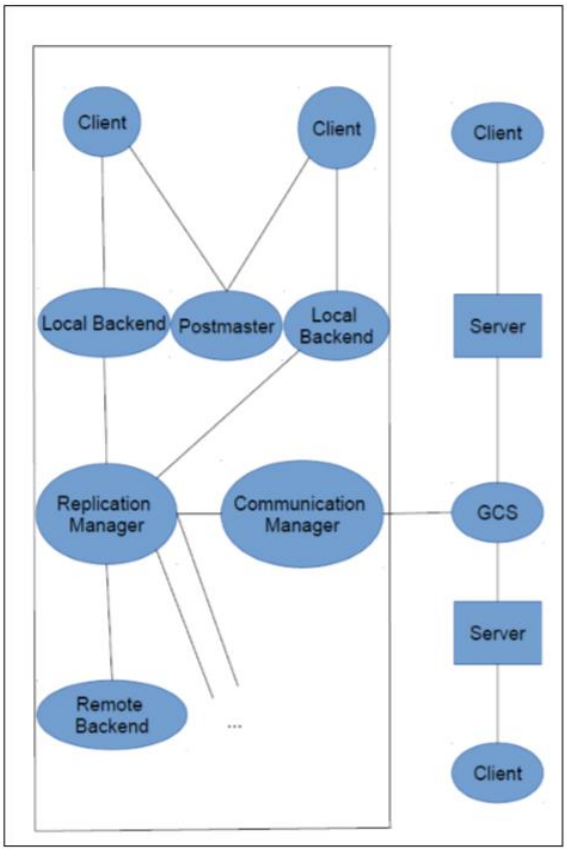

Figure 7 PostgresR

III. OUR APPROCH

We are proposing the system which provide high availability by using AWS cloud. We used EC2 and RDS in AWS cloud where EC2 instance used for web base application and RDS is providing database service.

- Amazon Elastic Compute Cloud (EC2):

EC2 [5] provide the virtual machine which is called as instance. The different templates are available in AMIs (*Amazon Machine Images*) to create instance. In this system we used Linux AMIs in US North virginia east zone. The instance would have different configuration and we are flexible to give that type of configuration (such as memory, storage, and networking capacity, CPU for our instances- it is depending on web application that what type of memory and storage it required).

- Amazon Relational Database Service- RDS:

RDS [4] is web service (and it stands for Relational Database Service). There is different database instance type (-optimized for memory and performance). It provides six different database engine such as MySQL, Amazon Aurora, PostgreSQL, MariaDB, Oracle Database, and SQL Server.

Here we used MySQL database engine. We create instance in RDS and made a database in MySQL. Here we use the concept of mirroring provide by RDS and create replica of primary database. We have created RDS instance in US Viginia region and a primary in one Availability Zone named as US east 1a , and a standby in a second Availability named as Zone US east 1c (fig 8). Data written to the primary will be synchronously replicated to the secondary DB. In case of primary DB fails, the standby becomes working as primary and it will start handling user's request (fig 9 and 10). We can see from the fig that all the client request id handle by the Primary RDS instance and it is synchronized with the standby RDS instance. Both the instance is in different availability zone.

The Failover mechanism automatically changes the DNS address of database instance to address of standby instance

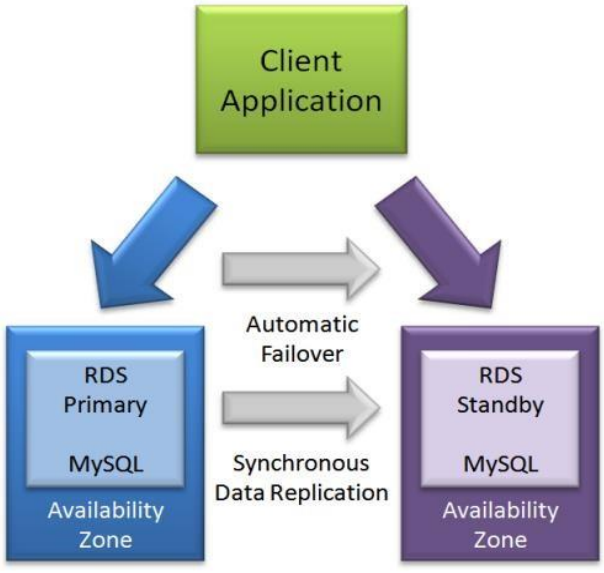

Figure 8 RDS instances (Multi A-Z deployment)

The whole concept of making RDS instance and availability zone is called as multi A-Z deployment.

Causes of database Failover

There are different situations in which the database failure occur.

- ⇒ Database crashes (primary DB fail)
- ⇒ Over load on DB
- ⇒ System goes down (due to lots of request at time)
- ⇒ Any physical damage in system such as hardware failure, network failure, etc..
- ⇒ Data corrupt from primary DB.

There are different wat to detect the failure of Multi-AZ DB instance

- ⇒ Notify via email or message that failover has occur, if and only if the Database event subscriptions can be setup.
- ⇒ View database events by using Amazon RDS console or API actions.
- ⇒ View the current state of Multi-AZ deployment by using the Amazon RDS console and API actions.

IV. Key Challenges

Here, we developed the system by using AWS cloud storage (Amazon Web Storage). There are following key challenges which we are targeting to solve.

- ⇒ **Active handover** from primary to back up database and **Reduce downtime**
- ⇒ **Reducing the complexity** and administrative overhead and privide high availiblity.
- ⇒ **Redirect client requests** to standby database (while failure occurs).
- ⇒ **Auto scaling** (Scale up the capacity to maintain performance)

while primary DB fail and secondary start working. the time in between that is called as downtime and we are working on this to reduce downtime and active handover, so there is no dataloss.

V. Tool and Technology

To implement the system, we used the following tools and technology.

We have used eclipse IDE to implement the web application. We used AWS cloud technology to provide high availability.

Bootstrap, HTML and CSS : - we used this technology to design the web pages

J2EE Frameworks

We have used **Spring MVC** which handles the client requests. The main aim of using MVC concept is to provide security. We used the **Hibernate framework** to perform different operation on database (it provides connectivity with DB). The main advantage of using hibernate is that it is working on the concept of ORM (object relational mapping) which is Database independent. In other word we can change the database engine at any time.

We have implemented the system using amazon services such as:

EC2 – it is webservice which provides scalable computing capacity in the (AWS) cloud.

RDS – it is a web service which provide different (six) database engines. The additional feature is scalability (scale a relational database engine in cloud).

VI. Design and Methodology

Here we are making a web application to demonstrate the system of high availability. The website is based on trust/ charity in which the donor can register if he or she is interested to donate some fund, study material, food, etc.. The given flow diagram shows the overall working of our system (how the client request is handle).

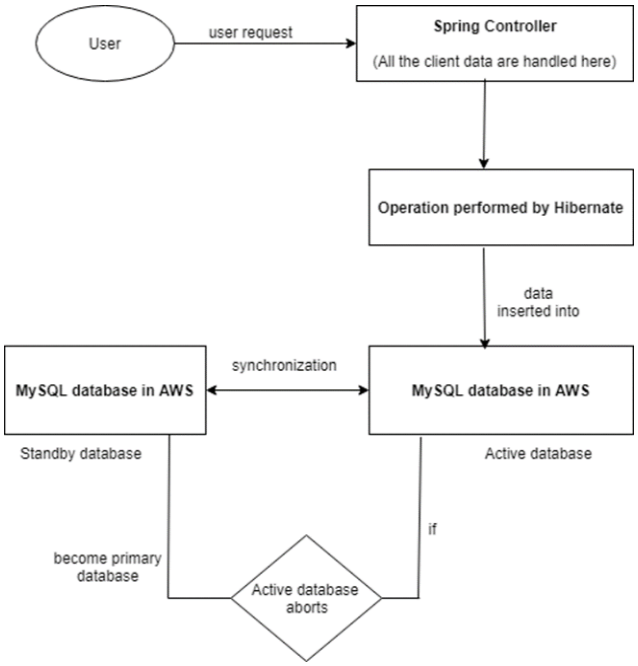

Figure 9 Flow diagram

When the user(donner) send the registration request at that time the data (such as users name, nationality, phone number, etc.) is send from web page to the spring server. The dispatcherserServlet accept that request and it will be handled by spring controller. The operation (insert, update, delete, etc.) is performed by hibernate framework. Hibernate perform the operation and fire the query according to the user's request. The result is stored in primary data base which is in MySQL (RDS instance). By default, the active (primary) data base is handles all the end user's request. If in case any failure occurs, the standby database start working as primary and there is no downtime (quickly handle all the client request. By using this system there is guarantee of no data loss and promise of high availability.

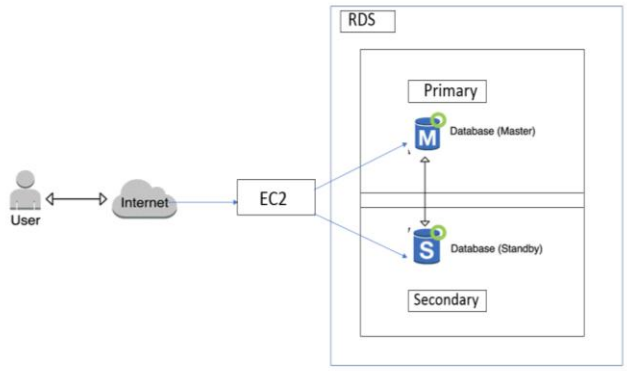

Figure 10 System flow

we can see that the Figure 8 gives the overview of the EC2 instance. It gives information regarding our instance which is Linux in our case. It also provides information regarding instance such as state, id, public IP address, network and the security group.

RDS instance provide information about it's engine type, state, class of instance, availability zone of primary and standby DB

Steps

1. Developed the web application using J2EE Framework.
2. Creating instance of Ec2 in some region and configure it by providing its network and security details and generation of key pair in .pem format.
3. Use of PuTTy tool for converting .pem key format to .ppk format so that EC2 instance can be accessed from the users computer.
4. Installation and configuration of required softwares such as java-1.8 ,tomcat-8, mysql-server in EC2.
5. Pulling of WAR file of our web app on tomcat server on our LINUX ec2 instance from github.
6. Creation of RDS instance and mirroring of that instance in some other availability zone (we used mySQL).
7. Connecting web application to the RDS instance primary database.
8. Accessing our web application using public IP address of EC2.

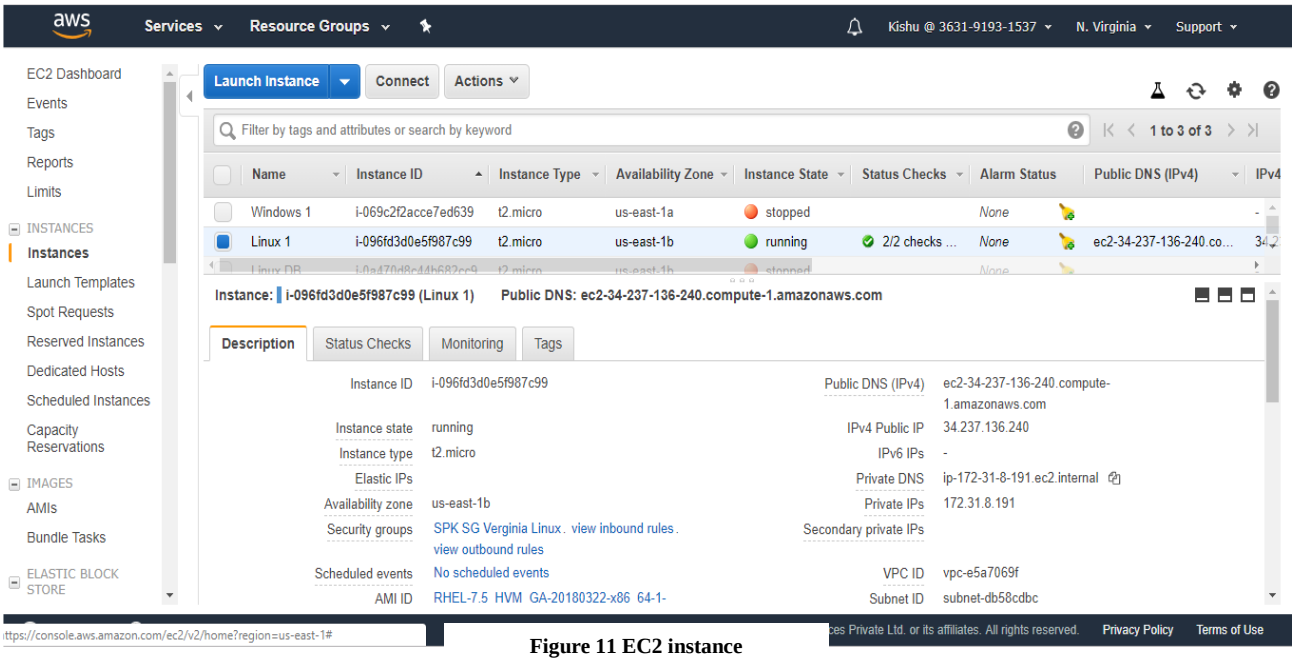

Figure 11 EC2 instance

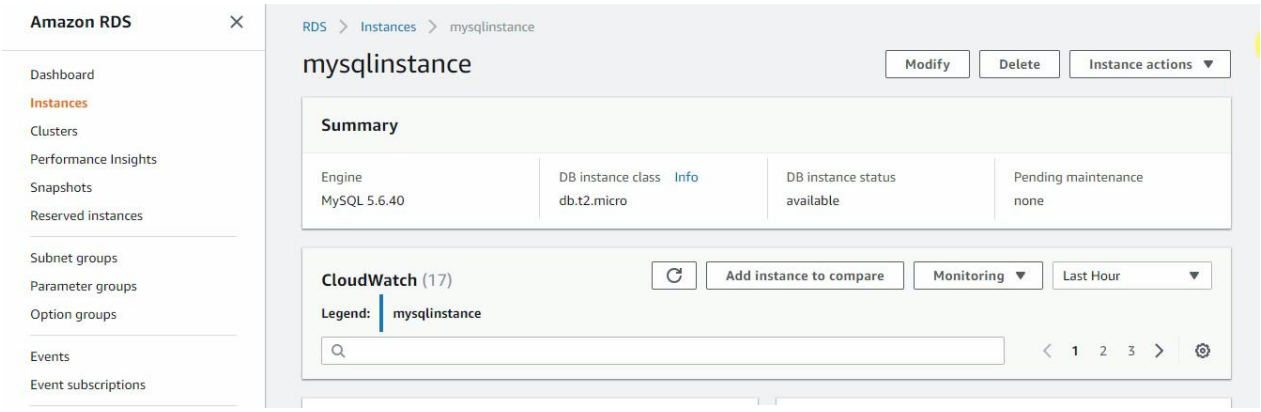

Figure 12 RDS instance

VII. Result Analysis

We can see the Result of research analysis in image

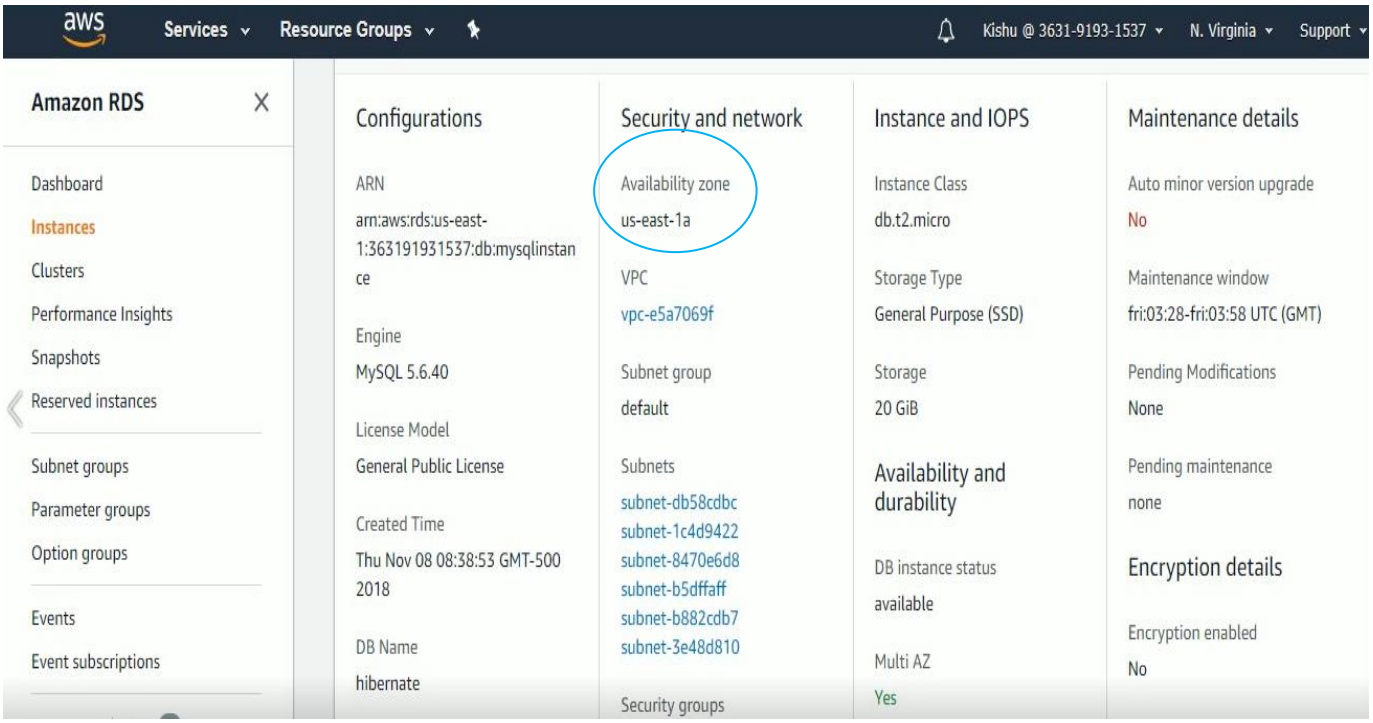

Figure 13 Availability zone before failure

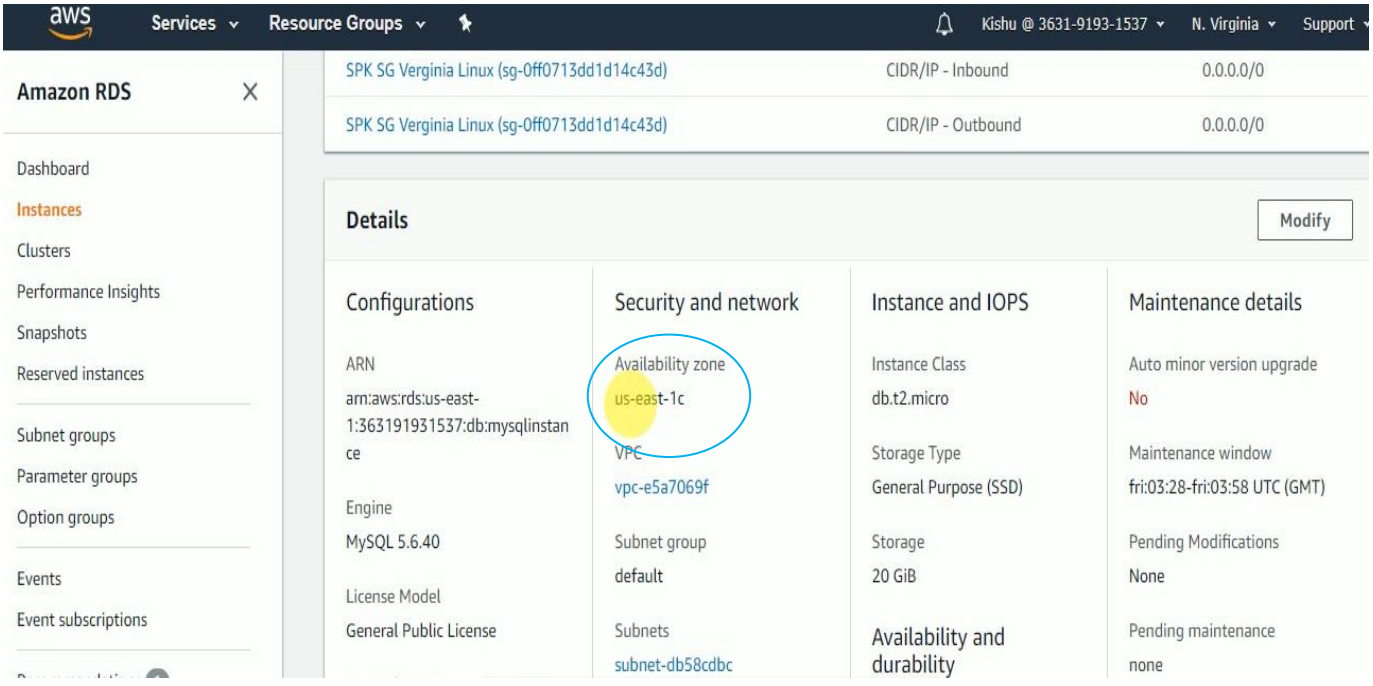

Figure 14 Availability zone after failure

Here we can see in the fig 13 that the availability zone was “us-east-1a”(it is working as primary dB). After failover occur the standby DB take that place and start working immediately. In fig 14 we can see that after failure the Availability zone change and it will be us-east-1c.

From this we can say that there will be no downtime during any failure it means no data loss. This system promises the high availability.

VIII. Conclusion And Future work

In present time, the user expect that the system must be highly available. Providing this facility is one of the challenging tasks for administrator (engineer). There are different existing technique which provide this feature such as RemusDB, SHADOW system, Postgres-R, etc. Here, we propose the system to fulfill different challenges. Our main focus was to slow different problem such as downtime, complexity of administrative overhead, Auto scaling and active handover (while redirecting client request). We have used hibernate to slow the problem of administrative overhead (hibernate is database independent and provide flexibility). We used the concept of EC2 to accomplice the challenge of auto scaling. Here we used the concept of A-Z deployment (in RDS) which solve the issues of downtime and active handover (while redirecting client request). We have used hibernate to slow the problem of administrative overhead (hibernate is database independent and provide flexibility). We used the concept of EC2 to accomplice the challenge of auto scaling. Here we used the concept of A-Z deployment (in RDS) which solve the issues of downtime and do active handover. The propose system guarantee that there will be no downtime that means no data loss and active handover. It also promises to give high availability to end users.

In future, we are going to work on fault tolerance for backend server

IX. Reference

[1] Jaemyung Kim, Kenneth Salem, Khuzaima Daudjee, Ashraf Aboulnaga, Xin Pan; "Database High Availability Using SHADOW Systems" https://cs.uwaterloo.ca/~kdaudjee/DaudjeeSoCC15

[2] U. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, A. Warfield, "RemusDB: Transparent High Availability for Database Systems," https://www.researchgate.net/publication/220538861_RemusDB_Transparent_High_Availability_for_Database_Systems

[3] Moiz, Salman Abdul, P. Sailaja, G. Venkataswamy, and Supriya N. Pal, "Database Replication: A Survey of Open Source and Commercial Tools," https://www.researchgate.net/publication/49607406_Database_Replication_A_Survey_of_Open_Source_and_Commercial_Tools

[4] RDS tutorial use to make RDS instance https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

[5] EC2 tutorial use to implement EC2 instance https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Instances.html

[6] EC2 instance in linux https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Linux-Server-EC2Rescue.html

[7] Multi A-Z deployement. https://aws.amazon.com/blogs/aws/amazon-rds-multi-az-deployment/