POC Document — Employee Data Processing with AWS Lambda & S3 Prepared by: Harikrishna T POC Date: 27th October 2025, 3:00 PM IST Environment: AWS Console (No local setup)

1. Objective Demonstrate a fully serverless data ingestion and transformation pipeline using AWS services. The POC reads employee data from a MongoDB collection, converts it into a Parquet format, and uploads it to Amazon S3. An AWS Lambda function automatically triggers upon upload to validate, log, or further process the file.

2. Scope Included:

3. Reading employee data from MongoDB Atlas via AWS Lambda
4. Generating a Parquet file dynamically
5. Uploading the file to Amazon S3
6. Triggering a Lambda function automatically on file upload
7. Logging process details in CloudWatch

Excluded: - On-premise or local development setups - Any UI or client-side component (handled separately in later phase)

1. Technology Stack Component Technology / Service Used Database MongoDB Atlas File Format Apache Parquet Cloud Storage Amazon S3 Compute AWS Lambda (Python 3.9 runtime) Logging Amazon CloudWatch IAM For Lambda execution and S3 permissions AWS Console Used for all configurations and testing

2. Architecture Overview Flow Diagram (Conceptual): MongoDB Atlas → AWS Lambda (Python) → Parquet Conversion → Amazon S3 (Upload) → Lambda Trigger → CloudWatch Logs

Step-by-Step Flow with Code Snippets: Step 1: Lambda connects to MongoDB Atlas

```
import pymongo
import os
MONGO_URI = os.environ['MONGO_URI']
client = pymongo.MongoClient(MONGO_URI)
db = client['employee_db']
collection = db['employees']
```

Step 2: Fetch employee records

```
records = list(collection.find({}, {'_id': 0, 'empId': 1, 'name': 1, 'dept':
1, 'salary': 1}))
```

Step 3: Convert data into Parquet

```
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

df = pd.DataFrame(records)
```

```
table = pa.Table.from_pandas(df)
parquet_file = '/tmp/employee_data.parquet'
pq.write_table(table, parquet_file)
```

Step 4: Upload Parquet file to S3

```
import boto3
s3 = boto3.client('s3')
bucket_name = 'employee-data-bucket-demo'
s3_key = 'uploads/employee_data_2025_10_27.parquet'
s3.upload_file(parquet_file, bucket_name, s3_key)
```

Step 5: S3 triggers Lambda on file upload - Configure S3 Event Notification to call `lambda-on-upload-trigger` whenever a new `.parquet` file is uploaded. Step 6: Triggered Lambda validates file & logs

```
import json

def lambda_handler(event, context):
    for record in event['Records']:
        s3_info = record['s3']
        bucket = s3_info['bucket']['name']
        key = s3_info['object']['key']
        print(f"INFO: Received file upload event for {key}")
    return {
        'statusCode': 200,
        'body': json.dumps('Processing completed successfully')
    }
```

1. Success Scenarios
2. MongoDB connection established successfully
3. Data fetched and Parquet file created correctly
4. File successfully uploaded to S3
5. Trigger Lambda executes automatically and logs event details
6. No manual intervention required

Expected CloudWatch Output:

```
INFO: Received file upload event for employee_data_2025_10_27.parquet
INFO: File size: 245 KB
INFO: Processing completed successfully.
```

1. Failure & Edge Cases Failure Scenario Root Cause Detection Mitigation / Handling MongoDB connection failure Wrong credentials / Network issue Lambda logs error 'Connection timeout' Validate connection string, add retry logic No data fetched Empty collection / wrong query Lambda logs 'No records found' Add conditional check and skip upload Parquet conversion failure Missing library / schema mismatch Lambda exception in conversion step Verify pyarrow/ pandas import and data types S3 upload failure IAM permission denied / bucket not found Error

'Access Denied' in CloudWatch Check IAM role policies for S3 PutObject Trigger not firing Misconfigured S3 event File upload but no logs in CloudWatch Verify S3 event notification configuration Lambda timeout Large dataset Timeout error Optimize query, use pagination or increase timeout

2. Assumptions & Dependencies

3. MongoDB Atlas instance is publicly accessible via connection string
4. AWS services (Lambda, S3, CloudWatch) are in the same region
5. Required Python libraries added as Lambda layers or inline

6. Validation Plan Test Case Expected Behavior Validation Method Successful file upload Trigger Lambda logs event in CloudWatch Check CloudWatch logs Empty dataset No file uploaded Verify no new object in S3 Invalid MongoDB credentials Error logged in Lambda Check CloudWatch error logs IAM permission issue Upload fails Observe AccessDenied error Large dataset Lambda runs within timeout Confirm execution time in logs

7. Risks & Mitigations

8. Lambda dependency size limit exceeded → Use external layer
9. Data sensitivity → Use encrypted S3 bucket and secure credentials
10. Event trigger misconfiguration → Test end-to-end before demo

11. Next Steps

12. Integrate React UI for manual upload and visualization
13. Automate pipeline using AWS Step Functions
14. Add DynamoDB for metadata storage
15. Extend to multiple file formats (CSV, JSON)

16. Appendix

17. AWS Region: ap-south-1 (Mumbai)
18. S3 Bucket Example: employee-data-bucket-demo
19. Lambda Names: lambda-fetch-employee, lambda-on-upload-trigger