SPARC Group 15: Kevin Nguyen, Kyle Li, Rulin Li, Sharon Chen

AWS Data Restore

Introduction

In an data-driven era, safeguarding critical information against various threats is paramount for any organization. One strategy to protect customer data is through the implementation of data backup protocols. This documentation outlines our AWS Data Restore project, which aims to fortify customer data by establishing redundancies through the creation of multiple data restore points across regions to ensure robust data protection and recovery capabilities.

Background

Maintaining multiple backups of customer data is essential for guaranteeing data durability. This practice ensures that stored information remains intact, easily retrievable, and consistently dependable, even when confronted with potential risks, failures, or disruptions. These threats to customer data can span from malicious actors performing unauthorized alterations to the customer's database to natural disasters devastating the customer's central data center where critical information is housed. Furthermore, customers often need to adhere to strict industry regulations that require robust data retention and protection to ensure legal compliance.

To mitigate the potential impact of these threats on our customers' data and aid our customer in complying with industry regulations, our AWS Data Restore project enacts a comprehensive backup strategy for our customers' MySQL databases. Customer data will be replicated across multiple regions within the continental United States and can even be replicated in regions across the world if desired. This cross-regional configuration ensures customer data accessibility in scenarios where data centers in certain regions become inaccessible due to unforeseen server malfunctions or malicious intrusions. Customers will be able to access any of the backups stored across these regions to initiate their data restoration process. Furthermore, our project extends the capability of restoring customer data backups into a non-relational database structure, granting greater flexibility for specific customer use cases (e.g. need for flexible schema designs or unstructured data) and enhanced horizontal scalability.

Requirements

Customers should expect the AWS Data Restore project to:

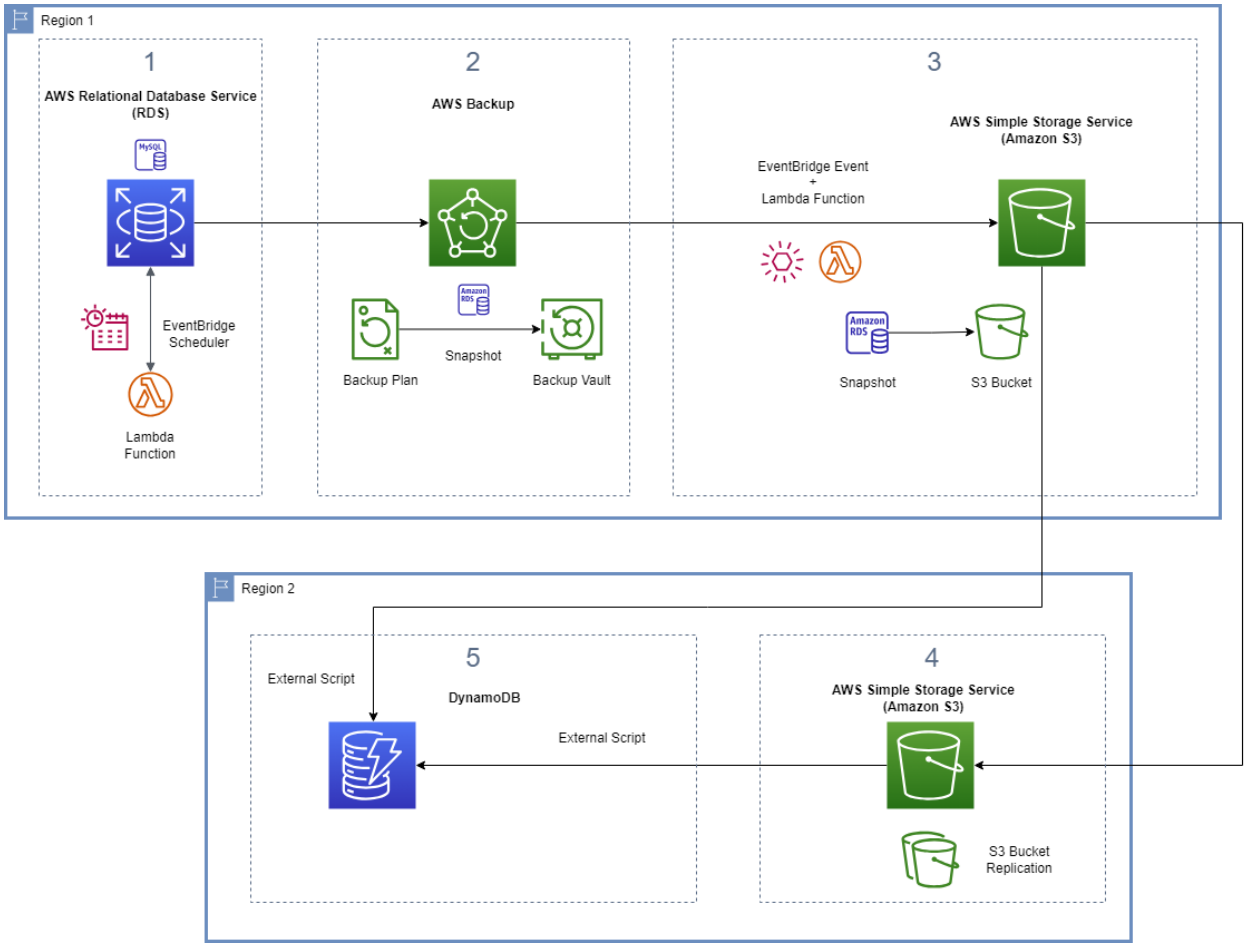
- Not modify the original structure or consent of the customer's data throughout the backup or restoration process.
- Periodically backup customer data in an AWS Relational Database Service (RDS) database and output the backed up data to a local AWS Simple Storage Service (S3) bucket.
- Periodically replicate the S3 bucket to a different region and labeled in a way such that it can be uniquely identified based on the time the backup was taken.
- Restore data from a specified backup object in any of the cross-regional S3 buckets is able to be restored in a non-relational database.

Out of Scope

Customers should NOT expect the AWS Data Restore project to:

- Handle data corruption issues that existed prior to the backup process.
- Ensure real-time updates to the backed-up data. Although the project's goal is to perform periodic backups, there may be a time delay between modifications to the database and backup creation.
- Guarantee complete global availability of backups as there may be limitations in specific regions' accessibility or regulatory requirements that prevent backups from being stored in certain geographical locations.
- Eliminate all potential instances of downtime or disturbances during the data restoration phase. While it aims for timely recovery, the project cannot provide instant retrieval or complete immunity from unforeseen challenges.
- Oversee the management of access controls or permissions for the restored data in the non-relational database.

Our Solution



1. To utilize our AWS Data Restore project, there must be an existing RDS database. The customer can either create their database through AWS RDS or they can use AWS Data Migration Service (AWS DMS) to transfer their existing database to AWS RDS.

    a. To simulate how a customer would use our project, we created a MySQL database on AWS RDS. Utilizing an API that pulls popular television show data, we populated the database with 100,871 rows of records containing information regarding television show title, air date, origin country, language, critic vote average, and vote count. To replicate real time modifications to the data, we created an AWS Lambda function that would access the database to either add a television show to the database with randomized data values, delete a random title from the database, or add individual votes to a present title, thus incrementing the vote count and altering the vote average of the title's data. This AWS Lambda function was triggered using AWS EventBridge with a scheduled expression rate of every 5 minutes to simulate a highly dynamic database that exhibits a high dependency on data durability.

2. Next, we harness AWS Backup to architect a meticulous backup strategy. This comprehensive plan orchestrates the generation of RDS backup snapshots for the

database at hourly intervals; these snapshots are preserved in a backup vault within AWS Backup to be used as designated recovery points. This methodical approach not only ensures the capture of crucial data states but also empowers streamlined retrieval options, guaranteeing the swift reconstitution of the system in the event of any operational disruptions. Furthermore, this backup strategy allows for conservation of data throughout its lifecycle, enabling turnbacks in cases where data alteration was unintentional.

3. We then utilized AWS EventBridge in order to trigger an AWS Lambda function every time a backup of the RDS database finishes creating a backup snapshot. Once a snapshot's creation is completed and stored in the AWS Backup vault, the AWS Lambda function would export the completed snapshot to an S3 bucket within the same region as the original primary database, providing another point of recovery the customer can restore their database from.

4. To bolster data redundancy and fortify data accessibility against unforeseen events, our project takes a proactive approach where the exported snapshot in the S3 bucket is replicated to a different S3 bucket situated in a distinct geographical region. This strategic safeguard not only amplifies the user's disaster recovery capabilities but also affords them the advantage of uninterrupted access to critical data. This preventative measure serves to migrate potential disruptions to their original region's data centers and underscores the commitment to maintaining seamless operations even in unexpected scenarios.

5. To conclude our final measures of data security and durability, we introduce a script that can extract snapshots from any S3 bucket, regardless of regional confines and export them to a non-relational database nestled within the AWS ecosystem: DynamoDB.

Security

To underline our commitment to process security, we strategically allocate limited permissions to customers. This deliberate approach ensures that while customers have access to data resources, their access is tightly controlled to preserve the integrity and confidentiality of the entire process. Customers should only have read access to snapshots from AWS Backup, S3 bucket, cross-region S3 buckets, and DynamoDB.

To safeguard the security of this project, we employ several effective techniques:

1. IAM Roles with Specific Policies: We utilize IAM roles meticulously configured with precise policies. This ensures that only authorized individuals can access and manipulate critical components.

2. KMS Encryption: Our approach involves the use of Key Management Service (KMS) for encrypting and decrypting sensitive data in RDS and DynamoDB. This advanced encryption mechanism enhances data protection throughout the backup and data recovery processes.

3. Environmental Variables for Secure Information Handling: Private login credentials are shielded within environmental variables in Lambda functions. This not only conceals sensitive data but also strengthens security when accessing the RDS database.

By implementing these strategies in combination, this project establishes a robust security framework that safeguards the integrity and confidentiality of the entire backup process.

Alternative Solution Considered

We considered utilizing Amazon Aurora Global Databases would allow for the creation of a read replica in a different region for disaster recovery purposes. Amazon Aurora automatically replicates data to secondary regions, reducing the need for manual intervention. However, we did not choose this solution as we wanted to explore as many AWS products as possible. Furthermore, having more control over each aspect of the backup process allows for greater flexibility and even more data restoration points the customer can choose from.

Future Planned Implementation

1. Replication of S3 buckets across AWS accounts
    a. This redundancy enhances data availability, minimizing the risk of data loss due to hardware failures, accidental deletions, or regional outages.
    b. It's important to note that while replicating S3 buckets across accounts offers these benefits, it also adds complexity to the architecture and requires careful planning and configuration with additional cost to the billing.
    c. Before implementing cross-account replication, customers should carefully assess their organization's needs, security requirements, compliance obligations, and operational considerations to determine if this approach aligns with business goals and provides the necessary benefits.
2. Data Lifecycle Management
    a. Implementation of data lifecycle policies for S3 buckets to automatically transition older snapshots to Amazon Glacier could increase cost savings and allow the option of eventual deletion after designated retention period to save storage space.

RDS vs. DynamoDB

In this project, we began with the RDS database and eventually exported data to a DynamoDB database. Specifically, we chose to create a MySQL database through RDS that offers advantages such as automated backup and recovery, as well as encryption at rest and in-transit. However,

RDS also has some disadvantages, such as a lack of direct access to SQL data, requiring users to administer the database themselves.

On the other hand, AWS has a NoSQL database service known as DynamoDB that boasts several advantages. The database service is fully managed, eliminating the need for users to maintain any underlying infrastructure. Additionally, DynamoDB's NoSQL database features a flexible schema, enables for horizontal scalability, and provides faster querying compared to SQL databases. Nevertheless, there are also drawbacks to using DynamoDB and similar NoSQL databases, including limited querying options and restricted storage capacities for items.

During our migration from an SQL database to a NoSQL solution, it's important to be mindful of potential risks in the process.One notable concern is the shift from MySQL's structured schema to the more flexible nature of a "schemaless" database. This transition could potentially introduce challenges related to maintaining consistent data types throughout the migration. Additionally, as we explore this transition, it's essential to consider the aspect of storage capacity. While RDS provides ample and scalable storage capabilities, DynamoDB imposes certain limitations on data capacity. This implies that the data that comfortably fits within RDS boundaries might require adjustments to align with DynamoDB's constraints during the migration process.

When weighing the merits between SQL RDS and NoSQL DynamoDB, customers have a range of compelling considerations that guide their decision-making. The structured nature of SQL RDS, exemplified by MySQL, provides a well-defined schema that ensures data consistency and relationships, making it an ideal choice for applications requiring complex queries and transactions. For instance, industries such as finance or healthcare, which demand rigorous data integrity and regulatory compliance, might lean towards SQL RDS due to its ability to enforce strict data validation and security protocols. On the other hand, the flexible schema of NoSQL DynamoDB accommodates dynamic and evolving data, making it a prime candidate for scenarios demanding scalability and rapid iteration. For example, ecommerce platforms with fluctuating product catalogs might favor DynamoDB's "schemaless" design to swiftly adapt to changing inventory. Ultimately, our project empowers customers with the freedom to opt for the database type that aligns most effectively with their distinct requirements.

Summary

The AWS Data Restore project offers businesses a robust and comprehensive solution for safeguarding and fortifying critical customer data in today's data-driven landscape. By implementing meticulous data backup strategies, cross-regional redundancy, and the capability to restore data into a non-relational database, the project ensures data durability, accessibility, and integrity in the face of potential threats and disruptions. With a strong focus on security through IAM roles, encryption, and environmental variables, the project establishes a secure foundation to preserve data confidentiality. Furthermore, by considering alternative solutions and planning for future enhancements like cross-account replication and data lifecycle management, the project showcases its commitment to continual improvement and adaptability. Ultimately, the AWS Data Restore project empowers businesses to navigate data uncertainties confidently, allowing them to concentrate on core objectives while knowing their critical data is well-protected and easily recoverable, fostering trust, compliance, and operational resilience.