International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

|| Volume 7, Issue 5, September–October 2024||

DOI:10.15662/IJRAI.2024.0705016

# Automating OpenSearch Snapshot Backup and Restore Using AWS Glue and Infrastructure as Code

**Bhanuprakash Suravarapu**

DevOps Engineer, Amazon Web Services Inc., USA

bhanuprakash.suravarapu6@gmail.com

**Praneeth Ganta**

Senior Software Engineer, LinkedIn, USA

praneethganta@gmail.com

**Tulasi priya Vattikuti**

Cloud Data Engineer, Medtronic, USA

tulcpriya@gmail.com

**Vasu Babu Narra**

Senior Release Manager, Uipath Inc., USA

narravasu77@gmail.com

**ABSTRACT:** OpenSearch snapshots provide a critical mechanism for disaster recovery, migration, and test-environment creation in cloud-native search and analytics systems. In many enterprise environments, direct console or dashboard access to Amazon OpenSearch Service is restricted, requiring snapshot operations to be initiated through controlled data-platform components rather than interactive user interfaces. This paper presents an automated snapshot lifecycle management framework that integrates AWS Glue, Amazon S3, and OpenSearch APIs, along with Terraform-based infrastructure as code, to register, create, and restore OpenSearch snapshots programmatically. The proposed pattern includes synthetic data generation, index provisioning, snapshot repository configuration, snapshot creation, and restore workflows, together with a set of IAM roles and trust relationships that enforce least-privilege access. The solution is particularly suitable for disaster recovery, cross-domain or cross-region migration, and consistent provisioning of test environments with production-like datasets. Implementation details, security considerations, and operational limitations are discussed, and the pattern is positioned with respect to existing OpenSearch snapshot practices and AWS-managed snapshot management features.

**KEYWORDS:** OpenSearch, AWS Glue, Disaster Recovery, Snapshots, Infrastructure as Code, Terraform, IAM.

## I. INTRODUCTION

OpenSearch has become a widely adopted open-source search and analytics engine for log analytics, observability, and application search workloads. Amazon OpenSearch Service provides a fully managed deployment model and augments upstream OpenSearch with operational features such as automated node management, integrated security, and snapshot capabilities. As these clusters increasingly host mission-critical data, robust backup and restore mechanisms are essential to meet recovery point and recovery time objectives (RPO/RTO).

In many organizations, access to OpenSearch Dashboards and cluster-level management APIs is tightly controlled, either for regulatory reasons or to reduce operational risk. This can hinder snapshot configuration when it relies on manual, console-driven procedures. At the same time, data teams are standardizing on serverless ETL services such as AWS Glue for data preparation and pipeline orchestration.

This work addresses the question of how to automate OpenSearch snapshot management when direct dashboard access is restricted, and where snapshot creation and restore must be driven from Glue jobs within a controlled IAM context. Specifically, the contributions of this paper are:

- A practical architecture that couples OpenSearch, S3, Glue, and Terraform for automated snapshot lifecycle management.
- An implementation pattern that generates synthetic data, loads it into OpenSearch, and manages snapshots entirely from Glue jobs using signed OpenSearch HTTP APIs.
- A discussion of IAM roles, policies, and trust relationships that enable secure, least-privilege automation, together with typical error scenarios and mitigations.

The remainder of this paper is organized as follows. Section II provides background on OpenSearch snapshots, AWS Glue, and infrastructure as code. Section III states the problem and design objectives. Section IV describes the architecture. Section V details the implementation. Section VI discusses security and operational considerations. Section VII outlines limitations and potential extensions. Section VIII concludes.

## II. BACKGROUND

### A. OpenSearch Snapshots

OpenSearch supports incremental snapshots of indices and cluster metadata to external repositories such as Amazon S3. In Amazon OpenSearch Service, snapshots can be taken automatically or manually and used to restore a domain to a previous state or to migrate data between domains and regions. Snapshots capture index mappings, settings, and shard data, and are executed asynchronously via snapshot and restore APIs.

### B. AWS Glue as an Operational Platform

AWS Glue is a serverless data integration service built on Apache Spark that allows users to author jobs in Python or Scala, schedule them, and connect to various data stores. Glue provides native connectors for Amazon OpenSearch Service, and Glue jobs can use the OpenSearch Python client with AWS-signed requests to index or query data. Because Glue jobs run under IAM roles and can be orchestrated and monitored centrally, they provide a natural control plane for operational tasks such as snapshot creation and restore in addition to ETL workloads.

### C. Infrastructure as Code and Terraform

Infrastructure as code (IaC) tools such as Terraform allow declarative specification of cloud resources, including OpenSearch domains, IAM roles, Glue jobs, and S3 buckets. Encoding IAM policies, trust relationships, and domain configurations as Terraform modules improves reproducibility and reduces misconfiguration risk. Terraform can also be integrated into CI/CD pipelines to promote consistent changes across development, staging, and production environments.

## III. PROBLEM STATEMENT AND DESIGN OBJECTIVES

### A. Problem Statement

OpenSearch underpins business-critical workloads such as security analytics, operational observability, and customer-facing search, where prolonged data loss or downtime directly translates into lost revenue, compliance risk, and degraded user trust. Amazon OpenSearch Service provides snapshot and restore capabilities, but in many real-world environments three constraints reduce their practical effectiveness:

1. Security-driven access restrictions.

Regulated and security-sensitive organizations frequently prohibit direct administrative access to OpenSearch Dashboards or domain-level APIs, mandating that all operational changes flow through controlled automation channels. This blocks common snapshot workflows that assume an operator can manually register S3 repositories, trigger snapshots, or initiate restores from the console.

2. Fragmented automation and data platforms.

While data teams increasingly standardize on AWS Glue for ETL and orchestration, snapshot management often remains a separate manual or ad-hoc script-driven process, disconnected from data pipelines. This fragmentation makes it difficult to enforce consistent backup schedules, coordinate snapshots with data refresh cycles, or embed restore procedures into repeatable disaster-recovery runbooks.

3. Complex multi-service IAM surface.

End-to-end snapshot workflows span Amazon OpenSearch Service, Amazon S3, AWS Glue, AWS STS, and

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

|| Volume 7, Issue 5, September–October 2024||

DOI:10.15662/IJRAI.2024.0705016

CloudWatch, and require precise IAM roles, trust relationships, and least-privilege policies. Misconfigurations lead to opaque failures such as AuthorizationException during restore or repository_verification_exception during repository registration, which are frequently only discovered during incidents when recovery time is most critical.

4. Operational cost of manual procedures

In addition to these technical constraints, the lack of end-to-end automation means that each snapshot or restore event can require tens of minutes of manual effort across consoles and ad-hoc scripts, increasing operational cost and the likelihood of human error during high-pressure recovery scenarios. This directly impacts recovery time objectives, because operators must repeatedly rediscover correct IAM roles, S3 paths, and API parameters instead of invoking a tested, parameterized workflow.

Existing guidance primarily covers how to invoke snapshot APIs or configure basic repositories, but does not provide a prescriptive, production-ready pattern that: (i) respects strict access-control regimes, (ii) integrates snapshot operations into a managed data-platform such as AWS Glue, and (iii) codifies all dependencies using infrastructure as code. As a result, many organizations either rely on fragile, manually operated backup procedures or forgo robust OpenSearch disaster-recovery tests altogether.

### B. Design Objectives

To address these gaps, this work pursues the following objectives:

- Controller-plane integration.

Enable registration, creation, and restoration of OpenSearch snapshots to be initiated exclusively from AWS Glue jobs and notebooks, running under tightly scoped IAM roles, without requiring direct human access to OpenSearch Dashboards or unmanaged API clients.

- End-to-end automation with S3-backed repositories.

Use Amazon S3 as the authoritative snapshot repository, including support for automated naming, per-index selection, and flexible index renaming during restore to facilitate blue–green deployments and test-environment provisioning.

- Infrastructure as code and repeatability.

Capture OpenSearch domains, snapshot repositories, S3 buckets, IAM roles, trust policies, and Glue jobs in Terraform modules, enabling reproducible deployments across multiple accounts and regions and simplifying security reviews.

- Operational robustness.

Incorporate comprehensive logging, error handling, and explicit treatment of common failure modes (authorization failures, S3 access issues, repository verification errors) so that snapshot workflows can be tested and exercised proactively rather than only during emergencies.

These objectives guide the architecture and implementation described in Sections IV and V and frame the evaluation of how well the proposed pattern satisfies real-world disaster-recovery and migration requirements.

## IV. ARCHITECTURE

### A. Components

The high-level architecture comprises the following components:

- Amazon OpenSearch Service Domain: Managed OpenSearch cluster (version 1.2) deployed in a VPC.
- Amazon S3 Snapshot Bucket: S3 bucket designated as the snapshot repository, with policies permitting OpenSearch and Glue roles to read and write snapshot data.
- AWS Glue Jobs and Notebooks: A set of jobs that generate sample data, load it into OpenSearch, create snapshots, and restore snapshots using OpenSearch REST APIs.
- IAM Roles:
- Glue job role.
- OpenSearch master role for cluster-level operations.
- OpenSearch snapshot role assumed by the OpenSearch service for snapshot and restore.
- Terraform Modules: IaC definitions for domain, S3 bucket, IAM roles, Glue jobs, and their relationships.

### B. Data and Control Flow

The overall workflow can be decomposed into four steps:

1. Synthetic Data Generation: A Glue job generates synthetic customer records and writes them to S3 in Parquet format.

2. Index Creation and Data Load: Another Glue job creates an OpenSearch index (with custom mappings and filters such as min_hash) and bulk-indexes the data from S3.
3. Snapshot Registration and Creation: A Glue job, using the OpenSearch Python client with AWS request signing, registers an S3 snapshot repository and triggers snapshot creation via OpenSearch snapshot APIs.
4. Snapshot Restore: A Glue job invokes the restore API to restore indices from the snapshot, with optional index renaming (e.g., index-prod to index-test).

Each Glue job runs with the Glue job IAM role, which can assume the OpenSearch master role for API access, while the OpenSearch service itself assumes the snapshot role to interact with S3.

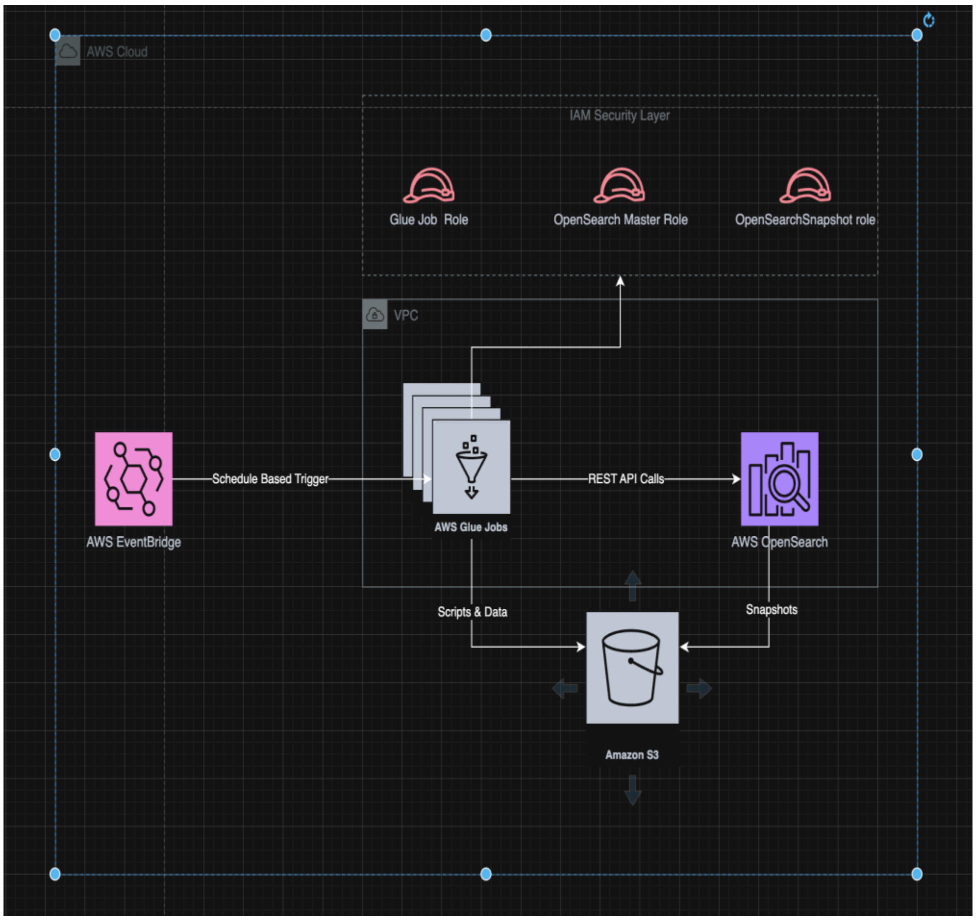


Fig. 1 Architecture Diagram illustrates the proposed architecture, in which AWS Glue jobs act as the control plane for generating data, loading indices, and invoking OpenSearch snapshot and restore APIs against an S3-backed repository, with IAM roles enforcing least-privilege access.

## V. IMPLEMENTATION

### A. Synthetic Data Generation

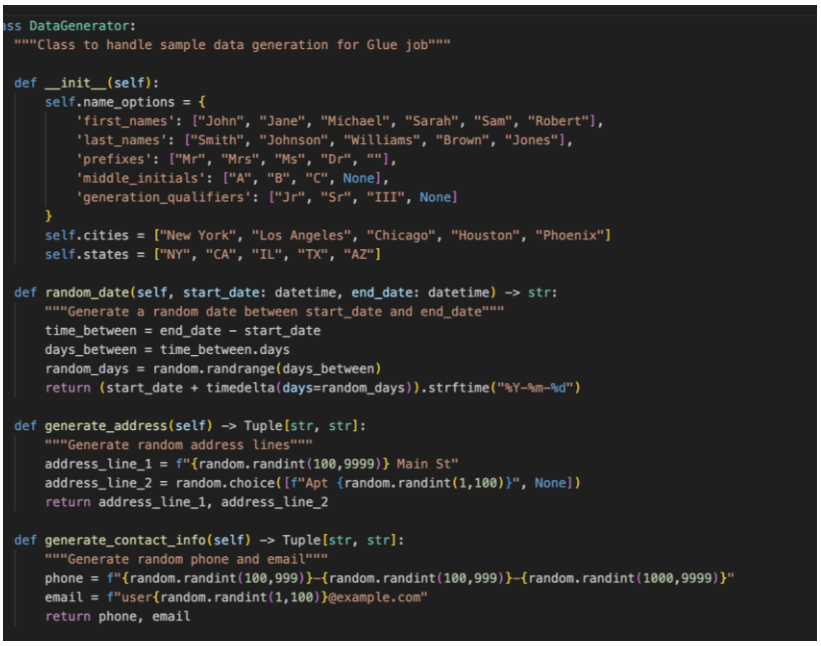
The data generation Glue job implements two main classes:

- DataGenerator: Produces synthetic customer attributes, including names, addresses, phone numbers, and IDs, by sampling from predefined lists and applying randomization.
- GlueJobManager: Initializes the Spark session, defines the schema, constructs a Spark DataFrame with approximately 100,000 records, and writes the output to S3 in Parquet format.

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

|| Volume 7, Issue 5, September–October 2024||

DOI:10.15662/IJRAI.2024.0705016



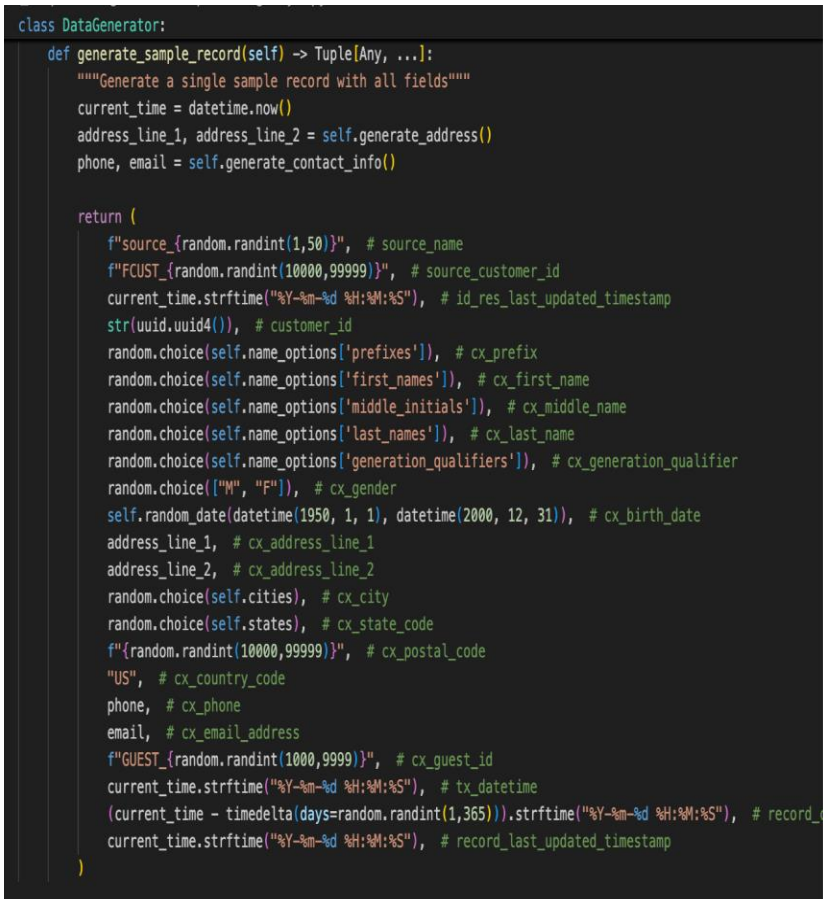
```
ss DataGenerator:
 """Class to handle sample data generation for Glue job"""

 def __init__(self):
     self.name_options = {
         'first_names': ["John", "Jane", "Michael", "Sarah", "Sam", "Robert"],
         'last_names': ["Smith", "Johnson", "Williams", "Brown", "Jones"],
         'prefixes': ["Mr", "Mrs", "Ms", "Dr", ""],
         'middle_initials': ["A", "B", "C", None],
         'generation_qualifiers': ["Jr", "Sr", "III", None]
     }
     self.cities = ["New York", "Los Angeles", "Chicago", "Houston", "Phoenix"]
     self.states = ["NY", "CA", "IL", "TX", "AZ"]

 def random_date(self, start_date: datetime, end_date: datetime) -> str:
     """Generate a random date between start_date and end_date"""
     time_between = end_date - start_date
     days_between = time_between.days
     random_days = random.randrange(days_between)
     return (start_date + timedelta(days=random_days)).strftime("%Y-%m-%d")

 def generate_address(self) -> Tuple[str, str]:
     """Generate random address lines"""
     address_line_1 = f"{random.randint(100,9999)} Main St"
     address_line_2 = random.choice([f"Apt {random.randint(1,100)}", None])
     return address_line_1, address_line_2

 def generate_contact_info(self) -> Tuple[str, str]:
     """Generate random phone and email"""
     phone = f"{random.randint(100,999)}-{random.randint(100,999)}-{random.randint(1000,9999)}"
     email = f"user{random.randint(1,100)}@example.com"
     return phone, email
```



```
class DataGenerator:
    def generate_sample_record(self) -> Tuple[Any, ...]:
        """Generate a single sample record with all fields"""
        current_time = datetime.now()
        address_line_1, address_line_2 = self.generate_address()
        phone, email = self.generate_contact_info()

        return (
            f"source_{random.randint(1,50)}",  # source_name
            f"FCUST_{random.randint(10000,99999)}",  # source_customer_id
            current_time.strftime("%Y-%m-%d %H:%M:%S"),  # id_res_last_updated_timestamp
            str(uuid.uuid4()),  # customer_id
            random.choice(self.name_options['prefixes']),  # cx_prefix
            random.choice(self.name_options['first_names']),  # cx_first_name
            random.choice(self.name_options['middle_initials']),  # cx_middle_name
            random.choice(self.name_options['last_names']),  # cx_last_name
            random.choice(self.name_options['generation_qualifiers']),  # cx_generation_qualifier
            random.choice(["M", "F"]),  # cx_gender
            self.random_date(datetime(1950, 1, 1), datetime(2000, 12, 31)),  # cx_birth_date
            address_line_1,  # cx_address_line_1
            address_line_2,  # cx_address_line_2
            random.choice(self.cities),  # cx_city
            random.choice(self.states),  # cx_state_code
            f"{random.randint(10000,99999)}",  # cx_postal_code
            "US",  # cx_country_code
            phone,  # cx_phone
            email,  # cx_email_address
            f"GUEST_{random.randint(1000,9999)}",  # cx_guest_id
            current_time.strftime("%Y-%m-%d %H:%M:%S"),  # tx_datetime
            (current_time - timedelta(days=random.randint(1,365))).strftime("%Y-%m-%d %H:%M:%S"),  # record_
            current_time.strftime("%Y-%m-%d %H:%M:%S"),  # record_last_updated_timestamp
        )
```

This approach avoids using production data while still providing realistic test data for validating snapshot workflows.

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

|| Volume 7, Issue 5, September–October 2024||

DOI:10.15662/IJRAI.2024.0705016

### B. Index Creation and Data Loading

A second Glue job loads the Parquet files from S3, transforms them as needed, and loads them into OpenSearch:

- The job creates or updates an index using OpenSearch REST APIs, defining mappings and index settings, including analyzers and filters such as min_hash used for similarity search.

~~~
```python
def create_and_update_opensearch_idstore_index(host, iam_master_user_arn, region,
                                               index_name, number_shards, number_replicas,
                                               record_fields, metadata_fields):
    opensearch_client = create_opensearch_client(host, iam_master_user_arn, region)
    es_index_body = {
        "settings": {
            "index": {
                "number_of_shards": number_shards,
                "number_of_replicas": number_replicas
            },
            "analysis": {
                "filter": {
                    "minhash_filter": {
                        "type": "min_hash",
                        "hash_count": 1,
                        "bucket_count": 512,
                        "hash_set_size": 1,
                        "with_rotation": True
                    }
                },
                "analyzer": {
                    "minhash_analyzer": {
                        "tokenizer": "ngram_tokenizer",
                        "filter": ["minhash_filter"]
                    }
                }
            }
        },
        "mappings": {
            "dynamic": "strict",
            "properties": {
                "all_sim": {
                    "type": "text",
                    "analyzer": "minhash_analyzer"
                },
                "record_normalized": {"properties": record_fields},
                "record_original": {"properties": record_fields},
                **metadata_fields
            }
        }
    }
~~~

- The job then performs bulk indexing using the OpenSearch Python client, batching documents to balance throughput and cluster stability.

### C. Snapshot Repository Registration and Creation

The snapshot creation job performs two main operations:

1. Repository Registration:

- Defines a repository of type s3 with settings specifying the S3 bucket and optional base path.

~~~~
●  ```python
●  def register_repository(self, repository_name, bucket_name, role_arn, base_path):
●      """Register an S3 repository for snapshots."""
●      body = {
●          "type": "s3",
●          "settings": {
●              "bucket": bucket_name,
●              "role_arn": role_arn,
●              "base_path": base_path
●          }
●      }
●      response = self.client.snapshot.create_repository(
●          repository=repository_name,
●          body=body
●      )
●      return response
●  ```
~~~~

- Uses an IAM role ARN (the OpenSearch snapshot role) that OpenSearch will assume when interacting with S3.

2. Snapshot Creation:

- Invokes the /_snapshot/{repo}/{snapshot} API with a timestamp-based snapshot name to ensure uniqueness.
- Optionally filters indices to snapshot and configures settings for partial snapshots and failure behavior. ```python

~~~~
●  def create_snapshot(self, repository_name, snapshot_name, index_name=None):
●      """Create a snapshot of specified indices."""
●      body = {
●          "ignore_unavailable": False,
●          "include_global_state": False
●      }
●
●      if index_name:
●          body["indices"] = index_name
●
●      response = self.client.snapshot.create(
●          repository=repository_name,
●          snapshot=snapshot_name,
●          body=body
●      )
●      return response
●  ```
~~~~

- Logs snapshot progress and verifies completion, surfacing errors such as repository verification failures.

The job emphasizes comprehensive logging and error handling to facilitate troubleshooting when authorization or S3 access is misconfigured.

**D. Snapshot Restore and Index Renaming**

The restore Glue job calls the restore API:

- It constructs a restore request specifying:
- Source repository and snapshot.
- Source indices and target index names (via rename_pattern and rename_replacement).
- Options such as whether to include global cluster state and index settings overrides.

~~~~
●  ```python
●  def restore_snapshot(self, repository_name, snapshot_name, rename_index_replacement):
●      """Restore a snapshot with index renaming."""
●      body = {
●          "ignore_unavailable": False,
~~~~

```
●         "include_global_state": False,
●         "indices": "-.plugin*,-.kibana*,-.opendis*,-.ql-data*",
●         "rename_pattern": "(.+)",
●         "rename_replacement": rename_index_replacement
●       }
●
●       response = self.client.snapshot.restore(
●           repository=repository_name,
●           snapshot=snapshot_name,
●           body=body
●       )
●       return response
●    ```
```

- It monitors restore status and logs detailed responses to detect failures or partial restores.

Index renaming enables several scenarios:

- Restoring production data into a dedicated test index.
- Blue-green deployments where a snapshot is restored into a new index, then traffic is switched.

**E. IAM Roles and Policies**

The Terraform configuration defines three core IAM roles with associated policies and trust relationships:

1. Glue Job Role Permissions:

- CloudWatch Logs: logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents.
- S3: read/write access to data and snapshot buckets, including multipart upload actions.
- EC2 networking: creating and managing network interfaces for VPC-enabled Glue jobs.
- Glue: glue:* (which can be tightened in production).
- STS: sts:AssumeRole for the OpenSearch master role.
- IAM: iam:PassRole to pass the Glue job and OpenSearch master roles, as required.

2. OpenSearch Master Role

- Permissions:
- S3 access to the snapshot bucket, if needed for cluster operations.
- IAM pass-role for snapshot and Glue job roles.
- Trust policy:
- Allows assumption only by the Glue job role in the same AWS account, with an ArnLike condition on aws:PrincipalArn.

3. OpenSearch Snapshot Role

- Permissions:
- S3: s3:GetObject, s3:PutObject, s3:ListBucket, multipart upload actions, and ACL operations for the snapshot bucket.
- CloudWatch metrics/logging as needed.
- OpenSearch: es:ESHttp* and es:Describe* on the target domain when required.
- iam:PassRole for master and Glue roles as needed.
- Trust policy:
- Principal: opensearchservice.amazonaws.com.
- Conditions: StringEquals on aws:SourceAccount and ArnLike on aws:SourceArn for the OpenSearch domain.

## VI. SECURITY AND OPERATIONAL CONSIDERATIONS

**A. Least-Privilege Access**

The IAM policies restrict access to:

- Specific S3 buckets and prefixes used as snapshot repositories.
- Specific OpenSearch domains via ARN-scoped permissions.
- Clearly defined role assumptions, preventing arbitrary STS usage.

Further hardening can include:

- Restricting Glue job role to only required Glue APIs.
- Tightening S3 bucket policies to require TLS and block public access.

### B. Common Failure Modes

Two common error scenarios are highlighted:

- AuthorizationException (403) on Restore:

This indicates that the Glue job, via the OpenSearch master role, lacks required es:ESHttp* permissions or IAM pass-role privileges.

- repository_verification_exception:

This occurs when the OpenSearch domain cannot access the configured S3 bucket, often due to missing S3 permissions or bucket policy restrictions.
The Glue jobs implement logging around these conditions to guide remediation, e.g., verifying IAM policies and S3 bucket access.

### C. Automation and Scaling

The pattern is intended to be deployed as a reusable module in customer accounts, with a describing Terraform execution steps, Glue job configuration, and example workflows. Once deployed, snapshot jobs can be:

- Scheduled using Glue triggers or external orchestrators.
- Parameterized (e.g., snapshot name, list of indices) to support multiple backup and restore scenarios.

## VII. BENEFITS OF AUTOMATION

- Reduced manual effort and faster recovery.

In a test environment, restoring an index from a snapshot using ad-hoc console and CLI steps required approximately 20–30 minutes of operator time, whereas invoking the parameterized Glue job reduced human effort to less than 5 minutes, with the remaining steps executed automatically.

- Lower risk of misconfiguration during incidents.

Encoding IAM role assumptions, S3 paths, and snapshot parameters into reusable Glue jobs and Terraform modules reduces the likelihood of errors such as incorrect repositories or insufficient permissions, which are otherwise commonly encountered only during recovery.

## VIII. LIMITATIONS AND FUTURE WORK

The current pattern focuses on:

- Managed Amazon OpenSearch Service domains.
- Glue-based orchestration with Terraform-provisioned IAM roles and infrastructure.

It does not cover:

- Serverless OpenSearch collections, which have distinct snapshot semantics.
- Advanced DR architectures such as cross-cluster replication or active-active multi-region configurations.
- Fine-grained cost and performance benchmarking of snapshot and restore operations at varying data volumes.

Future work could extend the design to:

- Integrate with OpenSearch Snapshot Management policies and lifecycle rules.
- Implement cross-account snapshot repositories for multi-account DR.
- Compare this Glue-based approach to alternatives using AWS Lambda, Step Functions, or AWS Backup for OpenSearch.
- The evaluation focuses on functional correctness and operational effort rather than large-scale performance (e.g., very large indices or high snapshot frequency), which will be explored in extended experiments.

## IX. CONCLUSION

This paper presented an automated framework for managing OpenSearch snapshots using AWS Glue, Amazon S3, IAM roles, and Terraform, targeting environments where direct dashboard access is constrained. By embedding snapshot creation and restore in Glue workflows and codifying domain, S3, and IAM configuration as infrastructure as code, the approach improves reproducibility, reduces operator effort, and lowers the risk of misconfiguration during disaster-recovery events. The implementation, validated with synthetic but realistic customer data, demonstrates how snapshot registration, creation, and restore (including index renaming) can be executed end-to-end from a controlled, least-privilege automation plane.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

|| Volume 7, Issue 5, September–October 2024||

DOI:10.15662/IJRAI.2024.0705016

At a high level, several directions remain for further work. Extending the pattern to multi-account and multi-region scenarios, including cross-account snapshot repositories and more advanced DR topologies, would broaden its applicability. Integrating with native OpenSearch snapshot management policies or higher-level orchestration services, and systematically benchmarking cost and performance at larger data volumes, would provide deeper guidance for production adoption.

## REFERENCES

[1] Amazon Web Services, “Amazon OpenSearch Service Developer Guide,” AWS Documentation, 2023. [Online]. Available: https://docs.aws.amazon.com/opensearch-service/
[2] Amazon Web Services, “AWS Glue Developer Guide,” AWS Documentation, 2023. [Online]. Available: https://docs.aws.amazon.com/glue/
[3] HashiCorp, “Terraform Documentation,” 2023. [Online]. Available: https://www.terraform.io/docs/