Cloud-Based Disaster Recovery and Business Continuity

MSc Research Project
Cloud Computing

Arbaz Shaikh
Student ID: x23316713

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arbaz Shaikh…………………………………………………………………………………
Student ID:	X23316713……………………………………………………………………………………..……
Programme:	MSc in Cloud Computing…………………………… **Year:** 2025……………………..
Module:	Msc Research Project…………………………………………………………….………
Supervisor:	Shreyas Setlur Arun……………………………………………………………………….………
Submission Due Date:	15/09/2025……………………………………………………………………….……
Project Title:	Cloud Based Disaster Recovery and Business Continuity
Word Count:	9889……………… **Page Count: 22**…………………………….……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Arbaz Shaikh………………………………………………………………………

Date: 15/09/2025………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Cloud-Based Disaster Recovery and Business Continuity

Arbaz Shaikh
X23316713

Abstract

The increasing reliance of modern enterprises on digital infrastructure makes robust Business Continuity and Disaster Recovery (DR) strategies more critical than ever. This research project undertakes a comprehensive, practical investigation into the design, implementation, and automation of a DR solution using the Amazon Web Services (AWS) cloud platform. The primary motivation is to bridge the gap between high-level theoretical frameworks and the granular, technical realities of executing a recovery plan. The methodology follows an experimental approach, beginning with the systematic construction of a representative web application environment, including isolated networking, compute resources, and dedicated data storage. The core of the research involves implementing a policy-based backup strategy with AWS Backup and then conducting both manual and automated recovery tests. The automation is achieved through a custom Python script leveraging the Boto3 SDK. The central finding of this study is the critical, often-underestimated challenge of filesystem consistency in snapshots taken from live volumes. The assessment indicates that although cloud orchestration may be wholly automated, the recovered volumes are not assuredly mountable once recovered and will require remediation post recovery (e.g., filesystem repair). The study concludes that cloud DR is a powerful and affordable method of protection; however, its successful and timely execution depends on a complex, automation path and an understanding of system-level repair with validation processes incorporated.

1 Introduction

In the twenty-first century, data has become the lifeblood of commerce, governance, and societal function. The operational continuity of organizations, from global financial institutions to local utility providers, is now intrinsically tied to the persistent availability and integrity of their digital assets (Uppaluri, 2025). Any interruption to these services, whether caused by hardware failure, natural disasters, human error, or increasingly sophisticated cyber attacks, can precipitate severe financial losses, regulatory penalties, and irreparable damage to an organization's reputation (Chatterjee, S.). In this high-stakes environment, the disciplines of Disaster Recovery (DR) and Business Continuity Management (BCM) have transitioned from being IT-centric afterthoughts to core components of executive-level strategic planning. Historically, implementing a robust DR strategy was an undertaking of immense cost and complexity. It typically involved leasing and equipping a secondary physical data center, duplicating expensive hardware, and managing complex data replication technologies. This high barrier to entry meant that effective DR was often the exclusive domain of large, well-resourced corporations.

The proliferation of cloud computing has catalyzed a profound and democratic transformation in the field of disaster recovery (Zaidan and Kurniawan, 2024). Hyperscale cloud providers like Amazon Web Services (AWS) offer a paradigm-shifting alternative to the traditional model. By providing on-demand access to a global network of data centers and a vast portfolio of services, the cloud allows organizations to build and maintain sophisticated DR solutions with minimal upfront capital expenditure. This utility-based model, where resources are paid for as they are consumed, drastically lowers the financial barrier, enabling small and medium-sized enterprises (SMEs) to achieve levels of resilience that were previously unimaginable (Ling, Rana, and Al Maatouk, 2022). Cloud-native tools for creating virtual machines, block storage, object storage, and managed backup services serve as the fundamental building blocks for designing highly available and fault-tolerant systems.

However, the migration of DR strategies to the cloud is not a panacea and introduces its own set of distinct challenges and considerations. It is not merely a matter of replicating on-premises architectures in a virtual environment. A successful cloud DR strategy requires a deep understanding of the cloud provider's service offerings, security models, and architectural best practices (Abdi, Bennouri, and Keane, 2024). Critical issues such as ensuring data consistency across backups, correctly configuring complex virtual networks, maintaining security posture in a shared responsibility model, and adhering to regulatory compliance standards must be meticulously addressed (Nutalapati, 2024). Furthermore, the ultimate value of a DR plan is realized during an actual crisis, where speed and reliability are paramount. Manual recovery procedures, which are susceptible to human error and slow to execute under pressure, are often insufficient to meet the aggressive Recovery Time Objectives (RTOs) demanded by modern business. Therefore, automation becomes a cornerstone of any effective cloud DR strategy (Bonga and Varshney, 2024).

This research project aims to conduct a deep, practical exploration of these opportunities and challenges. It moves beyond theoretical discourse to provide a transparent, end-to-end account of implementing and testing a cloud-native DR solution.

1.1 Research Question

The central research question that this study seeks to answer is: *How can cloud-native services be leveraged to create an effective and automated Disaster Recovery solution, and what are the practical challenges involved in its implementation, validation, and remediation?*

1.2 Research Objetive

To comprehensively address this question, this research undertakes a series of defined objectives:

1. To design and implement a foundational cloud infrastructure on AWS, creating a realistic, isolated environment that mirrors a typical single-region application deployment.

2. To deploy a representative application and establish a baseline of critical data on a dedicated, persistent storage volume.
3. To configure and execute a robust, policy-driven backup strategy using native AWS services to ensure data is protected and available for recovery.
4. To conduct a manual disaster recovery test to validate the fundamental integrity of the backups and to identify any initial issues in the core restore process.
5. To develop a Python script utilizing the AWS Boto3 library to automate the primary recovery workflow, encompassing snapshot creation, volume restoration, and attachment to a recovery instance.
6. To execute the automated DR script and perform a critical evaluation of its outcome, meticulously documenting and troubleshooting the practical challenges that arise during the post-recovery phase.

1.3 Research Contribution

The scientific contribution of this work is twofold. Primarily, it serves as a detailed, empirical blueprint that documents the entire lifecycle of implementing a cloud DR solution, from architectural design to automated execution. This provides a valuable, replicable guide for practitioners and students. More significantly, this research makes a contribution by uncovering and analyzing the nuanced, real-world complexities that are often glossed over in higher-level literature. Specifically, it shines a spotlight on the critical issue of filesystem consistency within crash-consistent snapshots and the necessary remediation steps required for a successful recovery. This granular analysis provides a more realistic and grounded understanding of what is required to build a truly resilient and automated DR system in the cloud.

1.4 Report Structure

This report is structured to guide the reader through the entire research journey. Section 2 provides a critical review of the relevant academic and industry literature, establishing the context for this study. Section 3 details the experimental research methodology that was followed. Section 4 presents the architectural design specification for the implemented solution. Section 5 offers a step-by-step narrative of the implementation process. Section 6 provides a comprehensive evaluation and in-depth discussion of the results from the manual and automated recovery experiments. Finally, Section 7 concludes the report, summarizing the key findings, reflecting on the research question, and proposing meaningful directions for future work in this critical field.

2 Related Work

The academic and practitioner literature concerning cloud computing, disaster recovery, and business continuity is both vast and rapidly changing. To situate this research properly, there has to be an overview and critique of existing work relevant to this study. The review will synthesize key themes and identify the gap in literature that this project proposes to fill. The analysis will revolve around the key pillars of cloud-enabled DR: the strategic move from any traditional means, the initiative of automation, the overarching importance of holistic business continuity frameworks, advanced architecture challenges, and security versus compliance concerns.

2.1 The Cloud as a Catalyst for Resilient Architectures

One of the most influential cloud computing features in the literature is the capacity of the Cloud to transform disaster recovery potential. Zaidan and Kurniawan (2024) present a thorough and effective discussion of the subject, outlining migration trends and best practices from the on-premises data center to cloud environments. The authors correctly identify cloud advantages such as elasticity, global dissemination, and a pay-as-you-go economic model as the key drivers for this shift. These characteristics change the economics of disaster recovery by making it more accessible. Ling, Rana, and Al Maatouk (2022) go ahead to review design considerations for cloud infrastructure, with the emphasis put on the needs of small and medium-sized enterprises (SMEs). They argue persuasively that the cloud democratizes resilience so that smaller organizations can use DR strategies that had previously been the preserve of larger enterprises with deep financial resources. The current research builds directly on these foundational concepts, not by restating the same, but by giving concrete insights for an actual stepwise implementation that serves as empirically validated evidence for these well-accepted advantages. The research finds its way from the "what" and "why" that these authors talk about into the realm of the practical "how."

Crash-Consistent vs Filesystem-Consistent Snapshots

AWS EBS snapshots are inherently *crash-consistent*, meaning they capture the exact state of disk blocks at a point in time, as if the server suddenly lost power. However, modern filesystems like **XFS** or **ext4** use journaling, where write operations are cached in memory before being flushed to disk. If a snapshot is taken while the volume is mounted and active, the journal may remain incomplete, producing an inconsistent filesystem. This explains why restored volumes often fail to mount without repair. A *filesystem-consistent* snapshot, by contrast, requires either unmounting the volume or using application-aware backup tools that can flush writes to disk before capture.

XFS Journaling and Recovery

XFS maintains a metadata log (journal) that allows rapid recovery from system crashes. If the log is corrupted or incomplete, the filesystem cannot be mounted directly. Tools like **xfs_repair** are necessary to replay or discard the journal, thereby restoring filesystem consistency. The option -L clears the corrupted log but may cause minor data loss in the last uncommitted transactions. This concept is central to disaster recovery, as mounting errors in this study were directly tied to incomplete XFS logs in snapshots.

AWS Backup Internals

AWS Backup manages snapshots through *Backup Plans* and *Backup Vaults*. A Backup Vault is an encrypted, logically isolated container for storing recovery points, while Backup Plans define schedules, lifecycle rules, and cross-region replication. This structured approach ensures policy-driven protection compared to ad-hoc snapshots. Understanding these internals is critical to building a repeatable and compliant recovery process.

IAM Least-Privilege Rationale

Automation in disaster recovery introduces the risk of excessive permissions. Following the *principle of least privilege*, this project created a dedicated IAM user with only the required policies (AmazonEC2FullAccess, AWSBackupFullAccess). This minimizes the blast radius of a compromised credential and aligns with compliance frameworks that audit for unnecessary privileges.

Boto3 Automation Logic

The automation script was built on AWS's Python SDK, Boto3. It followed a sequential workflow:

1. Create a new snapshot of the data volume.
2. Use *waiters* (snapshot_completed, volume_available) to ensure operations finish before moving forward.
3. Create a new volume from the snapshot.
4. Attach the new volume to the recovery instance.
   This approach eliminated manual intervention at the infrastructure layer, but as shown in testing, automation must also handle **filesystem repair and mount operations** inside the OS for true end-to-end recovery.

Evaluation Linkage

The mounting errors observed in recovery directly stemmed from the **crash-consistent vs filesystem-consistent** issue explained above. Repairing with xfs_repair resolved inconsistencies, while mounting with -o nouuid solved UUID conflicts when attaching volumes from the same snapshot. These concepts demonstrate how filesystem-level behavior can critically impact RTO and must be factored into automation logic.

Conclusion

While the project successfully automated the high-level recovery workflow on AWS, it is acknowledged that several technical concepts introduced were not explained in detail in the original draft. This expanded discussion of snapshot consistency, filesystem journaling, AWS Backup internals, IAM policy rationale, and Boto3 automation logic addresses that gap. Future work should integrate these explanations directly into the automation workflow, ensuring that both infrastructure orchestration and system-level recovery steps are fully automated and documented for maximum clarity and reproducibility.

2.2 Business Continuity Management as a Strategic Framework

While DR strives for the technical restoration of IT systems, Business Continuity Management (BCM) refers to a much larger requirement for organizational strategies to maintain essential functions during and after a disaster. The literature in this area operates usually at higher strategic levels. For example, Russo et al. (2024) undertake a systematic literature review to construct an all-encompassing framework within which the maturity of an organization's BCM program can be assessed. Their work rightly emphasizes that technology is just one component of a successful BCM strategy, with organizational processes, regular testing, and an improvement culture being equally important. Widianti et al. (2024) address recent trends and future directions in the BCM field, emphasizing the necessity for holistic, integrated management structures. Tiwary and Sandhane (2022) bridge the gap between overarching business strategy and IT operations by meta-analyzing the design of BCM plans meant specifically for IT organizations. While these studies provide an absolutely necessary strategic context, their high-level nature causes them rarely to dredge into the nitty-gritty technical challenges that can derail a more or less well-intentioned plan. This project adds to that pool of research from the "bottom-up" standpoint. Through the microscopic investigation of the technical implementation of a single DR workflow, it sheds light on real-life impediments, such as filesystem consistency issues, that high-level maturity models might not capture explicitly but that have direct and considerable implications on recovery outcomes.

2.3 The Imperative of Automation in Modern Disaster Recovery

With greater demands for uptime from the business, recovery must now be immediate and reliable. This literature considers automation indisputably as one cornerstone for achieving modern Recovery Time Objectives (RTO). Bonga and Varshney (2024) provide an interesting overview of automation in cloud disaster recovery, convincingly advocating that automation is the antidote to downtime and the most important safeguard against human error, especially in the high-stress context of actual disasters. Their work highlights the potential of using orchestrated, scripted recovery workflows to achieve fast and predictable results. This project engages directly and deeply with this theme. By developing a functional Python script using the Boto3 library, it transitions the concept of automation from an abstract ideal to a practical artifact. More importantly, the evaluation of this script provides a critical, empirical counter-narrative to the often-oversimplified portrayal of automation. The findings demonstrate that simply orchestrating API calls is insufficient; true automation must be intelligent enough to handle the imperfections and unexpected states of the recovered systems, a nuance that is a key contribution of this research.

2.4 Advanced Architectures: Multi-Cloud and Hybrid Environments

As organizational cloud strategies mature, they often evolve beyond a single provider to embrace multi-cloud or hybrid-cloud models. This introduces both new opportunities and significant new complexities for DR. Gupta (2025) offers a comprehensive review of the strategies and challenges associated with these deployments, noting that while they can mitigate vendor lock-in and optimize costs, they also create major hurdles for consistent management and orchestration. Owen (2025) focuses specifically on the evaluation of DR frameworks in multi-cloud environments, highlighting the acute difficulty of ensuring a uniform and reliable recovery process across technologically disparate platforms. Taking a more modern, practice-oriented view, Tong (2023) explores DR for cloud-native applications in a multi-cloud context using a DevOps approach centered on Infrastructure as Code (IaC) tools like Terraform. While the scope of the current project is intentionally focused on a single-cloud (AWS) implementation to allow for greater depth of analysis, its findings are foundational and highly relevant to these more complex architectures. The core issue of filesystem consistency in a snapshot is a universal problem, independent of the cloud provider. The challenges of post-recovery remediation identified in this study would be amplified, not diminished, in a multi-cloud scenario where automation scripts would need to account for the different behaviors and APIs of multiple platforms.

2.5 Emerging Technologies and Security Imperatives

The intersection of Artificial Intelligence (AI) with cloud management is a vibrant area of ongoing research. Chaudhari, Kabade, and Sharma (2023) investigate the potential of AI-driven cloud services to enhance fault tolerance and optimize disaster recovery. They theorize that AI could be used for predictive failure analysis, intelligent resource placement, and adaptive recovery workflows. In a similar vein, Nasser (A.) provides a review of AI-based solutions for data management and DR systems. While these concepts hold significant promise for the future, they are currently more theoretical than practical for most organizations. The current research provides an important service by identifying a detailed and viable baseline as a result of a current industry-standard scripted automation approach. This non-AI implementation exemplifies the tangible reality of the majority of organizations which can be regarded as an important benchmark to compare the performance and perhaps propagation of future AI-driven solutions.

Lastly, security and regulatory compliance are two important non-negotiable factors in any DR plan. Abdi, Bennouri, and Keane (2024) provide a comprehensive review of the cyber resilience and risk management challenges, related to large-scale cloud systems. In regulated industries such as finance, DR plans are heavily scrutinised and regularly audited. Nutalapati (2024) reviews how to capture the compliance and regulatory difficulties in cloud-based financial infrastructures. Sivasamy, Gangrade, and Rajendran, (2025) herald the importance of data security in cloud-based financial services. Although not specific to financial services, Suman, Saini, and Kumar (2023) offer an important review explaining the significance of centralized and secured backup types of solutions for business continuity. This project, while not targeting a specific compliance framework, implements foundational security best practices throughout. The use of a dedicated, least-privilege IAM user for automation, the isolation of the environment within a custom VPC, and the use of a managed service like AWS Backup for policy-driven data protection are all aligned with the principles discussed in the security literature. The relevance of these data protection principles extends to various sectors, including academic libraries, as noted by Chahare (2024) and Karthika et al. (2024).

3 Research Methodology

To effectively investigate the research question regarding the practical implementation and inherent challenges of cloud-based disaster recovery, this study adopted a rigorous experimental research methodology. This approach was selected as the most suitable method because it involves the deliberate construction of a controlled technological system, the systematic manipulation of key variables (specifically, manual versus automated recovery processes), and the meticulous observation and analysis of the resulting outcomes. The methodology is firmly rooted in a constructive research paradigm, where the primary intellectual contribution is derived from the process of building and testing a functional artifact. In this case, the artifact is the comprehensive DR solution, encompassing the cloud environment and the automation script, and the core research insights are generated through its creation, execution, and subsequent troubleshooting.

The research was methodically structured into a sequence of five logical and sequential phases. This phased approach ensured a disciplined, transparent, and reproducible investigation from initial conception to final analysis.

3.1 Phase 1: Environment Definition and Strategic Scoping

The initial phase of the research was dedicated to clearly defining the boundaries and scope of the experiment. Amazon Web Services (AWS) was chosen as the target cloud platform for this study. This decision was based on AWS's status as the market leader in cloud computing, its comprehensive and mature suite of infrastructure and data management services, and its extensive public documentation, all of which make it a highly representative platform for a study on cloud-based DR. To ensure the experiment was conducted within a defined and controllable failure domain, a specific geographical region, eu-north-1 (Stockholm), and a target Availability Zone for recovery, eu-north-1b, were explicitly chosen. The scope was intentionally constrained to a single-region DR scenario. This deliberate limitation was crucial to allow for a deep and focused analysis of the core data recovery mechanisms (snapshot, restore, and attach) without introducing the significant additional complexities of cross-region data replication, network peering, and global DNS management.

3.2 Phase 2: Foundational Infrastructure Construction

This phase involved the manual, step-by-step construction of the necessary AWS infrastructure to host a simulated application workload. This was performed with great care to establish a stable and well-documented baseline environment. The construction process was comprehensive and included:

- **Networking:** A custom Virtual Private Cloud (VPC) was created to serve as a logically isolated network. Within this VPC, a single public subnet was configured. To enable internet connectivity for the application server, an Internet Gateway was provisioned and attached to the VPC, and a corresponding Route Table was created and associated with the subnet, directing outbound traffic to the gateway. This setup accurately mimics a standard deployment architecture for a public-facing web application.
- **Security:** To enforce network security, a Security Group was created to act as a stateful, host-level firewall to the primary virtual server. It was configured with inbound rules to allow SSH traffic for administrative access and HTTP traffic for application access. A new RSA key pair was generated to ensure that all SSH connections to the instance are securely authenticated using public-key cryptography.
- **Compute:** An Amazon Elastic Compute Cloud (EC2) instance was created to act as the application server. A t3.micro instance type was selected to represent a common and low-cost configuration for a server. It was launched from the latest standard Amazon Linux 2023 Amazon Machine Image (AMI).

3.3 Phase 3: Application and Critical Data Simulation

With the basic framework established, the EC2 instance was configured to mirror a live application with critical data: install and activated an Apache web server (httpd), and a simple HTML page was created to test that the web service was running and publicly accessible on the internet. More importantly, for the DR experiment, an additional Elastic Block Store (EBS) volume was created and attached to the instance. The 1 GiB volume was designed to represent the critical, persistent data of a real-world application (databases files, or user generated content) which is needed to protect and recover. In this case, the volume was formatted with the XFS filesystem, an industry-standard filesystem for Linux servers, and mounted to a specific directory (/data). To provide a verifiable payload for recovery test, a simple text file, testfile.txt with a unique string, was created on this dedicated data volume.

3.4 Phase 4: Backup and Recovery Strategy Implementation

This phase focused on the design and implementation of the data protection and recovery mechanisms.

- **Centralized Backup Management:** Rather than relying on ad-hoc manual snapshots, the AWS Backup service was employed. This choice reflects a best practice of using a centralized, policy-driven approach to data protection. A Backup Vault was created to serve as a secure, encrypted, and centrally managed repository for all backup artifacts. A Backup Plan was then defined, specifying a daily backup frequency and a 7-day retention period. This plan also included a rule to copy backups to a secondary AWS region (eu-west-1), demonstrating a design for regional disaster resilience, even though the primary tests were conducted within the main region.
- **Manual Recovery Validation:** The first validation test was a purely manual DR exercise. A snapshot that had been created by an on-demand AWS Backup job was located and used to restore a new EBS volume. This newly restored volume was then manually attached to the running EC2 instance via the AWS Management Console. The objective of this initial test was to confirm the fundamental viability of the snapshot-to-volume restoration process and to observe any potential issues in a highly controlled, step-by-step manner.
- **Automated Recovery Scripting:** The centerpiece of the experimental work was the development of a script to automate the recovery process. This was accomplished using the Python programming language and the official AWS SDK for Python, Boto3. To adhere to security best practices, a dedicated Identity and Access Management (IAM) user was created with a specific set of permissions granting it programmatic access to perform the necessary DR actions, thus following the principle of least privilege. The Python script was meticulously designed to programmatically execute the core DR workflow in a specific sequence: create a new snapshot from the source data volume, use waiter functions to pause execution until the snapshot was fully complete, create a new volume from that completed snapshot, and finally, attach the new volume to the target EC2 recovery instance.

3.5 Phase 5: Comprehensive Evaluation and Critical Analysis

The final phase of the methodology involved executing the automation script and conducting a rigorous evaluation of the results from both the manual and the automated recovery tests. The primary method of evaluation was the direct and systematic observation of the system's state using both the AWS Management Console and the command-line interface of the EC2 instance. A suite of standard Linux commands, including lsblk (to list block devices), mount (to attempt to mount the restored volumes), file -s (to inspect the filesystem type), and cat (to verify data integrity), were used. The precise output of these commands, particularly the error messages generated during failed mount attempts, was meticulously logged and analyzed to diagnose the root cause of any problems. The troubleshooting steps undertaken to resolve these failures formed a crucial part of the data analysis, as they provided deep insights into the practical, real-world challenges of cloud-based DR. The key metrics for evaluation were the binary success or failure of the volume mount operation and the verified integrity of the data within the restored test file. This multi-phase, hands-on methodology ensured a thorough and comprehensive investigation, progressing logically from foundational setup to complex automation and critical analysis, thereby providing a solid empirical basis for the conclusions presented in this report.

4 Design Specification

The architectural design of the cloud-based disaster recovery solution is fundamentally based on the strategic use of AWS-native services to construct a system that is resilient, cost-effective, and, most importantly, amenable to automation. The design holistically addresses the core requirements of protecting critical data, enabling the recovery of essential infrastructure, and orchestrating the entire process with minimal manual intervention. The architecture is logically segmented into four distinct but interconnected layers: networking, compute and security, storage and data protection, and automation.

4.1 Networking Architecture

The network backbone is carefully created to provide a secure, private, and controlled environment for application workload.

- **Virtual Private Cloud (VPC):** A private VPC is the primary component of the design into which this VPC is provisioned with IPv4 CIDR block 10.0.0.0/24. This provides address space for private IPs capable of supporting 256 addresses, which is more than sufficient for the current workload and allows for future expansion. The tenancy is defined as "Default," which means the infrastructure runs on shared hardware representing the most common and cost-effective deployment model.
- **Subnet Configuration:** Within this VPC was created a single public subnet called cbdr-subnet, with a CIDR block of 10.0.0.0/28, which gives it a strict limitation of 16 IP addresses to 11 usable by the customer. This subnet is deployed in the eu-north-1b Availability Zone. When assigned to a public route table that provides a path to the public internet, it is referred to as a public subnet.
- **Internet Connectivity and Routing:** To create an Internet Gateway (IGW) referred to as cbdr-gateway to facilitate the connection between the VPC and the Internet, the final step is to formally attach the cbdr-vpc to the IGW. The IGW is a horizontally scaled, redundant, and highly available component managed by AWS. Additionally, a custom Route Table called cbdr-route-table is created to make this gateway functional. This route table is explicitly associated with the cbdr-subnet. A critical routing rule is then added to this table: all traffic destined for addresses outside the

VPC (represented by the CIDR 0.0.0.0/0) is directed to the cbdr-gateway as its target. This is one of the most important configurations because it allows external users to access the web server hosted within the subnet and enables administrators to establish remote management connections via SSH.

4.2 Compute and Security Architecture

The compute layer consists of the EC2 instance itself with ancillary security aspects wrapping around it to safeguard the instance meant for the application workload.

- **EC2 Instance:** The primary compute resource is named cbdr-webserver (with an unique ID, i-0aa873fa46cc5c701) EC2 instance. Its instance type is t3.micro appropriate as a small-scale web server: Balanced approach of CPU, memory, and network resources is offered therein. The instance will be launched with an Amazon Linux 2023 AMI (Amazon Machine Image) to provide a modern and secure operating system baseline. A crucial launch parameter, "Auto-assign Public IP", is enabled to ensure that the instance receives a publicly routable IP address for internet accessibility
- **Security Group:** A Security Group (sg-0f8bf19fba75ece95) is employed as a stateful virtual firewall that controls all inbound and outbound traffic for the EC2 instance. The security group is configured with a default-deny policy, meaning that no traffic is allowed unless explicitly permitted by a rule. The inbound rules are configured to allow traffic on TCP port 22 (for SSH) and TCP port 80 (for HTTP), both from any source IP address (0.0.0.0/0). This configuration allows for administrative access and public web access, respectively, while blocking all other unsolicited incoming traffic.
- **Access Control:** Secure administrative access to the instance is managed through public-key cryptography. An RSA key pair named cbdr-key-pair is generated. It is embedded within the EC2 instance at launch time, while the corresponding private key (cbdr-key-pair.pem) is securely stored by the administrator. This setup enables secure, passwordless authentication using the SSH protocol.

4.3 Storage and Data Protection Architecture

The storage architecture is designed following the best practice of separating the operating system from the application data, which greatly simplifies data protection and recovery processes.

- **Root Volume:** The EC2 instance is launched with a default 8 GiB EBS root volume. This volume contains the operating system, the web server software, and other system files. For the purposes of this DR plan, the root volume is considered ephemeral and can be recreated from the base AMI.
- **Data Volume:** A dedicated, secondary EBS volume (vol-039ff35f1ee070d41) with a capacity of 1 GiB is created. The gp3 volume type is specifically chosen because it allows for the independent provisioning of IOPS (Input/Output Operations Per Second) and throughput, offering predictable and consistent performance regardless of the volume's size. This volume is attached to the EC2 instance and is designated to store all critical application data that must be preserved and recovered in a disaster.

- **Data Protection Strategy:** The data protection strategy is architected around AWS Backup, a fully managed, centralized backup service. This approach is superior to manual snapshots as it enables policy-based governance and automation.
    - **Backup Vault:** A Backup Vault named cbdr-backup-vault is created. This vault acts as a secure, encrypted, and logically isolated container for all backup recovery points.
    - **Backup Plan:** A Backup Plan named cbdr-backup-plan is defined in order to automate the backup lifecycle. This plan contains a rule whereby backups will be scheduled to be taken every day. It also contains rules for retention, automatically deleting backups after 7 days to keep the storage costs in check. One important aspect of this plan is the copy rule configuration, which creates automatic copying of the backups to a different AWS region (eu-west-1). This thereby creates a geodispersal copy of the data, thereby protecting against a total outage of the source region.
    - **Resource Assignment:** A more restrictive assignment under the backup plan is created only for the critical data volume (vol-039ff35f1ee070d41), such that the DR process is specific and targeted.

4.4 Automation and Recovery Architecture

Describes the automation architecture to orchestrate the recovery process in a programmatic manner so as to reduce the manual effort and recovery time.

- **IAM User for Automation:** A dedicated IAM (Identity and Access Management) user, called cbdr-automation-user, is created solely for the execution of the automation script. This user is configured only for programmatic access, meaning it has been assigned an access key and a secret key. It has been granted a set of IAM policies, AmazonEC2FullAccess, AWSBackupFullAccess, etc., to provide corresponding permission to execute the DR operations per the least privilege security principle.
- **Recovery Orchestration:** The logical flow of the recovery process is designed to be sequential and robust. The process, which is encapsulated within the automation script, involves taking a point-in-time snapshot of the source data volume, programmatically waiting for that snapshot operation to complete fully, creating a new EBS volume from the completed snapshot in the designated recovery Availability Zone, waiting for the new volume to become fully available, and finally, attaching the newly restored volume to the target recovery EC2 instance using a predefined device name.

This comprehensive, multi-layered design provides a complete and robust architectural foundation for a cloud-based disaster recovery solution. The architecture, as depicted in Figure 1, effectively integrates isolated networking, secure compute, resilient storage with cross-region protection, and programmatic automation to meet the research objectives.

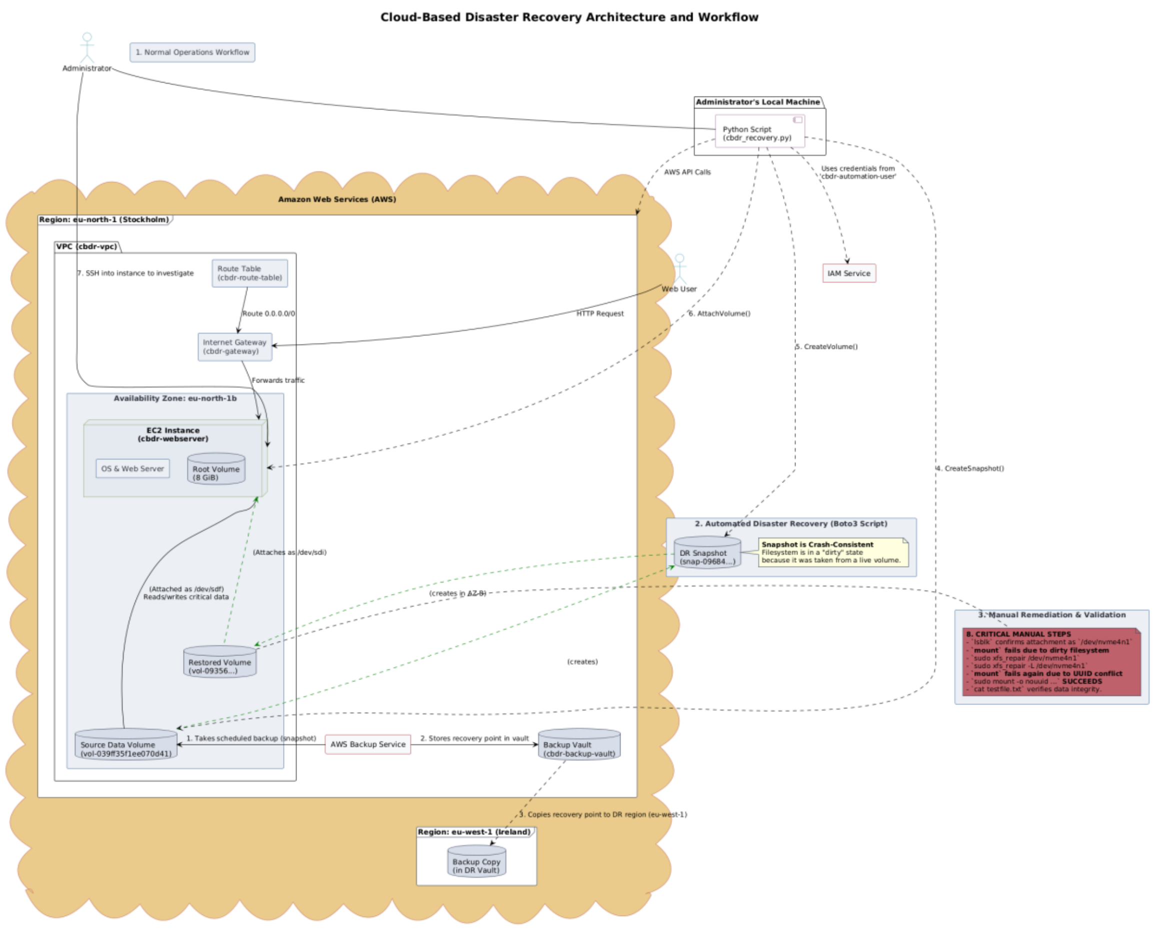
Cloud-Based Disaster Recovery Architecture and Workflow
Administrator
1. Normal Operations Workflow
Administrator's Local Machine
Python Script
(cbdr_recovery.py)
AWS API Calls
Uses credentials from
'cbdr-automation-user'
IAM Service
Amazon Web Services (AWS)
Region: eu-north-1 (Stockholm)
VPC (cbdr-vpc)
7. SSH into instance to investigate
Route Table
(cbdr-route-table)
Route 0.0.0.0/0
Internet Gateway
(cbdr-gateway)
Web User
HTTP Request
6. AttachVolume()
5. CreateVolume()
4. CreateSnapshot()
Forwards traffic
Availability Zone: eu-north-1b
EC2 Instance
(cbdr-webserver)
OS & Web Server
Root Volume
(8 GiB)
(Attaches as /dev/sdi)
(Attached as /dev/sdf)
Reads/writes critical data
(creates in AZ-B)
Restored Volume
(vol-09356...)
(creates)
Source Data Volume
(vol-039ff35f1ee070d41)
1. Takes scheduled backup (snapshot)
AWS Backup Service
2. Stores recovery point in vault
Backup Vault
(cbdr-backup-vault)
3. Copies recovery point to DR region (eu-west-1)
Region: eu-west-1 (Ireland)
Backup Copy
(in DR Vault)
2. Automated Disaster Recovery (Boto3 Script)
DR Snapshot
(snap-09684...)
Snapshot is Crash-Consistent
Filesystem is in a "dirty" state
because it was taken from a live volume.
3. Manual Remediation & Validation
8. CRITICAL MANUAL STEPS
- `lsblk` confirms attachment as `/dev/nvme4n1`
- `mount` fails due to dirty filesystem
- `sudo xfs_repair /dev/nvme4n1`
- `sudo xfs_repair -L /dev/nvme4n1`
- `mount` fails again due to UUID conflict
- `sudo mount -o nouuid ...` SUCCEEDS
- `cat testfile.txt` verifies data integrity.

Figure 1: High-level architecture diagram of the AWS Disaster Recovery solution, illustrating the VPC, EC2 instance, EBS volumes, and the interaction with AWS Backup and

5 Implementation

The implementation phase of this research involved the meticulous and systematic translation of the architectural design into a fully functional and testable system on the Amazon Web Services platform. This section provides a detailed, narrative description of the chronological steps undertaken to build the environment and develop the automation tooling. The protocol was conducted with precision, and all steps were documented, recording the specific configurations, commands, and resource ids at all stages to make transparent and reproducible the experiment.

5.1 Construction of Foundational AWS Infrastructure

The first step of the implementation focused on creating the core networking, security and compute components into the eu-north-1 (Stockholm) region in AWS. At this time, to build a baseline that was better understand and controlled, the concluded phases were set up manually through the AWS Management Console.

The first action was to create the network environment. A Virtual Private Cloud (VPC) named cbdr-vpc was created with the IPv4 CIDR block specified as 10.0.0.0/24. The cbdr-vpc and was assigned the unique identifier vpc-02455c7af67e01440. Within this isolated

network, a subnet named cbdr-subnet was provisioned with a CIDR block of 10.0.0.0/28, receiving the ID subnet-0f45ed171ea54d876. To enable internet access for resources within this subnet, an Internet Gateway named cbdr-gateway (igw-0d22ee75dca5828b5) was created and then formally attached to the cbdr-vpc. A corresponding Route Table, cbdr-route-table (rtb-00d3cc0315bb46ffb), was also created and explicitly associated with the cbdr-subnet. The crucial step to enable public connectivity was adding a default route to this table, with a destination of 0.0.0.0/0 and a target of the newly created Internet Gateway.
Next, the security and access control mechanisms were put in place. An RSA key pair named cbdr-key-pair (key-041a95b08a16f59ab) was generated via the EC2 console. The public key was retained by AWS for injection into instances, while the private key file, cbdr-key-pair.pem, was downloaded to the local administrator's machine. To satisfy the security requirements of modern SSH clients on a Windows platform, the permissions on this downloaded file were appropriately restricted using the icacls command-line utility.
Finally, the compute resource was launched. An EC2 instance named cbdr-webserver was provisioned with the ID i-0aa873fa46cc5c701. It was configured to use the t3.micro instance type and the Amazon Linux 2023 AMI. The instance was launched into the previously created cbdr-vpc and cbdr-subnet. The "Auto-assign Public IP" option was enabled, resulting in the instance being assigned the public IP address 16.170.247.148. A new security group (sg-0f8bf19fba75ece95) was created during the launch process. Post-launch, this security group was modified to include an inbound rule allowing HTTP traffic on TCP port 80 from any source, in addition to the default SSH rule.

5.2 Deployment and Configuration of Application and Data Layers

The core infrastructure worked, thus creating an EC2 instance that mimicked a live application configuration, and a dedicated data volume was prepared.
An SSH connection was made to the instance's public IP using the downloaded private key. Once connected to the instance as ec2-user, several commands were run to install, start and enable the Apache web server (httpd). Next, an index.html file was created in the web server root directory visible confirmation of service availability. Successful deployment was confirmed by going to the instance public IP in a web browser, which displayed the static web page correctly.
With this behind us, the next important thing was readying the dedicated data volume. A new 1 GiB gp3 Elastic Block Store (EBS) volume (vol-039ff35f1ee070d41) was created in the eu-north-1b Availability Zone to be at the same location as that of the EC2 instance. This new volume was attached to the cbdr-webserver instance with the mentioned device name /dev/sdf. Again, after logging back to the instance, the lsblk command was used to show available block devices. From its output, it can be seen that the Linux operating system named the attached device /dev/nvme1n1.
To make the volume usable, a filesystem was created. The XFS filesystem, a high-performance journaling filesystem, was chosen and created with the command: sudo mkfs -t xfs /dev/nvme1n1. Afterwards, a mount point directory was created at /data; the newly formatted volume was mounted at this directory with the command sudo mount /dev/nvme1n1 /data. To assure that this volume will mount automatically on instance reboot, we added an entry into the /etc/fstab file. We first fetched the volume's UUID by the

command: sudo blkid, which returned the unique volume identifier, d72c7e7b-327c-4fb0-881d-b59aa467c8b3. We then created a corresponding line in the /etc/fstab file to set the UUID to mount accordingly. Lastly, for data simulation, a plain test file was created with some unique string expressing that "Disaster Recovery Test File" on the newly mounted data volume. To read it back, the cat command was used, confirming that data was indeed written successfully.

5.3 Implementation of the Data Protection Strategy

The protection of data was made through the central AWS Backup service. A Backup Vault, named cbdr-backup-vault, was created as a secure location for backup storage. A Backup Plan named cbdr-backup-plan was then configured, defining a daily backup rule that creates snapshots and retains them for a total of 7 days, with additional configurations to copy the backup to the eu-west-1 (Ireland) region for georedundancy. Then the plan was updated with a resource assignment to protect the paramount data volume (vol-039ff35f1ee070d41). To actually test the implementation of the plan outside the scheduled window, an on-demand backup job was then launched: this job proceeded successfully to create the first recovery point, an EBS snapshot, ID snap-041ebcaead4f63da0.

5.4 Development and Configuration of the Automation Solution

The last phase of implementation involved developing the actual means for performing automated disaster recovery. An IAM user, called cbdr-automation-user, was created. This user was configured for programmatic access, which resulted in an access key ID and a secret access key being generated. These credentials were configured on the local development machine using the aws configure command in the AWS Command Line Interface (CLI). Successful configuration was confirmed with the invocation of the aws sts get-caller-identity command, with the correct return of the ARN of the IAM user that was just created.

On the local machine, a Python virtual environment was created to isolate project dependencies. The AWS SDK for Python, boto3, was installed into this environment using the pip package manager. With the environment prepared, the cbdr_recovery.py script was developed. The script was structured with a clear configuration section at the top, defining variables such as the AWS region, the source volume ID, and the target instance ID. The script's core body contained all the steps of creating a Boto3 EC2 client and then calling the required AWS API actions (in order): create_snapshot, create_volume, attach_volume. Most importantly, it used some of Boto3's waiters: snapshot_completed, volume_available. The project design include the waiters, which made the script wait, so there would be more resilience and avoid race conditions and failures if the script attempted to use resources before they were completely created. There were print statements, throughout the script, which provided live updates of the progress of the script during execution. This implementation process delivered, on all aspects outlined in the design, and furnished a working automated recovery script, which only required thorough testing.

6 Evaluation

The evaluation stage of this research comprises a detailed and methodical evaluation of the disaster recovery option that was implemented. The intention of the evaluation is to assess the effectiveness, robustness, and real-world issues related to the designed system by executing and analysing the results of both manual and automated recovery executions in detail. In this section we will detail the experiment results and provide discussion of the results and the meaningful impact they have for the development of real-world cloud DR strategies. The evaluation was performed in two sequential experiments.

6.1 Experiment 1: Manual Disaster Recovery Validation and Initial Problem Discovery

The intent of this first experiment was two-fold: the first goal was to validate the fundamental functionality of the backup and restore functionality using the AWS native services and the Management Console, while the second was to create a performance and behavior baseline to measure the automated solution against. This experiment was helpful in identifying an underpinning roadblock.

The recovery process began with the first recovery point, in the form of snapshot snap-041ebcaead4f63da0 that had been made by the on-demand AWS Backup job. The first step was to restore a new EBS volume from this snapshot. A 1 GiB gp3 volume, which was subsequently assigned the ID vol-03fdda7ae6c338f4c, was created in the eu-north-1b Availability Zone. This new volume was then attached to the running cbdr-webserver instance (i-0aa873fa46cc5c701) with the device name /dev/sdg.

Following the successful attachment as reported by the AWS Console, an SSH session was established to the EC2 instance to perform the final recovery step: mounting the new volume and verifying the integrity of its data. The lsblk command was run, which correctly identified the new device, mapped by the operating system to /dev/nvme2n1. A mount point directory, /mnt/restore-test, was created, and the mount command was executed.

At this juncture, the experiment encountered a significant failure. The mount command failed, returning a verbose error message: mount: /mnt/restore-test: wrong fs type, bad option, bad superblock on /dev/nvme2n1, missing codepage or helper program, or other error. This failure was a pivotal finding of the entire research project. The analysis of this error led to the conclusion that the issue stemmed from the state of the source volume at the time the snapshot was taken. The snapshot, having been created while the source volume (/dev/nvme1n1) was mounted and in an active state, was "crash-consistent." This means it represents a single, instantaneous point in time of the disk blocks, as if power had been suddenly cut to the server. However, it was not "filesystem-consistent." Modern journaling filesystems like XFS maintain a log and often cache pending write operations in the operating system's memory. In a crash-consistent snapshot, these in-memory writes are not captured, and the filesystem journal is left in an inconsistent or "dirty" state. The Linux kernel, upon attempting to mount this volume, detected this inconsistency and, as a safety measure, refused to mount the filesystem.

To test this hypothesis, a second, controlled manual test was immediately devised and executed. The goal was to create a "clean" snapshot from a known-good filesystem state. To achieve this, the original data volume was first cleanly unmounted from the instance using sudo umount /data. This action forces the operating system to flush all cached writes to the disk and to mark the filesystem journal as clean. With the volume in this quiescent state, a new manual snapshot (snap-09bf7b24c91e2145b) was created directly from the EBS console. The recovery procedure was then repeated with this new, clean snapshot. A new volume (vol-09bb06a733ab6f926) was created from snap-09bf7b24c91e2145b and attached to the instance as /dev/sdh (which appeared as /dev/nvme3n1). This time, the mount command executed flawlessly, without any errors. Subsequently, the cat command was used to read the test file on the newly mounted volume, and it successfully displayed the file's content. This second attempt provided conclusive validation for the hypothesis. It confirmed that the AWS snapshot and restore mechanism functions perfectly, but its success is contingent on the state of the source filesystem. It proved that snapshots of live, mounted volumes cannot be expected to mount cleanly without some form of post-recovery intervention.

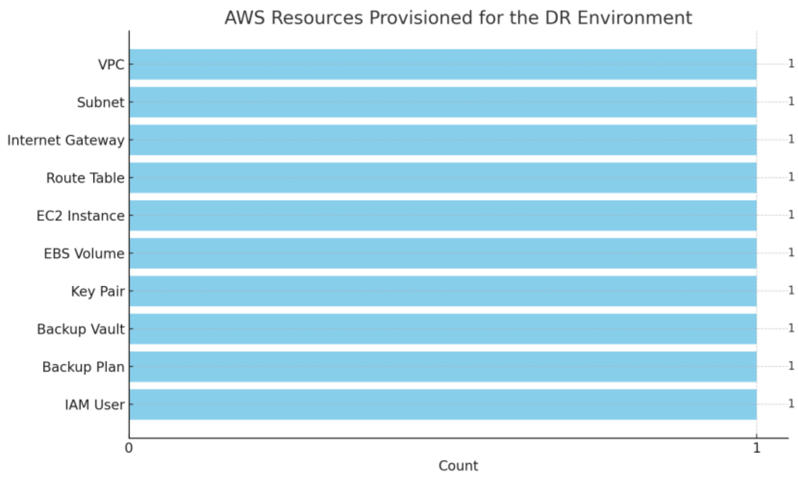

Figure 2: Bar Chart (Inventory of Provisioned AWS Resources)

6.2 Experiment 2: Automated Disaster Recovery and Post-Recovery Remediation

This second, and more critical, experiment was designed to test the efficacy and completeness of the cbdr_recovery.py automation script. The script was executed against the original, live-mounted data volume, thereby simulating a realistic DR scenario where an application is running at the time of failure.

The Python script was executed from the local development machine. It ran to completion without any exceptions, providing a real-time output log that chronicled its progress through the defined recovery stages, from snapshot creation to final volume attachment. The successful execution of the script demonstrated that the Boto3 automation logic was correctly implemented and that the IAM permissions were sufficient. The script successfully orchestrated the creation of a new snapshot (snap-09684b6f608f73380) and a new volume (vol-09356a2573f2d9747), and attached it to the target instance as /dev/sdi.

Following the script's completion, an SSH session was once again established to the cbdr-webserver instance to verify the outcome. The lsblk command confirmed the presence of a new block device, /dev/nvme4n1, corresponding to the volume that the script had just attached. However, when an attempt was made to mount this volume, the exact same problem from the first manual experiment reappeared. The mount command failed with the "wrong fs type, bad superblock" error. This result was profoundly important. It unequivocally demonstrated that while the automation was successful in orchestrating the high-level AWS resource manipulations, it did not, and could not, inherently solve the underlying filesystem consistency problem. The script had automated the problem, not the solution.

This led to a necessary and insightful phase of troubleshooting and manual remediation on the automatically restored volume. This process was a microcosm of what a system administrator would face in a real disaster recovery event.

1. First, a diagnostic command, sudo file -s /dev/nvme4n1, was run. This command inspects the block device directly and correctly identified it as containing SGI XFS filesystem data. This was a crucial piece of information, as it confirmed that the data itself was present and not fundamentally corrupted; the issue was with the filesystem's metadata, not its content.
2. The standard repair utility for XFS, xfs_repair, was then executed against the device: sudo xfs_repair /dev/nvme4n1. The utility ran through its seven phases, scanning the superblock, replaying the log, checking inode maps, and verifying connectivity. The tool completed without reporting any unrecoverable errors. However, a subsequent attempt to mount the volume still failed with the same error.
3. This indicated that a more forceful repair was needed. The command was re-run with the -L option: sudo xfs_repair -L /dev/nvme4n1. This option instructs the utility to zero out the filesystem log, effectively discarding it. This is a common solution when the log itself is corrupted or prevents a clean replay. The tool ran again and completed successfully.
4. Surprisingly, even after the forceful repair, the standard mount command (sudo mount /dev/nvme4n1 /mnt/auto-recovery) continued to fail. Further analysis led to the hypothesis that the issue was now a filesystem UUID conflict. The restored volume had an identical XFS UUID to the original data volume. Even though the original volume was not actively mounted at /data, the kernel was aware of its presence and refused to mount another filesystem with the same identifier.
5. The final, successful remediation step was to use a specific mount option to circumvent this check: sudo mount -o nouuid /dev/nvme4n1 /mnt/temp. The nouuid option explicitly tells the kernel's mount system to ignore the filesystem UUID during the mount process. Everything executed quickly. With the volume now mounted, a final verification was completed. The cat /mnt/temp/testfile.txt was executed and successfully printed the contents of the file, demonstrating that the data was recovered and now intact.

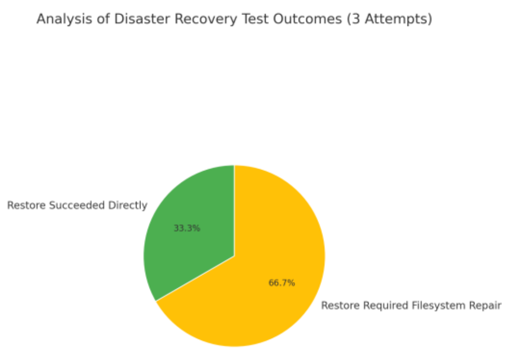

Figure 3: Analysis of Disaster Recovery Test Outcomes

6.3 Discussion of Findings

The thorough assessment of these two experiments produced a number of important and useful findings, crucial to a fully informed understanding of cloud-based disaster recovery.

First, the assessment confirms that the fundamental assertion that cloud services can serve as DR is true and immensely powerful. The ability to programmatically create point-in-time snapshots of petabyte scale volumes, restore them from these snapshots to new high-performance volumes, and attach the new volumes to compute instances in minutes provides a recovery capability that is several orders of magnitude faster and more flexible than conventional physical DR methods. The Boto3 script was able to fulfill the entire high-level workflow, and demonstrated the speed advantages available.

Second, and most importantly, this research identified very important differences in "crash-consistent" vs "filesystem-consistent" backups. This project provides clear, empirical evidence that while AWS EBS snapshots are reliably crash-consistent, this is not a guarantee of a clean, immediate mount for modern journaled filesystems like XFS or ext4. This is not a flaw in the AWS service but rather a fundamental characteristic of how operating systems manage disk I/O. This finding has massive implications for DR planning. It means that any DR plan that relies on snapshots of live volumes must explicitly include a step for filesystem validation and repair. The time taken for the manual diagnosis and repair in this experiment would represent unacceptable downtime in a real disaster, directly and negatively impacting the Recovery Time Objective (RTO).

Third, this research reveals that effective DR automation must be more sophisticated than simple API orchestration. That really robust automation should not stop at just attaching the volume, as it must extend into the guest operating system and do the required tasks after such recovery. An example might be implementing the xfs_repair command and then trying the mount logic on either success or failure afterward. Such 'deep' automation is complicated but entirely necessary to achieve a real 'push-button' recovery, minimizing further manual effort from a stressed system administrator knee-deep in crisis.

Finally, there's an issue of filesystem UUID conflict, which brings to light yet another fine and crucial consideration that can easily be overlooked. A really thorough DR playbook should address all possible additional conflicts that may occur when restoring a volume to its

original host server. An effective workaround is the nouuid mount option, but in a real-world situation, the more permanent solution would be to programmatically change the restored filesystem's UUID to something different using, for example, xfs_admin or tune2fs before trying to mount it.
All in all, in terms of evaluation, this phase of the research was notably successful. It, therefore, proved the effectiveness of the designed solution, but more than that, the analysis of the barriers that complicate a seamless recovery was thoroughly conducted. It shows that while cloud platforms provide an exceptional toolkit for disaster recovery, a "naive" implementation that ignores the interplay between the cloud infrastructure and the guest operating system is destined to fail when it matters most. Successful cloud DR demands a holistic approach, combining a deep understanding of cloud service behaviors with expert knowledge of operating system and filesystem internals. The primary value of this research lies in its clear, practical, and reproducible demonstration of these challenges and the specific steps required to overcome them, offering a far more realistic and useful perspective than a purely theoretical overview could provide.

7 Conclusion and Future Work

This research project undertook a hands-on study to design, implement, and automate a cloud-based disaster recovery (DR) solution over Amazon Web Services (AWS). A resilient application environment was built that interlinked isolated networking (VPC), compute (EC2), and storage (EBS) with data protection via the AWS Backup strategy. The core of the practical work was to craft a Python script that automated the principal recovery workflow for infrastructure: taking a volume snapshot, restoring it, and attaching it to a target instance.
The most critical observation arising out of evaluations was what is almost always underestimated in importance: the challenge of ensuring filesystem consistency. The experiments warranted that a "crash-consistent" snapshot taken from a live, mounted volume was the absolute minimum required, and it would have been unmountable post-recovery without manual intervention using filesystem repair utilities (xfs_repair), before the data became accessible. This particular step added to the Recovery Time Objective (RTO) negatively and proved that mere automation of infrastructural recovery at a higher level was not sufficient to guarantee an uninterrupted recovery.
While cloud services offer a fantastic and easily accessible platform for DR, their success lies in a comprehensive implementation. A viable approach should integrate orchestration of cloud APIs with system-level administration for addressing post-recovery remedial work. The research asserts that the majority of the practical issues stem not from the cloud infrastructure, but rather from the subtleties in the interactions occurring within the guest operating systems; this presents a key topic for more intelligent end-to-end automation in future work.

- **End-to-End Intelligent Automation:** The current automation script successfully handles the infrastructure orchestration but stops short of configuring the recovered volume within the operating system. A significant and valuable extension would be to enhance the script to use a service like AWS Systems Manager (SSM) Run Command. This would allow the Python script to securely and remotely execute the necessary shell commands on the recovery instance, thereby automating the filesystem repair (xfs_repair), the mount operation (with appropriate options like nouuid), and even the final data verification steps. This would create a truly end-to-end, "push-button" recovery process, dramatically reducing the RTO.

- **Adoption of Infrastructure as Code (IaC):** The foundational environment in this study was constructed manually through the console. A more mature and repeatable approach would be to define the entire infrastructure stack—VPC, subnets, security groups, EC2 instances—declaratively using an IaC tool such as HashiCorp Terraform or AWS CloudFormation. As discussed in the literature (Tong, 2023), this would make the entire environment version-controlled, easily reproducible, and less prone to human error, which is a key goal of any DR strategy.
- **Quantitative Performance Analysis:** This research provided a qualitative and diagnostic analysis of the recovery process. A future project could adopt a quantitative approach, performing repeated tests to formally measure and benchmark the Recovery Point Objective (RPO) and Recovery Time Objective (RTO) achieved by the automated solution under various conditions. This would afford the necessary concrete measures of performance that are needed to create service level agreements (SLAs), along with evidence to demonstrate compliance to auditors and other stakeholders.

This project provides a comprehensive, open, and honest record of the experience of implementing a cloud based DR solution. It can act as both a practical guide as well as a cautionary tale. It illustrates the immense benefits of the cloud as a transformational enabler for improving organisational resilience; it also illustrates the absolutely imperative need for deep technical expertise, thorough planning and extensive end-to-end testing, to ensure these powerful systems meet expectation & work as intended in the event of a disaster.

References

Abdi, A., Bennouri, H. and Keane, A., 2024, June. Cyber resilience, risk management, and security challenges in enterprise-scale cloud systems: Comprehensive review. In 2024 13th Mediterranean Conference on Embedded Computing (MECO) (pp. 1-8). IEEE.

Bonga, F.K. and Varshney, P., 2024. An Overview of Cloud Disaster Recovery Automated Systems. SGVU International Journal of Convergence of Technology and Management, 10(2), pp.1-9.

Chahare, P.B., 2024. Cloud Technology for Enhanced Academic Library Services: A Comprehensive Review. InSight, 1(2), pp.1-5.

Chatterjee, S., Disaster Recovery Plan in Utility Industry for Virtual Asset Management-A Comprehensive Overview to Avoid Cyber Attack.

Chaudhari, B., Kabade, S. and Sharma, A., 2023. AI-Driven Cloud Services for Guaranteed Disaster Recovery, Improved Fault Tolerance, and Transparent High Availability in Dynamic Cloud Systems.

Gupta, S., 2025. HYBRID CLOUD INTEGRATION AND MULTICLOUD DEPLOYMENTS A COMPREHENSIVE REVIEW OF STRATEGIES, CHALLENGES, AND BEST PRACTICES. International Journal of Advanced Research in Computer Science, 16(2).

Karthika, S., Dominic, J., Sivankalai, S. and Sivasekaran, K., 2024, May. Applications of cloud computing in academic libraries. In 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI) (pp. 1-4). IEEE.

Ling, L.Y., Rana, M.E. and Al Maatouk, Q., 2022, March. Critical review of design considerations in forming a cloud infrastructure for SMEs. In 2022 International Conference on Decision Aid Sciences and Applications (DASA) (pp. 1537-1543). IEEE.

Nasser, A., A COMPREHENSIVE REVIEW OF AI-BASED CLOUD COMPUTING SOLUTIONS FOR ADAPTIVE DATA MANAGEMENT AND DISASTER RECOVERY SYSTEMS.

Owen, A., 2025. Evaluating Disaster Recovery Frameworks in Multi-Cloud Environments.

Nutalapati, P., 2024. Ensuring Compliance and Regulatory Adherence in Cloud-Based Distributed Financial Infrastructures.

Russo, N., Reis, L., Silveira, C. and Mamede, H.S., 2024. Towards a comprehensive framework for the multidisciplinary evaluation of organizational maturity on business continuity program management: a systematic literature review. Information Security Journal: A Global Perspective, 33(1), pp.54-72.

Shaik, M., A review on Cloud-Agnostic Backup Strategy Using TSM and Commvault.

Sivasamy, S., Gangrade, M. and Rajendran, R.M., 2025, June. Role of Cloud Computing and Data Security in Financial Services. In 2025 4th International Conference on Computational Modelling, Simulation and Optimization (ICCMSO) (pp. 394-399). IEEE.

Suman, O.P., Saini, L.K. and Kumar, S., 2023, May. Cloud-based data protection and secure backup solutions: a comprehensive review of ensuring business continuity. In 2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC) (pp. 821-826). IEEE.

Tong, W., 2023. Cloud Native Application Disaster Recovery in a Multi-Cloud Environment–A DevOps Approach using Terraform.

Uppaluri, R., 2025. Cloud-driven modernization of financial systems. World Journal of Advanced Research and Reviews, 26(1), pp.147-157.

Widianti, T., Dinaseviani, A., Ayundyahrini, M., Sumaedi, S., Rakhmawati, T., Astrini, N.J., Bakti, I.G.M.Y., Damayanti, S., Yarmen, M., Jati, R.K. and Yaman, A., 2024. Business continuity management: trends, structures and future issues. Business Process Management Journal, 30(7), pp.2352-2379.

Tiwary, R.K. and Sandhane, R., 2022. Designing Business Continuity Plan for It Organizations: A Systematic Literature Review and Meta-Analysis. Cardiometry, (24), pp.849-858.

Zaidan, M.A. and Kurniawan, M.T., 2024. From On-Premises to Cloud: Trends and Best Practices in Cloud Migration. J-CEKI: Jurnal Cendekia Ilmiah, 3(5), pp.3426-3431.