International Journal of Cybersecurity Engineering and Innovation

ARTICLE INFO

Article History: Received: 29-03-2026, Revised: 07-06-2026, Accepted: 03-11-2026
Corresponding author Email: mho.phdscholar@lincoln.edu.my

This is an open access article under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

Published by ITAP Publisher.



Vol. 2027 No.1

SCOBDAF: Secure Cloud Operation for Big Data Availability Framework

Mony Ho[1*], Sopheaktra Huy[2], Sokroeurn Ang[3], Midhunchakkaravarthy Janarthanan[4]

[1, 2, 3, 4] School of AI Computing and Multimedia, Lincoln University College, Selangor, Malaysia

Abstract

Ensuring the continuity of big data operations during system failures remains a significant challenge, as Hadoop, Sqoop, and Python-based workflows may be interrupted when dependent services become unavailable. Although existing cloud recovery mechanisms can restore failed instances, they do not necessarily guarantee workflow continuity, leaving data processing pipelines vulnerable to disruptions. To address this challenge, this study proposes SCOBDAF, a layered cloud security and continuity framework that integrates routing continuity, computational fault tolerance, and data persistence. These components are organized through the Continuity Assurance Formula, which provides a systematic and measurable approach to evaluating workflow continuity. The proposed framework was implemented on Amazon Web Services (AWS) and experimentally evaluated through load balancer failover, auto-scaling recovery, and persistent storage synchronization tests. Experimental results demonstrated that all continuity variables returned Boolean values of True, resulting in an overall True evaluation of the Continuity Assurance Formula. Furthermore, no workflow interruptions or data inconsistencies were observed during the conducted experiments. The findings demonstrate the effectiveness of the proposed framework in maintaining operational continuity and data consistency under the evaluated failure scenarios. This study contributes a structured continuity assurance model for cloud-based big data environments and presents a practical approach to supporting uninterrupted Hadoop, Sqoop, and Python-based workflows during node and service failures.

Keywords: Cybersecurity, Big Data continuity, cloud availability, Hadoop resilience, workflow recovery, data persistence, fault tolerance, and the SCOBDAF framework.

1. Introduction

A real-time, data-integrated, data-driven system enables continuous decision-making operations [1]. Big Data systems depend on a continuous, iterative quality-monitoring and enforcement process to maintain data reliability across the entire lifecycle [2]. A study demonstrates that cloud platforms provide a scalable environment for Big Data by enabling Hadoop clusters to automatically expand or shrink based on real-time workloads. [3]. Ensuring system availability under uncertain disturbances requires resilient strategies that solve both physical and cyber disruptions to maintain acceptable operational performance [4]. Big Data applications continuously generate data that must be processed and analyzed in real time to support uninterrupted operational workflows [5].

The article highlights that Big Data systems lack automatic recovery, as they suffer from downtime and low availability and require manual confirmation from staff before initiating disaster recovery processes [6]. NameNode is the single point of availability failure. If a NameNode goes down, the DataNodes would not be able to make any sense of the data blocks on them [7]. Sqoop cannot operate independently because it relies on Hadoop daemons and MapReduce, and similarly, Python jobs executed through Hadoop Streaming or Pydoop depend on the same Hadoop services, meaning that when Hadoop fails, both Sqoop and Python jobs fail [8], [9]. Cloud platforms are susceptible to service interruptions caused by instance dormancy or failure, during which no requests can be processed and normal service resumes only when the node returns to the active state [10]. Despite extensive research on Big Data disaster recovery, Hadoop availability limitations, Sqoop–Hadoop dependency, and cloud instance interruptions, existing studies do not integrate cloud fault-tolerance mechanisms with the real-time continuity of Big Data workflows. As a result, failures at either the Hadoop layer or the cloud infrastructure layer still cause immediate job termination and system downtime.
Given the limitations of existing Big Data systems in achieving automatic recovery and operational continuity across distributed environments, this study seeks to address the following research question: How can a cloud-layered architecture ensure continuous Big Data operations during system failures?

The novelty of this study is reflected in the following distinct scientific contributions:

Proposes SCOBDAF (Secure Cloud Operation for Big Data Availability Framework), a new cloud-layered security and availability model specifically designed to maintain continuous Big Data operations during system failures.
Introduces the first integrated framework that links cloud-layered architectural resilience (access, compute, and storage) with end-to-end Big Data service continuity for Hadoop, Sqoop, and Python workloads.
Establishes a cybersecurity-driven perspective in Big Data system design by jointly addressing availability and integrity as primary operational goals.
Demonstrates the feasibility of the model through a real cloud-based Big Data implementation, moving beyond theoretical discussion to practical operational validation. This study enriches the current body of literature by offering the following contributions:
Proposes a practical cloud-layered security and fault-tolerance model (SCOBDAF) that enhances availability and operational resilience in Big Data environments.
Demonstrates how Hadoop, Sqoop, and Python workflows can maintain continuous operation during system or node failures through an integrated architectural approach.
Provides a structured mapping of the CIA Triad: confidentiality, integrity, and availability, to Big Data workloads operating in cloud environments.
Addresses a critical literature gap by linking cloud resilience mechanisms with Big Data system availability, an area previously examined in isolation.
Offers a replicable cloud architecture that supports fault-tolerant Big Data processing and can be adapted across multiple cloud service providers.

The SCOBDAF adopts a three-layer architectural model designed to ensure continuous data ingestion, storage, and processing even in the presence of node or instance failures. First is the Access and Routing Layer manages traffic routing and access control, providing secure and reliable entry into the system. Second is the Compute and Fault-Tolerance Layer delivers auto-scaling compute capacity and virtual machine resilience to maintain uninterrupted Hadoop, Sqoop, and Python operations. And third is the Data Persistence Layer ensures durability through persistent volumes and distributed storage, enabling fault-tolerant management and retrieval of Big Data throughout the processing lifecycle. To design and evaluate a cloud-layered security and availability framework that ensures continuous Big Data operations during system or node failures. Accordingly, the study pursues the following objectives:
- Identify the key availability challenges affecting Hadoop, Sqoop, and Python workflows in cloud-based Big Data environments.
- Develop the SCOBDAF model using a multi-layer cloud architecture that strengthens access control, compute resilience, and data persistence.
Ensure uninterrupted Big Data processing by enabling continuous Hadoop, Sqoop, and Python operations during node, service, or infrastructure failures.
- Map the proposed framework to the CIA Triad, emphasizing availability and integrity in cloud-based Big Data systems.
- Evaluate the effectiveness of cloud-layered fault-tolerance mechanisms in enhancing the security and resilience of Big Data workloads.

1.1 Challenges in Maintaining Continuous Big Data Operations

Prior studies show that equipment failures in distributed environments can cause significant service downtime and disrupt operational continuity, emphasizing the need for proactive failure management in large-scale systems [11]. The Hadoop NameNode remains a single point of failure; once it becomes unavailable, the entire cluster becomes inaccessible and requires manual recovery procedures. The Sensors and Journal of Big Data articles indicate that Sqoop depends on Hadoop daemons because its import and export tasks run as Hadoop MapReduce jobs, and that Python is used inside Hadoop's processing engines; therefore, if the Node handles Hadoop fails, all dependent services stop functioning [13],[14]. Cloud environments lack automated recovery, requiring manual intervention that causes service delays and operational risk, alongside HDFS inconsistencies that arise from metadata and block-state failures in traditional clusters [15], [16]. On-premise Big Data infrastructures face complexity, high operational cost, manual management requirements, and scalability limitations [17].

1.2 Cloud-Based Approaches for Big Data Resilience

Elastic scaling allows a system to automatically expand or reduce compute capacity in response to changing Big Data workloads [18]. Routing mechanisms that improve availability by redirecting traffic away from failed nodes [19]. Auto-healing mechanisms detect failed compute instances and automatically replace them with new ones to minimize service disruption. Modern cloud environments increasingly incorporate these self-healing capabilities to ensure continuous availability [20]. Persistent storage ensures that stateful workloads retain their data even when compute components fail. Studies show that state variables must be stored in durable storage structures that remain accessible after failures, enabling applications to recover correctly and continue execution without data loss [21]. Replication distributes copies of data across independent sites or failure domains to enhance availability and fault tolerance, ensuring that services remain operational even when one location or zone becomes unreachable. This redundancy makes replication a key mechanism for achieving high availability in cloud-based Big Data environments [22], [23].

1.3 Cybersecurity and Availability in Cloud-Based Big Data Systems

Availability is a core element of the CIA Triad, and Big Data systems relying on continuous ingestion and real-time processing become insecure when availability is disrupted, because protections fail once underlying layers are inaccessible [24]. Fault-tolerant techniques such as replication and resilient key-distribution are recognized as cybersecurity mechanisms because they ensure secure services remain available even when some servers fail or are under attack [25]. Service interruptions create security vulnerabilities, and DoS/DDoS attacks purposely exploit downtime by overwhelming critical services, preventing legitimate requests and halting dependent operational workflows [26]. Identity and access control mechanisms, insecure interfaces, weak APIs, and misconfigured authorization boundaries are recognized cloud security threats that can directly disrupt service availability by allowing unauthorized access or blocking legitimate users [27].

1.4 Cloud Service Providers and Their Role in Big Data Continuity

Cloud service providers support Big Data availability through infrastructure services such as AWS ALB and ASG, Azure VM Scale Sets, and Google Cloud Instance Groups. These mechanisms use load balancing, redundancy, and virtualization-based failover to maintain uptime, and multi zone or multi region deployments further improve resilience by distributing resources across independent failure domains. Despite these strengths, infrastructure level availability does not guarantee workflow continuity for tightly coupled Big Data pipelines. Cloud high availability mechanisms can replace failed virtual machines but cannot preserve application state or restore in progress execution. Workflows dependent on Hadoop, Sqoop, and Python therefore remain vulnerable when any core node fails. Similar limitations are noted in medical Big Data environments, where cloud based telecardiology systems still face challenges with latency, confidentiality, interoperability, and reliable access [28]. Studies on distributed cloud computing further show that workflow stability is strongly affected by inter node latency, region placement, and cross zone orchestration constraints. Research on multi region deployments demonstrates that even with resilient cloud infrastructure, distributed applications experience performance variability due to asynchronous communication, inconsistent network conditions, and coordination delays. These findings align with Big Data workload research emphasizing that tasks spanning multiple nodes, services, or regions require application-level continuity mechanisms beyond what cloud native tools provide [29], [30]. Therefore, while cloud providers deliver strong foundations for service availability, they do not inherently maintain uninterrupted execution of complex Big Data workflows. This gap highlights the need for layered continuity frameworks such as SCOBDAF that integrate routing

continuity, compute recovery, and data persistence into a unified design capable of preserving operations during node and service failures.

1.5 The Need for a Unified Continuity Operation Solution for Big Data Systems

The review of existing literature reveals several critical gaps that hinder the development of uninterrupted and secure Big Data operations in cloud environments.

Gap 1: Existing research does not provide a unified cloud layered security and availability model capable of maintaining continuous Hadoop, Sqoop, and Python operations during failures.

Gap 2: Availability remains underexplored as a cybersecurity requirement in Big Data environments, even though service disruptions directly affect security, integrity, and operational continuity.

Gap 3: Current cloud architectures do not offer minimal service designs specifically intended to sustain uninterrupted Big Data workflows across distributed systems.

Gap 4: The literature lacks a formal formula driven or model driven framework for designing and evaluating cloud based Big Data continuity mechanisms.

Gap 5: Existing studies emphasize infrastructure resilience rather than workflow resilience, leaving Hadoop, Sqoop, and Python processes vulnerable when a single component fails.

These gaps demonstrate that despite advancements in cloud infrastructure resilience, there is still no comprehensive approach to maintaining uninterrupted Big Data workflows. Existing work does not provide a unified cloud layered model, does not treat availability as a core cybersecurity requirement, and does not offer minimal service cloud architectures that support end to end workflow continuity. Furthermore, the absence of formal formula driven frameworks limits the design and evaluation of continuity strategies. Most studies address compute, storage, or network redundancy but do not extend resilience to operational workflows, where the failure of a single Hadoop component can halt dependent Sqoop and Python processes. These unresolved limitations collectively justify the need for a specialized framework such as SCOBDAF.

2. Methodology

2.1 Framework Design Methodology

The framework design followed three steps, as illustrated in Figure 1. Step 1 identified four key availability requirements from the literature: workflow-level continuity, automated failover, persistent data durability, and secure routing and access control. Step 2 mapped these requirements onto the three-layer SCOBDAF architecture, consisting of the Access and Routing Layer, the Compute and Fault Tolerance Layer, and the Data Persistence Layer. Step 3 defined the interactions among these layers, where the routing layer redirects traffic to healthy nodes, the compute layer automatically replaces failed instances, and the persistence layer maintains synchronized and durable storage, ensuring uninterrupted Hadoop, Sqoop, and Python operations.

To formalize the operational behavior of the SCOBDAF framework, this study introduces the Continuity Assurance Formula (CAF) is adapted from the quantitative reasoning framework [31]. The formula mathematically is expressed as:

$$CAF = R_c \times C_f \times D_p \quad (1)$$

Where:

R_c (Routing Continuity): Represents the ability of the access and routing layer to maintain uninterrupted request flow to healthy compute nodes through resilient routing policies and failover mechanisms.

C_f(Compute Fault-Tolerance): Captures the reliability of the compute layer, including automated instance recovery, elastic scaling, and consistency of the execution environment across replacement nodes.

D_p (Data Persistence): Represents durable and consistent data availability achieved through replication, synchronization mechanisms, and distributed storage redundancy.

To operationalize Equation (1), CAF is evaluated using a Boolean dependency rule:

- CAF = Success if and only if$(R_c = \text{True}) \wedge (C_f = \text{True}) \wedge (D_p = \text{True})$
- CAF = Fail otherwise.

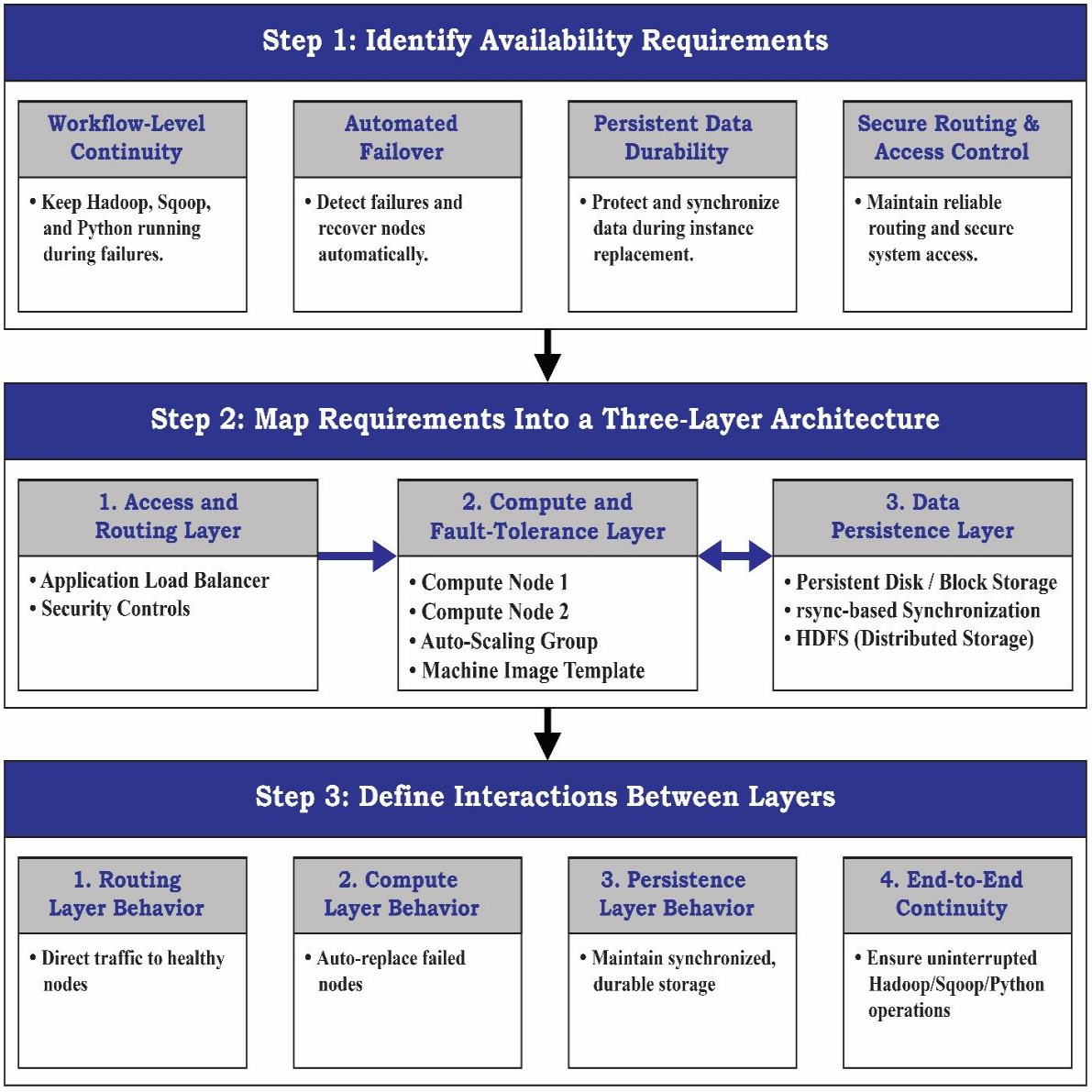

Figure 1. SCOBDAF Framework Design Diagram.

This logical interpretation reflects the tri-factor dependency structure of the original Three-Tenet model, where an outcome is achieved only when all required conditions co-occur. Accordingly, the SCOBDAF framework preserves workflow continuity only when routing continuity, compute fault-tolerance, and data persistence remain fully operational.

2.2 Implementation Environment and System Deployment

The SCOBDAF framework was implemented and deployed consisted of five core components, each corresponding to the framework layers including:

1. EC2 Instance Preparation: A baseline EC2 instance (Server01) running Ubuntu was provisioned to host Hadoop, Sqoop, Python, and Django. Security groups were configured to allow only essential ports, including HTTP, SSH, Django (8000), and the required Hadoop service ports. Big Data Processing Stack Installation:
Installation of Java and verification of Hadoop daemons using *jps*.
Configuration and validation of Sqoop integration with Hadoop.
Installation of Python, Django, and supporting libraries necessary for Big Data workflow execution.
2. Persistent Storage Configuration: An Amazon EBS volume was created, formatted, and permanently mounted to store Django project files and datasets. Volume persistence across instance restarts and replacements was confirmed using *df -hT*.
3. AMI Creation for Compute Replication: A custom Amazon Machine Image (AMI) was generated from Server01. The AMI captured all installed components, including Hadoop, Sqoop, Python, Django, and the associated storage configuration. This AMI served as the template for provisioning Server02 and additional nodes within the Auto Scaling Group.
4. EBS Synchronization between Nodes: To maintain data consistency across compute nodes, *rsync* was configured through *cron* to synchronize Django directories from Server01 to Server02 every minute. Directory comparison tests verified successful synchronization and data consistency.
5. Load Balancer and Auto Scaling Group Configuration
An Application Load Balancer (ALB) was configured to distribute HTTP traffic between Server01 and Server02 and to automatically reroute requests when a node becomes unhealthy. An Auto Scaling Group (ASG) was configured to automatically replace failed instances using the prepared AMI and maintain the required number of active compute nodes. Together, these components provide automated fault recovery, consistent compute availability, and uninterrupted Big Data workflow execution even during node failures.

2.3 Operational Workflow of SCOBDAF

This workflow documents how the system functions under normal and failure conditions.

2.3.1 Normal Operation Workflow

1. User traffic enters through the ALB (Access Layer).
2. Requests are routed to a healthy compute node.
3. Hadoop, Sqoop, and Python execute workflows on the active node.
4. Persistent storage ensures data is continuously available across nodes.
5. EBS synchronization keeps project data updated across replicas.

2.3.2 Failure Operation Workflow

When Server01 is manually or accidentally stopped, as demonstrated in the results section:

1. ALB immediately redirects traffic to Server02.
2. ASG detects the unhealthy instance and launches a replacement.
3. The replacement instance inherits the AMI and reattaches persistent storage.
4. Synchronization resumes automatically upon activation of the new node.
5. Hadoop, Sqoop, and Python workflows continue without requiring manual recovery.

2.4 Evaluation Method: Failure Simulation and Continuity Testing

To evaluate SCOBDAF, controlled node-failure experiments were conducted. These tests measured workflow continuity and the behavior of cloud recovery mechanisms during failure events. Table 1 presents the failure simulation and continuity testing scenarios

Table 1. Failure Simulation and Continuity Testing Scenarios

Test Scenario	1: Load Balancer Continuity Test (R_c)	2: Auto Scaling Recovery Test (C_f)	3: Persistence and Synchronization Test (D_p)
Procedure	• Stop Server01 • Access the web service through ALB DNS	• Modify ASG capacity or terminate a running instance	• Create, modify, or delete files on Server01 • Validate whether the same changes appear on Server02
Expected Outcome	• ALB continues serving the application using Server02	• ASG creates a replacement instance with the same configuration	• rsync replicates all changes from Server01 to Server02, ensuring consistent data across both nodes

2.5 Evaluation Metrics

The evaluation metrics used to assess system behavior during failure and recovery conditions are aligned with the Continuity Assurance Formula (CAF), where continuity is achieved only when Routing Continuity (R_c), Compute Fault-Tolerance (C_f), and Data Persistence (D_p) are all satisfied. The success and failure indicators corresponding to these three CAF components are summarized in Table 2.

Table 2. Evaluation Metrics Mapped to CAF Components

Metric Type	Indicators	CAF Components
Success Indicators	• Continuous service accessibility through ALB	R_c
	• Automated instance recovery via ASG	C_f
	• No interruption to Hadoop, Sqoop and Python operations	C_f
	• No loss of stored data during failover	D_p
Failure Indicators	• Workflow termination	C_f
	• Inconsistent dataset replication	D_p
	• ASG failover malfunction	C_f

Failure indicators do not include R_c because no routing-layer malfunction occurred. The ALB performed correctly in all tests, and the experiment did not simulate an ALB failure; therefore, no R_c failure case was observed or recorded.

3. Results and Discussion

3.1 Implementation of the SCOBDAF Framework

The SCOBDAF framework was deployed on AWS to validate its ability to maintain Big Data workflow continuity during node and service failures. This implementation directly responds to Section 2.2 by applying the defined deployment configuration in a real cloud environment. As shown in Figure 2, the architectural design of the deployed system integrates multiple AWS components, including load balancing, auto scaling, persistent storage, and Big Data processing tools to support continuous data ingestion, processing, and web-based application operations.

3.1.1 Create EC2 Instance

Before implementing the distributed cloud-based system, an AWS account is created and accessed through the AWS Management Console. The first EC2 instance (Server01) is launched using an Ubuntu AMI, a new key pair (Server01.ppk), and the t2.medium instance type to support Hadoop processing requirements. A security group is configured to allow HTTP

(80), SSH (22), Hadoop (54310), and Django (8000). An additional 10 GB EBS volume is attached to store Django project files and datasets. These settings establish the baseline environment for deploying Hadoop, Sqoop, Python, and Django under the SCOBDAF framework.

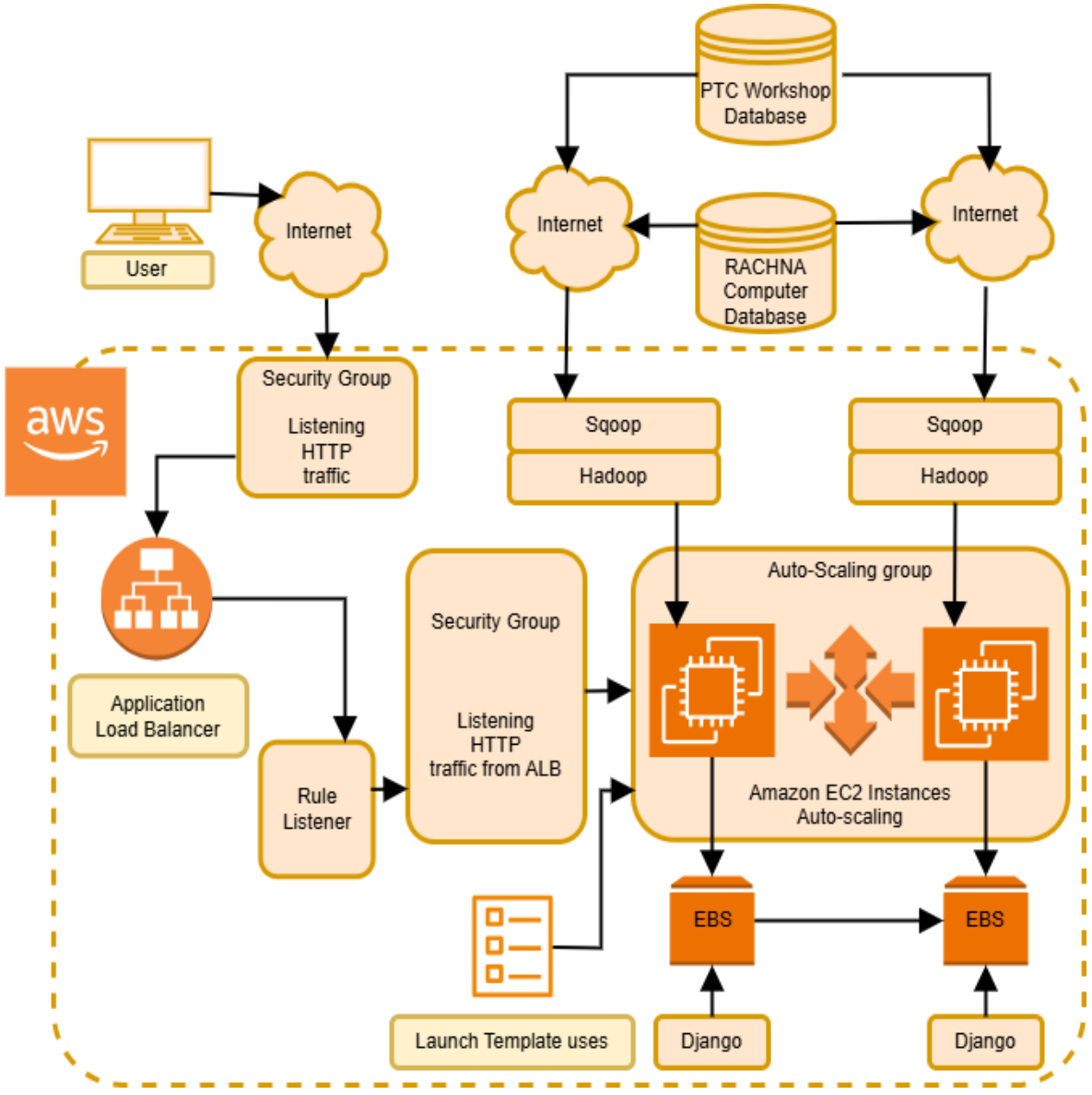

Figure 2. Architectural Diagram for the System Deployed on AWS

3.1.2 Remote to AWS EC2 Instance

After creating the EC2 instance, the next step is to connect from the local computer to the cloud server for administration. The connection is made using the instance's public DNS, the default Ubuntu username, and the private key downloaded during instance creation. AWS provides SSH connection instructions directly in the console, and tools such as PuTTY can be used to establish the remote session from Windows. Once connected, the EC2 instance can be fully managed through the terminal.

3.1.3 Java and Apache Hadoop Installation

This subsection outlines the installation, configuration, and verification of Apache Hadoop on an AWS EC2 instance as part of the SCOBDAF deployment. The setup includes installing Java, configuring system paths, enabling SSH communication, and updating Hadoop's core configuration files. Once configured, Hadoop services were started and validated using diagnostic commands to ensure that all essential daemons were running correctly. This establishes the foundational Big Data processing environment required for later integration with Sqoop and Python workflows.

3.1.4. Sqoop Installation

This subsection outlines the configuration and verification of Apache Sqoop on the AWS EC2 instance. Environment variables were added to .bashrc, and the sqoop-env.sh file was updated to link Sqoop with the Hadoop installation. After configuring the necessary paths, the installation was confirmed using the Sqoop version command which verifies that Sqoop was correctly integrated into the SCOBDAF framework.

3.1.5 EBS Mounting

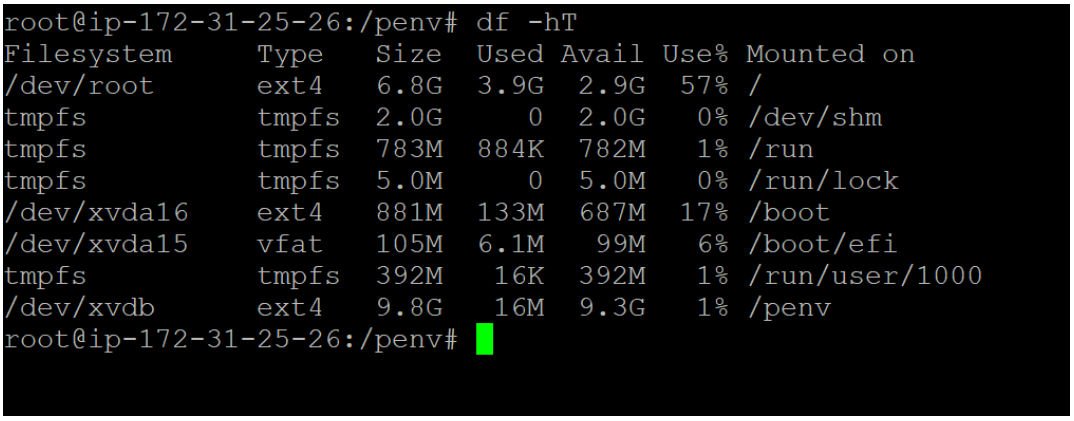

Figure 3. Checking mounted partition information

An additional EBS volume was attached to the EC2 instance to provide dedicated storage for the Django project and Big Data processing tasks. After attaching the new volume, it was formatted with the ext4 filesystem and configured in /etc/fstab for persistent mounting. Once the configuration was applied, the system recognized the volume and mounted it to the project directory. Final verification was performed using the df -hT command, as illustrated in Figure 3, confirming that the EBS volume was successfully formatted, mounted, and accessible to the application environment.

3.1.6 Python and Django Installation

After testing the web server, an Amazon Machine Image (AMI) was created from the first EC2 instance to capture its configured operating system, installed applications, Hadoop/Sqoop environment, Django project, and attached EBS volume. This AMI, named Ubuntu01, serves as a reusable template for launching additional instances with identical configurations, enabling consistent, fault-tolerant deployment within the SCOBDAF framework.

3.1.7. AMI Creating

After validating the web server on the first EC2 instance, an Amazon Machine Image (AMI) was created to capture the full system state, including the operating system, Hadoop, Sqoop, Django, and the mounted EBS volume. This AMI serves as a reusable template for deploying identical environments. Using this saved AMI, a second EC2 instance (Server02) was launched, inheriting all configurations from Server01. Since Django requires each server to specify its own public DNS, the ALLOWED_HOSTS value in the Django settings.py file was updated to match the new instance's hostname, as shown in Figure 4, to ensure proper application access.

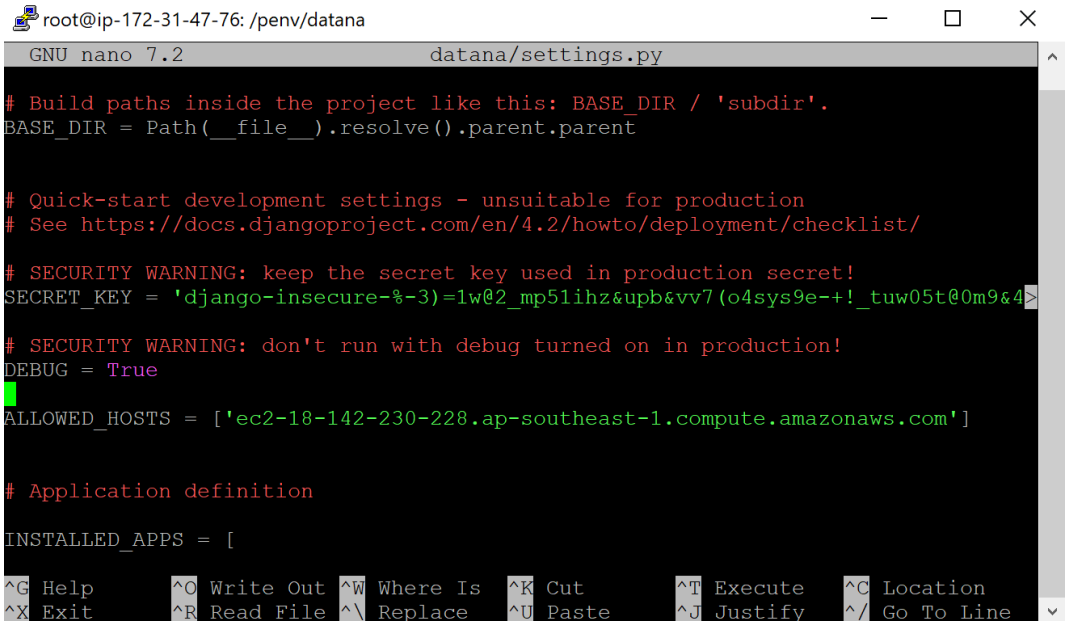

Figure 4. Change Allowed Hosts in setting.py of Django project

3.1.8 Application Load Balancing

Application Load Balancing (ALB) was configured to ensure continuous service availability and automatic redirection during instance failures. The configuration included registering both EC2 instances (Server01 and Server02) as target nodes, enabling multi-zone coverage, and applying appropriate security group rules for HTTP (80), SSH (22), custom TCP (8000), and the Hadoop service port (54310). After completing the setup, the ALB became active, as shown in Figure 5, confirming that it is ready to distribute incoming traffic and transparently redirect user requests to the healthy instance whenever a failure occurs.

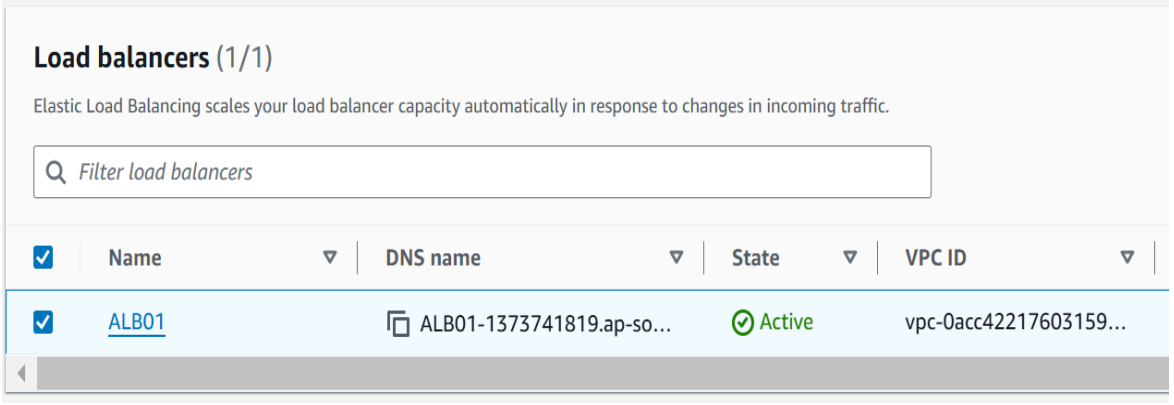

Figure 5. ABL List

3.1.9 Auto Scaling Group

Auto Scaling Group (ASG) enables the system to recover from failures and continue operating without service interruption. ASG is implemented in two steps: creating a launch template and configuring the ASG. The launch template defines instance properties such as AMI, instance type, security groups, and attached EBS volumes. Afterward, the ASG is created

by specifying capacity settings, availability zones, health checks, and load balancer integration. Once deployed, the ASG automatically adds or replaces instances to maintain system availability and ensure uninterrupted Big Data workflow operations, as illustrated in Figure 6.

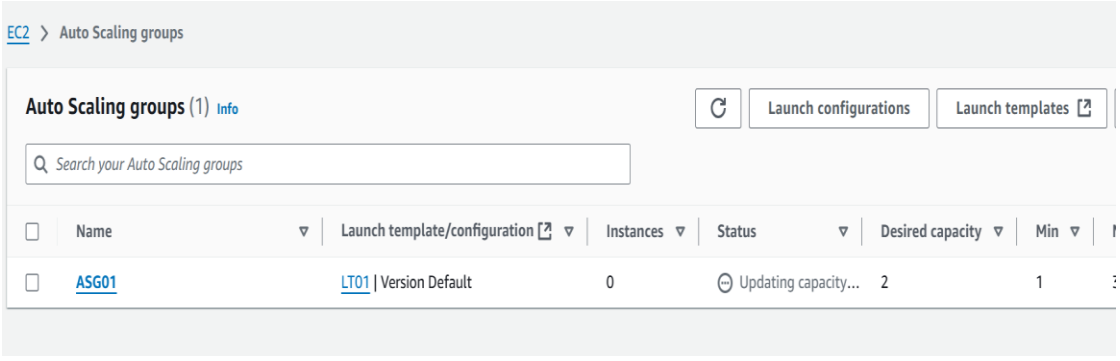

Figure 6. Listing ASG

3.2 Operational Workflow Validation

This validation demonstrates that the SCOBDAF framework functions as designed and provides evidence of its operational reliability. The results confirm the operational workflow described in Section 2.3. During normal operation, the system behaved as expected: the ALB consistently directed traffic to healthy compute nodes, Hadoop, Sqoop, and Python workflows executed without interruption, and persistent storage remained accessible across all nodes. Under failure conditions, the interaction between the layers performed exactly as defined in the workflow model. When Server01 was intentionally stopped, the ALB redirected traffic to Server02, the ASG initiated automatic replacement of the failed instance, and data synchronization continued through EBS and rsync. These observations validate that the SCOBDAF framework operates according to its designed workflow in both steady-state and failure scenarios.

3.3 Evaluation Method Implementation

This validation step demonstrates that the evaluation procedures were correctly executed and provides direct evidence that the SCOBDAF framework can be tested in a reliable and repeatable way. The experiments conducted follow the evaluation procedures outlined in Section 2.4. Each test scenario, including the ALB continuity test, the ASG recovery test, and the persistence synchronization test, was executed exactly as defined in the methodology. Server shutdown actions, ASG capacity adjustments, and file level replication processes were performed according to the specified steps, ensuring a controlled and repeatable testing environment. These procedures enabled accurate simulation of node failures and effective assessment of the system response to disruptions. The consistent success of all test scenarios confirms that the evaluation method was correctly implemented and supports the strength and reliability of the SCOBDAF framework.

3.4 Failure Continuity Testing

This section applies the Continuity Assurance Formula (CAF = $R_c \times C_f \times D_p$) to evaluate the SCOBDAF framework. The Rc is validated through the ALB failover test, C_f is validated through the ASG recovery test, and D_p is validated through the EBS and rsync synchronization test. Full continuity is achieved only when all three components remain functional.

3.4.1 Testing Routing Continuity (R_c) on ALB Failover Test

This section demonstrates how the ALB is tested to ensure high availability and continuous service delivery. As shown in Figure 7, the failover test begins by stopping the first EC2 instance to simulate a node failure. The website is then accessed using the public DNS address of the ALB, confirming that traffic is correctly routed to the remaining healthy instance. Even when one instance becomes unavailable, the ALB continues to forward requests to the active node, ensuring uninterrupted service for users. The second instance and the load balancer maintain normal operation without disruption,

as further illustrated in Figure 8, where the web application remains accessible through the load balancer's public DNS address. This successful failover confirms that:

$$R_c = True$$

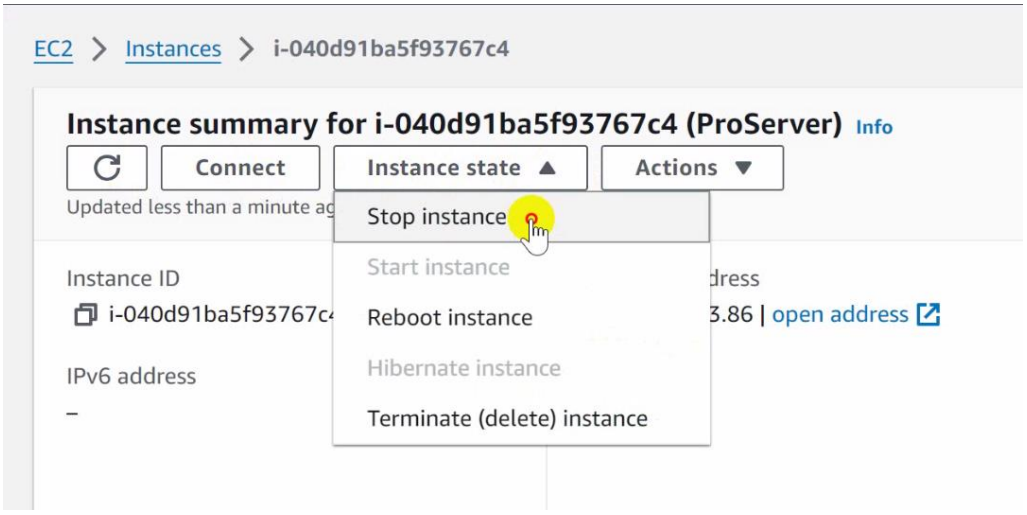

Figure 7. Stopping first instance to test Load Balancing

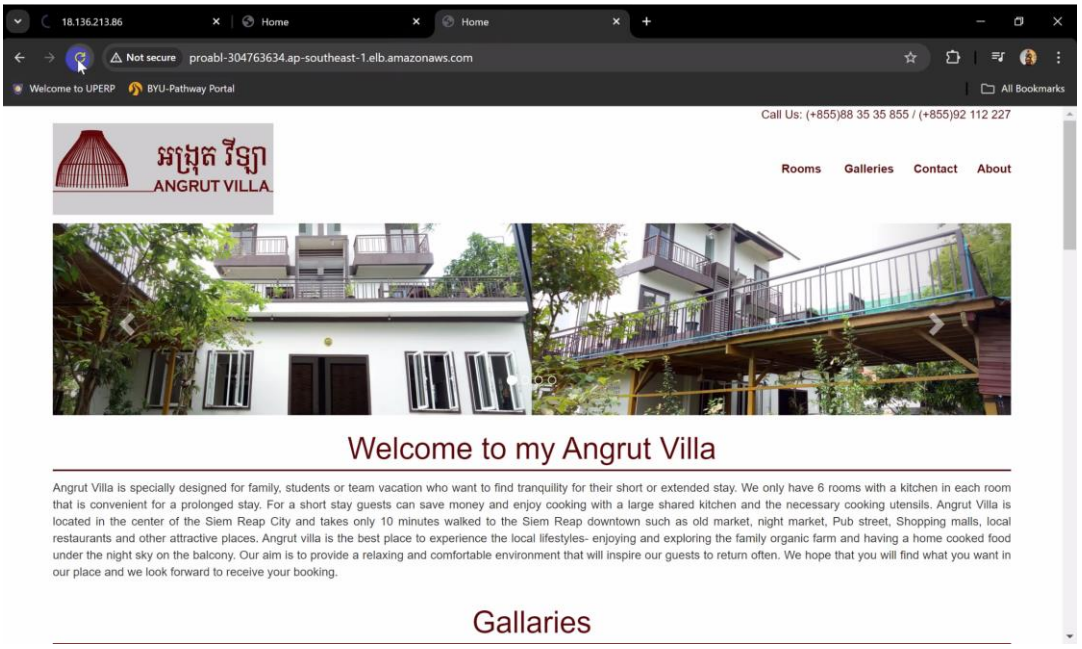

Figure 8. Website still responds by using Load Balancing pubic DNS address

3.4.2 Testing Compute Fault Tolerance (C_f) on ASG Auto-Recovery Test

This section verifies that the Auto Scaling Group (ASG) can dynamically adjust the number of running EC2 instances based on the configured scaling parameters. By editing the ASG settings, as illustrated in Figure 9, the desired, minimum,

and maximum capacities can be modified to trigger scale out or scale in behavior. This successful auto recovery confirms that:

$$C_f = True$$

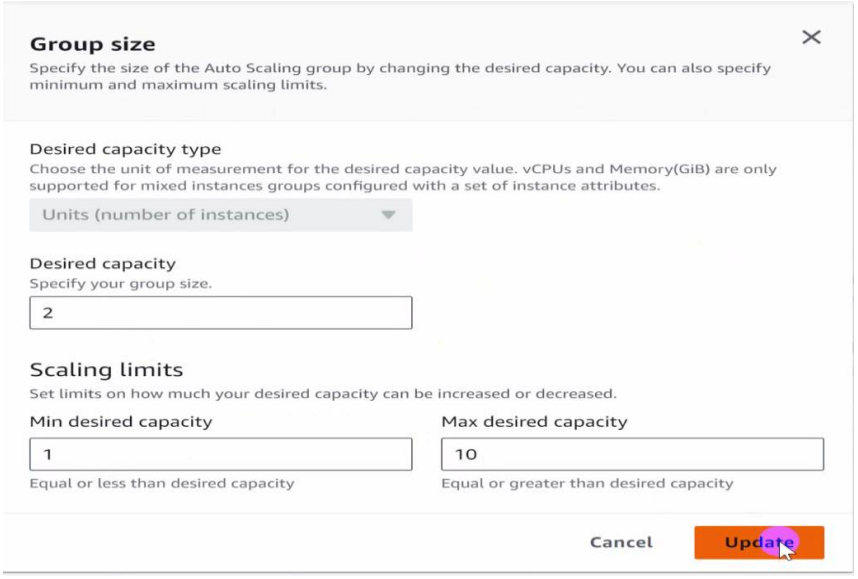

Figure 9. Updating Auto Scaling number of instances

3.4.3 Testing Data Persistence (D_p) on EBS rsync Synchronization Test

This subsection demonstrates the EBS synchronization process between Server01 and Server02 using the rsync command scheduled through crontab. After configuring SSH key based authentication and the necessary permissions, an automated sync job was executed every minute to replicate the Django project directory from Server01 to Server02. The comparison shown in Figure 10 confirms that both servers maintain identical directory structures and files, validating that the synchronization mechanism works reliably within the SCOBDAF framework. This successful replication confirms that:

$$D_p = True$$

3.5 Evaluation Metrics Assessment

The evaluation metrics defined in Section 2.5 were used to assess the continuity and resilience of the SCOBDAF framework. Each CAF component was validated through its corresponding test: routing continuity (R_c) through the ALB failover test, compute fault tolerance (C_f) through the ASG auto recovery test, and data persistence (D_p) through the EBS and rsync synchronization test. All three components returned True, indicating that the routing layer successfully redirected traffic, the compute layer automatically replaced failed instances, and the persistence layer maintained consistent data across the nodes. All success indicators were achieved. The ALB provided continuous service access, the ASG deployed replacement instances as expected, Hadoop, Sqoop, and Python workflows continued without interruption, and all data remained consistent during the failure events. None of the failure indicators, including workflow termination, inconsistent data replication, or malfunction of the ASG, were observed during the experiments. Based on these results, the Continuity Assurance Formula is evaluated as follows:

$$CAF = R_c \times C_f \times D_p = True \times True \times True = \boldsymbol{True}$$

This confirms that the SCOBDAF framework fully satisfies the evaluation metrics and achieves complete workflow continuity under the tested failure conditions.

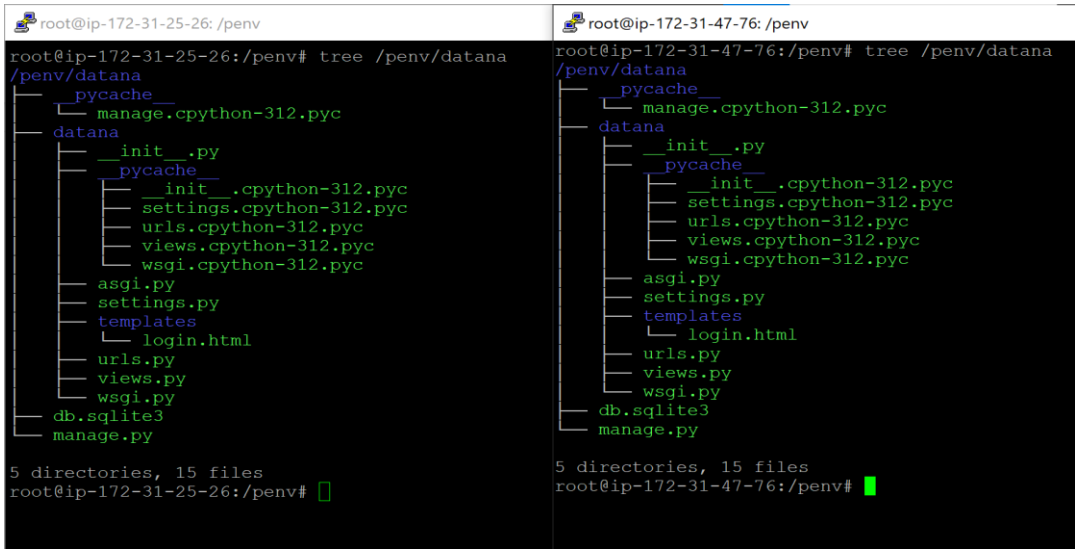

Figure 10. Comparing synced data from Server01 to Server02

4. Conclusions

This study proposed the SCOBDAF framework, a cloud layered security and continuity model designed to maintain uninterrupted Big Data operations during node and service failures. Addressing the documented limitations of existing Hadoop, Sqoop, and Python ecosystems as well as the shortcomings of traditional cloud recovery mechanisms, the framework introduces a structured three-layer architecture that integrates routing continuity, compute fault tolerance, and data persistence. Together with the Continuity Assurance Formula, these components provide a formal and measurable foundation for sustaining workflow level continuity during system disruptions.

The experimental deployment on AWS demonstrates the operational effectiveness of the framework. Routing continuity was maintained through ALB failover, compute resilience was achieved through automatic instance replacement by the ASG, and data persistence was ensured through EBS storage and rsync synchronization. All success indicators defined in the evaluation metrics were achieved, and none of the failure indicators were observed. The results confirm that the Continuity Assurance Formula was fully satisfied, as routing continuity, compute fault tolerance, and data persistence remained functional in all test scenarios.

Overall, the findings verify that SCOBDAF offers a practical and sound approach for enhancing availability, resilience, and data integrity in cloud based Big Data environments. The framework demonstrates robust behavior under real failure events and delivers a measurable model that can guide future system design and evaluation. Future research may extend the framework to multiple region architectures, incorporate predictive failure detection, or examine performance under larger distributed workloads. SCOBDAF provides a strong foundation for advancing continuity and security in modern Big Data systems.

Declarations

Acknowledgements
NA

Funding
NA

Contributions
All authors; Conceptualization, All authors; Investigation; All authors; Writing (Original Draft), All authors; Writing (Review and Editing) Supervision, All authors; Project Administration.

Ethics declarations
This article does not contain any studies with human participants or animals performed by any of the authors.

Data Availability Statement
All the datasets used in this study are publicly available from the CSE-CIC-IDS2018 [48] and UNSW-NB15 [49] repositories. The following processed data, feature mappings, model configurations, prediction outputs, explanation scores, risk-assessment results and validation scripts are provided in this study.

Use of Generative Artificial Intelligence
The authors declare that no generative artificial intelligence tools were used in the preparation of this manuscript.

Competing interests
All authors declare no competing interests.

References

[1] Cabrera, V. E., Barrientos-Blanco, J. A., Delgado, H., & Fadul-Pacheco, L. (2020). Symposium review: Real-time continuous decision making using big data on dairy farms. Journal of Dairy Science, 103(4), 3856–3866. https://doi.org/10.3168/jds.2019-17145
[2] Taleb, I., Serhani, M. A., Bouhaddioui, C., & Dssouli, R. (2021). Big data quality framework: a holistic approach to continuous quality management. Journal of Big Data, 8(1). https://doi.org/10.1186/s40537-021-00468-0
[3] Jannapureddy, R., Vien, Q.-T., Shah, P., & Trestian, R. (2019). An Auto-Scaling Framework for Analyzing Big Data in the Cloud Environment. Applied Sciences, 9(7), 1417. https://doi.org/10.3390/app9071417
[4] Cremer, F., Sheehan, B., Fortmann, M., Kia, A. N., Mullins, M., Murphy, F., & Materne, S. (2022). Cyber risk and cybersecurity: a systematic review of data availability. The Geneva Papers on Risk and Insurance - Issues and Practice, 47(3), 698–736. https://doi.org/10.1057/s41288-022-00266-6
[5] Ariyaluran Habeeb, R. A., Nasaruddin, F., Gani, A., Amanullah, M. A., Abaker Targio Hashem, I., Ahmed, E., & Imran, M. (2019). Clustering-based real-time anomaly detection—A breakthrough in big data technologies. Transactions on Emerging Telecommunications Technologies, 33(8). Portico. https://doi.org/10.1002/ett.3647
[6] Chang, V. (2015). Towards a Big Data system disaster recovery in a Private Cloud. Ad Hoc Networks, 35, 65–82. https://doi.org/10.1016/j.adhoc.2015.07.012
[7] Singh, K., & Kaur, R. (2014). Hadoop: Addressing challenges of Big Data. 2014 IEEE International Advance Computing Conference (IACC), 686–689. https://doi.org/10.1109/iadcc.2014.6779407
[8] Awaysheh, F. M., Aladwan, M. N., Alazab, M., Alawadi, S., Cabaleiro, J. C., & Pena, T. F. (2022). Security by Design for Big Data Frameworks Over Cloud Computing. IEEE Transactions on Engineering Management, 69(6), 3676–3693. https://doi.org/10.1109/tem.2020.3045661
[9] Taylor, R. C. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinformatics, 11(S12). https://doi.org/10.1186/1471-2105-11-s12-s1
[10] Zhang, Y., & Xu, X. (2021). Equilibrium analysis of cloud user request based on the Markov queue with variable vacation and vacation interruption. RAIRO - Operations Research, 55(5), 2807–2825. https://doi.org/10.1051/ro/2021130
[11] Su, C.-J., & Huang, S.-F. (2018). Real-time big data analytics for hard disk drive predictive maintenance. Computers &Amp; Electrical Engineering, 71, 93–101. https://doi.org/10.1016/j.compeleceng.2018.07.025
[12] Al-Masadeh, M., & Al-Zahrani, F. (2021). The Role of Vertical Elastic Namenode in Handling Big Data in Small Files. Arabian Journal for Science and Engineering, 47(2), 1609–1617. https://doi.org/10.1007/s13369-021-05929-5
[13] Diaconita, V., Bologa, A.-R., & Bologa, R. (2018). Hadoop Oriented Smart Cities Architecture. Sensors, 18(4), 1181. https://doi.org/10.3390/s18041181
[14] Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data, 2(1). https://doi.org/10.1186/s40537-015-0032-1
[15] Kim, J.-B., Choi, J.-B., & Jung, E.-S. (2024). Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM. Applied Sciences, 14(9), 3914. https://doi.org/10.3390/app14093914
[16] Hakimzadeh, K., Peiro Sajjad, H., & Dowling, J. (2014). Scaling HDFS with a Strongly Consistent Relational Model for Metadata. Distributed Applications and Interoperable Systems, 38–51. https://doi.org/10.1007/978-3-662-43352-2_4
[17] Sarathy, V., Narayan, P., & Mikkilineni, R. (2010). Next Generation Cloud Computing Architecture: Enabling Real-Time Dynamism for Shared Distributed Physical Infrastructure. 2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 48–53. https://doi.org/10.1109/wetice.2010.14
[18] Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W.-G., & Wu, Y. (2016). Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies. IEEE Transactions on Parallel and Distributed Systems, 27(1), 130–143. https://doi.org/10.1109/tpds.2015.2398438
[19] Sundararajan, T. V. P., Sumithra, M. G., & Maheswar, R. (2014). A Novel Smart Routing Protocol for Remote Health Monitoring in Medical Wireless Networks. Journal of Healthcare Engineering, 5(1), 95–122. Portico. https://doi.org/10.1260/2040-2295.5.1.95
[20] Brogi, A., Carrasco, J., Durán, F., Pimentel, E., & Soldani, J. (2021). Self-healing trans-cloud applications. Computing, 104(4), 809–833. https://doi.org/10.1007/s00607-021-00977-z

[21] Bakhshi, Z., Rodriguez-Navas, G., & Hansson, H. (2023). Analyzing the performance of persistent storage for fault-tolerant stateful fog applications. Journal of Systems Architecture, 144, 103004. https://doi.org/10.1016/j.sysarc.2023.103004
[22] Amjad, T., Sher, M., & Daud, A. (2012). A survey of dynamic replication strategies for improving data availability in data grids. Future Generation Computer Systems, 28(2), 337–349. https://doi.org/10.1016/j.future.2011.06.009
[23] Ladin, R., Liskov, B., Shrira, L., & Ghemawat, S. (1992). Providing high availability using lazy replication. ACM Transactions on Computer Systems, 10(4), 360–391. https://doi.org/10.1145/138873.138877
[24] Wilson, K. S., & Kiy, M. A. (2014). Some Fundamental Cybersecurity Concepts. IEEE Access, 2, 116–124. https://doi.org/10.1109/access.2014.2305658
[25] Reiter, M. K., Birman, K. P., & van Renesse, R. (1994). A security architecture for fault-tolerant systems. ACM Transactions on Computer Systems, 12(4), 340–371. https://doi.org/10.1145/195792.195823
[26] Huseinovic, A., Mrdovic, S., Bicakci, K., & Uludag, S. (2020). A Survey of Denial-of-Service Attacks and Solutions in the Smart Grid. IEEE Access, 8, 177447–177470. https://doi.org/10.1109/access.2020.3026923
[27] Aldossary, S., & Allen, W. (2016). Data Security, Privacy, Availability and Integrity in Cloud Computing: Issues and Current Solutions. International Journal of Advanced Computer Science and Applications, 7(4). https://doi.org/10.14569/ijacsa.2016.070464
[28] Hsieh, J.-C., Li, A.-H., & Yang, C.-C. (2013). Mobile, Cloud, and Big Data Computing: Contributions, Challenges, and New Directions in Telecardiology. International Journal of Environmental Research and Public Health, 10(11), 6131–6153. https://doi.org/10.3390/ijerph10116131
[29] Endo, P. T., Rodrigues, M., Gonçalves, G. E., Kelner, J., Sadok, D. H., & Curescu, C. (2016). High availability in clouds: systematic review and research challenges. Journal of Cloud Computing, 5(1). https://doi.org/10.1186/s13677-016-0066-8
[30] Furnadzhiev, R., Shopov, M., & Kakanakov, N. (2025). Efficient Orchestration of Distributed Workloads in Multi-Region Kubernetes Cluster. Computers, 14(4), 114. https://doi.org/10.3390/computers14040114
[31] Hughes, J., & Cybenko, G. (2013). Quantitative Metrics and Risk Assessment: The Three Tenets Model of Cybersecurity. Technology Innovation Management Review, 3(8), 15–24. https://doi.org/10.22215/timreview/712