Project Title: Cloud-based Disaster Recovery Using IBM Cloud Resiliency Orchestration

Team members : Shashi Kumar Sai G, Hemanth kumar B Patil ,Varshith ,Vineet

Note : we tried to this project with IBM cloud but we faced some issues so we have used AWS cloud for our project

IBM Cloud Object Storage (we have created buckets and uploaded Objects)

IBM Cloud Object Storage is a scalable, secure, and durable cloud storage solution designed to handle unstructured data. It provides high availability and reliability for storing backup files, disaster recovery data, and large datasets.

Key Features:

- **Scalability: Automatically scales to accommodate growing storage needs.**
- **Security: Supports encryption and access control policies.**
- **Durability: Ensures data integrity with built-in redundancy.**
- **Multi-Region Availability: Supports geo-dispersed storage for data resilience.**
- **Cost Efficiency: Offers flexible pricing based on usage and storage class.**

IBM Cloud Object Storage is often used in conjunction with disaster recovery solutions to provide an additional layer of redundancy and long-term data retention. It integrates with IBM Cloud services and supports API-based access for seamless data management

Project Overview

This project focuses on implementing a disaster recovery (DR) solution using AWS Elastic Disaster Recovery (AWS DRS). The goal is to ensure business continuity by replicating critical workloads to AWS and enabling failover in case of an outage. The project involves creating an EC2 instance as the source environment, configuring IAM policies, setting up the AWS DRS agent on a Linux machine, and performing recovery operations using recent snapshots.

Objectives

- **Implement a disaster recovery solution that ensures minimal downtime.**
- **Reduce data loss by maintaining continuous replication.**
- **Automate failover and failback processes.**
- **Optimize recovery time objective (RTO) and recovery point objective (RPO).**
- **Manage resources effectively to minimize costs after the project is completed.**
- **Enhance security by implementing IAM policies and encryption mechanisms.**

- **Ensure high availability and fault tolerance using AWS services.**

AWS Services Used

1. Amazon EC2

- **Created the source instance for replication.**
- **Used to test failover and failback scenarios.**
- **Configured appropriate instance types and monitoring settings.**

2. AWS IAM

- **Configured user roles and policies to enable secure access.**
- **Created users and generated API and secret keys for AWS DRS configuration.**
- **Implemented least privilege access to improve security.**

3. AWS Elastic Disaster Recovery (AWS DRS)

- **Installed and configured the agent on the source instance.**
- **Enabled continuous replication and created recovery instances.**
- **Managed snapshot consistency and performed regular recovery drills.**

4. Linux Machine

- **Used to download and install the replication agent.**
- **Ensured agent communication with AWS DRS.**
- **Maintained logs for troubleshooting and monitoring.**

5. Amazon S3

- **Used to store logs and backup data.**
- **Ensured that historical snapshots were available for recovery.**
- **Implemented lifecycle policies to manage storage costs efficiently.**

6. AWS CloudFormation

- **Automated deployment of infrastructure for disaster recovery.**
- **Reduced manual intervention in setting up recovery environments.**
- **Created templates for quick and repeatable DR deployments.**

7. AWS CloudWatch

- **Monitored DR metrics and application health during recovery.**
- **Configured alerts for potential issues during replication and failover.**

- **Analyzed logs to troubleshoot replication inconsistencies.**

8. AWS SNS

- **Configured to send notifications about DR events and failures.**
- **Integrated with CloudWatch for automated alerting.**
- **Enabled timely responses to potential DR incidents.**

Architecture

The architecture follows a structured approach:

1. **Source Environment: Created an EC2 instance as the primary system for replication.**
2. **AWS DRS Replication: Installed AWS DRS agent on a Linux machine, initializing replication.**
3. **Snapshot Management: AWS DRS took periodic snapshots for consistency.**
4. **Recovery Execution: Selected the latest snapshot for recovery, launching a new instance.**
5. **Monitoring and Validation: Monitored recovery jobs to ensure instance creation was successful.**
6. **Resource Cleanup: Deleted resources post-recovery to avoid unnecessary costs.**

Implementation Steps

1. Setup AWS DRS

- **Configured AWS DRS in the AWS Management Console.**
- **Installed the AWS DRS agent on a Linux machine.**
- **Verified replication status to ensure data consistency.**
- **Performed initial tests to confirm successful synchronization.**

2. Configure IAM and Networking

- **Created IAM users with necessary policies.**
- **Generated API and secret keys for AWS DRS.**
- **Configured security groups to allow controlled access.**
- **Ensured VPC configurations were optimized for DR scenarios.**

3. Replication Process

- **Monitored the AWS DRS agent as it initiated replication.**

- **Ensured that the system took snapshots at required intervals.**
- **Checked data integrity and verified consistency between source and replicated instances.**

4. Perform Failover Testing

- **Selected the most recent snapshot for recovery.**
- **Created recovery instances using AWS DRS.**
- **Validated system functionality post-recovery.**
- **Conducted simulated disaster scenarios to measure RTO and RPO.**

5. Monitor Recovery Jobs

- **Used AWS CloudWatch to monitor DR metrics.**
- **Checked job progress in AWS DRS and ensured successful recovery.**
- **Implemented logging and alerts for real-time tracking.**

6. Cleanup and Cost Optimization

- **Deleted unused resources to manage AWS costs.**
- **Disabled unnecessary replication processes after project completion.**
- **Implemented cost-saving measures such as reserved instances and S3 lifecycle policies.**

Challenges Faced

- **Initial Replication Delays: AWS DRS took some time to initialize and start replication.**
- **Snapshot Storage Costs: Regular snapshots required monitoring to control costs.**
- **Recovery Time Considerations: Ensuring minimal downtime during failover testing.**
- **Security Concerns: Managing IAM roles and preventing unauthorized access.**
- **Networking Issues: Ensuring low latency between source and recovery regions.**

Best Practices for AWS Disaster Recovery

1. **Regular Testing: Periodically test disaster recovery plans to ensure effectiveness.**
2. **Cost Management: Optimize storage and replication settings to control AWS costs.**
3. **Security Measures: Implement IAM policies to restrict access to DR resources.**
4. **Automation: Use AWS Lambda for automated failover and recovery processes.**

5. **Monitoring and Alerts: Utilize AWS CloudWatch and SNS for real-time monitoring.**
6. **Multi-Region Redundancy: Ensure disaster recovery across different AWS regions.**
7. **Data Encryption: Use AWS KMS to encrypt backups and snapshots.**
8. **Compliance and Auditing: Regularly audit DR policies for regulatory compliance.**

Conclusion

This project successfully demonstrated disaster recovery implementation using AWS Elastic Disaster Recovery. It provided hands-on experience with AWS services such as EC2, IAM, AWS DRS, and S3, ensuring a seamless failover and failback process. After the project was completed, resources were deleted to prevent unnecessary billing, highlighting the importance of cost management in cloud-based DR solutions. Security policies were optimized, and failover procedures were tested to ensure high availability and resilience.

Future Enhancements

- **Implement AWS Lambda for automated DR processes.**
- **Optimize storage strategies using Amazon S3 lifecycle policies.**
- **Enhance security using AWS WAF and AWS Shield.**
- **Test multi-region failover to ensure global availability.**
- **Use AWS Backup for additional redundancy and data integrity.**
- **Develop a detailed DR runbook for faster incident response.**
- **Explore AI-based anomaly detection to proactively identify risks.**
- **Integrate AWS Config for compliance and governance in DR setups.**