System Design
Real-Time CDC Pipeline: Data transfer from Amazon RDS → Elasticsearch

1. Overview

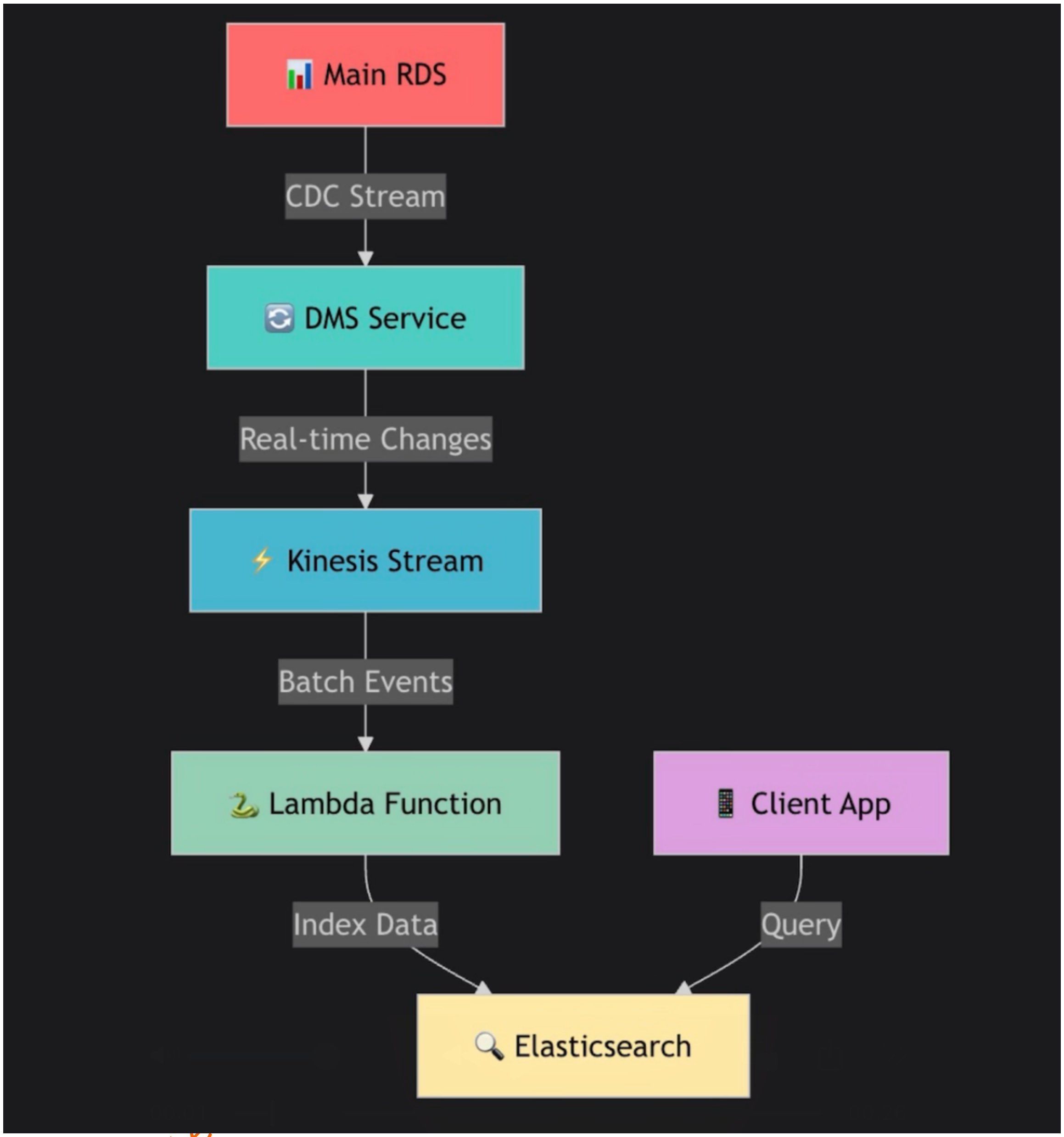
This document describes the architecture for a real-time Change Data Capture (CDC) pipeline that captures data modifications from Amazon RDS and delivers them into Elasticsearch. The goal is to enable sub-second search, analytics, and offloading of query load from the primary operational database.

Business Objectives

- Enable low-latency full-text and structured search for user-facing applications
- Ensure data consistency between source of truth (RDS) and Elasticsearch
- Maintain real-time analytics visibility into operational data
- Decouple storage and compute to improve scalability

2. High-Level Architecture

The pipeline uses AWS DMS for CDC, Kinesis for real-time event streaming, Lambda for transformation and indexing, and Elasticsearch as the search indexing layer.



2.2 Component Specifications

2.2.1 Source Database (Amazon RDS)

Database Engines Supported: MySQL, PostgreSQL, Oracle

CDC Requirements:

- MySQL: binlog_format = ROW, binlog_row_image = FULL
- PostgreSQL: wal_level = logical, rds.logical_replication = 1

Configuration: Custom Parameter Groups for CDC settings

2.2.2 AWS DMS (Data Migration Service)

Instance Type: dms.t3.medium to dms.r5.large (scale based on load)

Task Configuration:

- Migration Type: Replicate data changes only (CDC)
- Target Table Preparation: Truncate (initial load) / Do nothing (CDC only)
- LOB Settings: Limited LOB mode (for large objects)

2.2.3 Kinesis Data Streams

Shard Configuration:

- 1 shard = ~1MB/sec write capacity, ~2MB/sec read capacity
- Scale based on peak write throughput: TotalShards = PeakThroughput / 1000

Retention Period: 24 hours (configurable up to 365 days)
Encryption: KMS server-side encryption

2.2.4 AWS Lambda (Stream Processor)

Runtime: Python 3.9 / Node.js 16.x
Memory: 512MB - 3008MB (based on batch size)
Timeout: 5 minutes (for large batches)
Concurrency: Reserved concurrency = Number of Kinesis shards

2.2.5 Elasticsearch Cluster

Version: Amazon Elasticsearch 7.10+ / OpenSearch 1.0+
Instance Type: r6g.large.search (data nodes), c6g.large.search (master nodes)

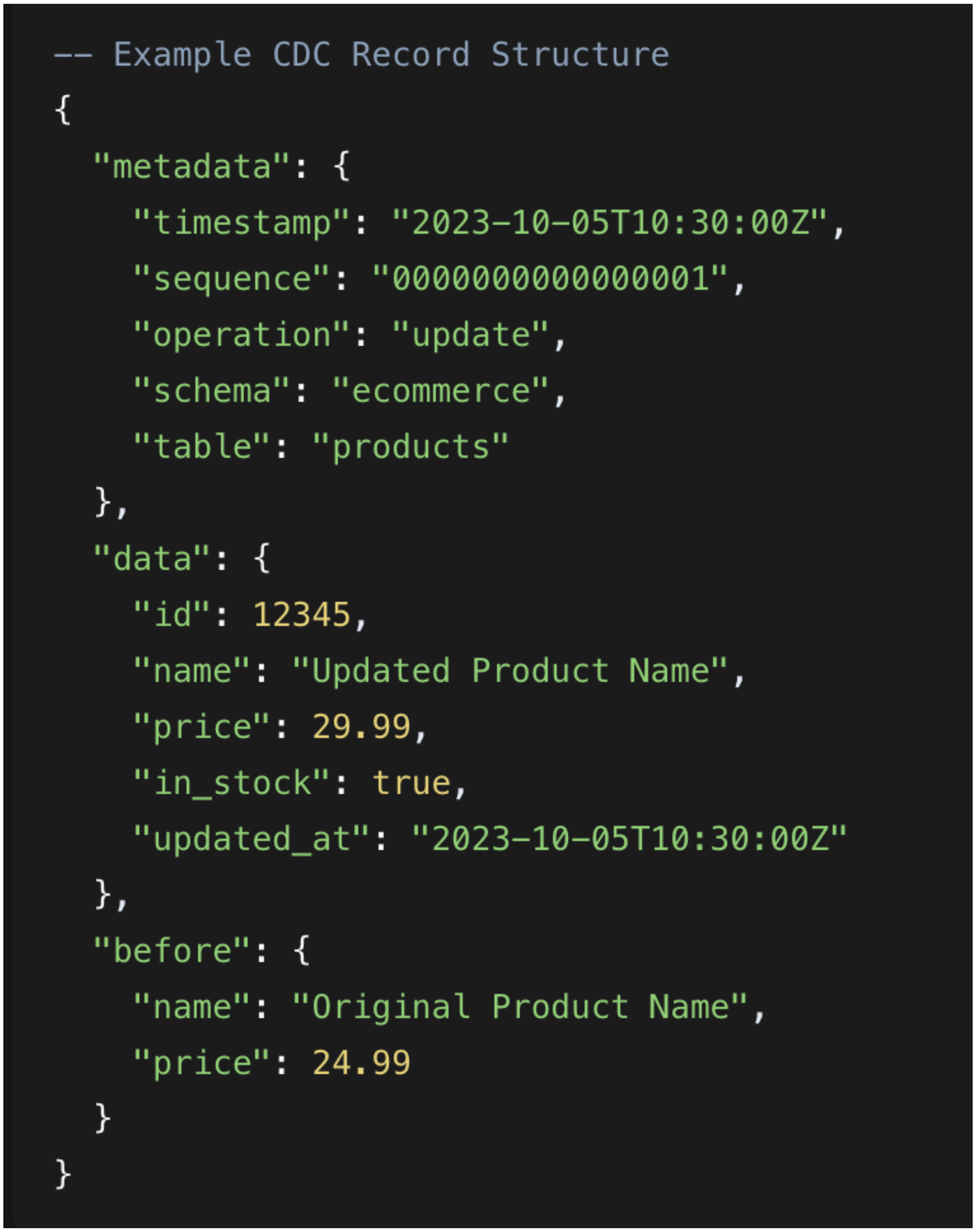
Sharding Strategy:

- Primary shards: Based on data volume (20-50GB per shard)
- Replica shards: 1 (for high availability)

3. Data Processing Logic

Example Lambda processing logic for indexing:

3.1 Change Capture Phase



```
-- Example CDC Record Structure
{
  "metadata": {
    "timestamp": "2023-10-05T10:30:00Z",
    "sequence": "0000000000000001",
    "operation": "update",
    "schema": "ecommerce",
    "table": "products"
  },
  "data": {
    "id": 12345,
    "name": "Updated Product Name",
    "price": 29.99,
    "in_stock": true,
    "updated_at": "2023-10-05T10:30:00Z"
  },
  "before": {
    "name": "Original Product Name",
    "price": 24.99
  }
}
```

3.2 Stream Processing Logic

```python
def lambda_handler(event, context):
    records = []

    for record in event['Records']:
        # Decode Kinesis data
        payload = base64.b64decode(record['kinesis']['data']).decode('utf-8')
        change_event = json.loads(payload)

        # Transform DMS format to ES document
        es_action = transform_to_es_bulk_action(change_event)
        records.append(es_action)

    # Execute bulk request
    if records:
        response = es_client.bulk(
            body=records,
            index='products-index',
            _source=True
        )

        # Handle partial failures
        if response['errors']:
            handle_failed_items(response['items'])
```

3.3 Elasticsearch Mapping

```json
{
  "mappings": {
    "properties": {
      "id": {"type": "keyword"},
      "name": {
        "type": "text",
        "fields": {"keyword": {"type": "keyword"}}
      },
      "price": {"type": "scaled_float", "scaling_factor": 100},
      "in_stock": {"type": "boolean"},
      "created_at": {"type": "date"},
      "updated_at": {"type": "date"},
      "categories": {"type": "keyword"},
      "description": {"type": "text"}
    }
  },
  "settings": {
    "number_of_shards": 5,
    "number_of_replicas": 1,
    "refresh_interval": "1s"
  }
}
```

4. Security Implementation

4.1 Data Protection

Encryption at Rest:

- RDS: AES-256 encryption
- Kinesis: KMS customer-managed keys
- Elasticsearch: Node-to-node encryption, KMS for indices

Encryption in Transit:

- TLS 1.2+ for all inter-service communication
- VPC Endpoints for AWS services

5. Cost Optimization

5.1 Resource Right-Sizing

- DMS Instance: Start with dms.t3.medium, monitor CPUUtilization
- Kinesis Shards: Use on-demand capacity for variable workloads
- Elasticsearch Instances: Reserved Instances for steady-state workloads
- Lambda Memory: Optimize for execution time vs. memory cost

5.2 Storage Optimization

Elasticsearch Index Lifecycle:

- Hot tier: Current month (r6g.4xlarge.search)
- Warm tier: 1–6 months old (r6g.2xlarge.search)
- Cold tier: 6+ months old (UltraWarm / S3)

6. Success Metrics & SLIs

6.1 Service Level Indicators

- Data Freshness: P95 latency < 10 seconds
- Data Completeness: 99.9% of source records indexed in ES
- System Availability: 99.9% uptime for search functionality
- Query Performance: P95 search latency < 500ms

6.2 Monitoring Dashboard

- Data Flow Metrics: Records ingested, processed, indexed
- System Health: CPU, memory, disk utilization across all components
- Business Metrics: Search volume, popular queries, zero-result rates

7. Error Handling & Reliability

7.1 Failure Scenarios & Mitigation

Failure Point	Impact	Mitigation Strategy
DMS Task Failure	CDC stops	CloudWatch Alarms + Auto-restart tasks
Kinesis Throttling	Data loss	Monitor WriteProvisionedThroughputExceeded, auto-scale shards
Lambda Errors	Processing halted	DLQ for failed batches, retry logic
ES Cluster Issues	Indexing fails	Circuit breaker pattern, backoff retries
Network Partitions	Temporary outage	Idempotent processing, checkpointing

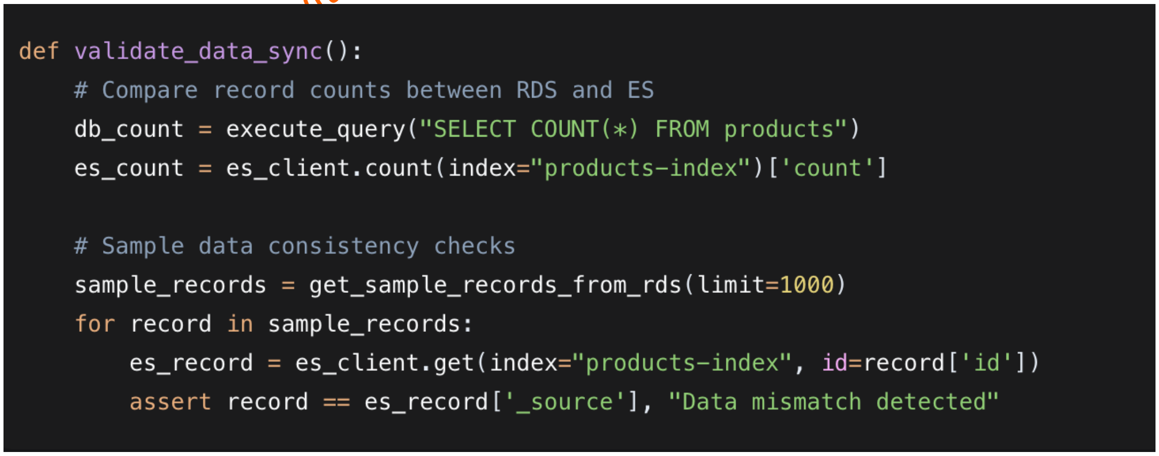
8. Disaster Recovery

RPO (Recovery Point Objective): 5 minutes
RTO (Recovery Time Objective): 30 minutes

Backup Strategy:

- **RDS:** Automated daily snapshots + transaction log retention
- **Elasticsearch:** Automated snapshot to S3 (every 30 minutes)
- **Recovery:** Point-in-time recovery for both systems

9. Data Validation



```python
def validate_data_sync():
    # Compare record counts between RDS and ES
    db_count = execute_query("SELECT COUNT(*) FROM products")
    es_count = es_client.count(index="products-index")['count']

    # Sample data consistency checks
    sample_records = get_sample_records_from_rds(limit=1000)
    for record in sample_records:
        es_record = es_client.get(index="products-index", id=record['id'])
        assert record == es_record['_source'], "Data mismatch detected"
```