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 To 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:**

* Reading employee data from MongoDB Atlas via AWS Lambda
* Generating a Parquet file dynamically
* Uploading the file to Amazon S3
* Triggering a Lambda function automatically on file upload
* 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 **Conceptual Flow:** MongoDB Atlas → AWS Lambda (Python) → Parquet Conversion → Amazon S3 (Upload) → Lambda Trigger → CloudWatch Logs

**Step-by-Step Flow:** 1. Lambda connects to MongoDB Atlas via connection string. 2. Fetches employee records (empId, name, dept, salary). 3. Converts data into Parquet file using pandas/pyarrow. 4. Uploads generated Parquet file to S3 bucket. 5. S3 event notification triggers another Lambda function. 6. Triggered Lambda validates file and logs details to CloudWatch.

1. Success Scenarios

* MongoDB connection established successfully
* Data fetched and Parquet file created correctly
* File successfully uploaded to S3
* Trigger Lambda executes automatically and logs event details
* 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 | 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 | Verify S3 event notification configuration| | Lambda timeout | Large dataset | Timeout error | Optimize query, use pagination or increase timeout |
2. Assumptions & Dependencies

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

1. 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 |
2. Risks & Mitigations

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

1. Next Steps

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

1. Appendix: Code Snippets **Lambda Fetch Employee Data:**

import pymongo, os, pandas as pd, pyarrow as pa, pyarrow.parquet as pq, boto3  
  
MONGO\_URI = os.environ['MONGO\_URI']  
client = pymongo.MongoClient(MONGO\_URI)  
db = client['employee\_db']  
collection = db['employees']  
  
records = list(collection.find({}, {'\_id': 0, 'empId': 1, 'name': 1, 'dept': 1, 'salary': 1}))  
  
# Convert to Parquet  
df = pd.DataFrame(records)  
table = pa.Table.from\_pandas(df)  
parquet\_file = '/tmp/employee\_data.parquet'  
pq.write\_table(table, parquet\_file)  
  
# Upload to S3  
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)

**Lambda Trigger on S3 Upload:**

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

1. Appendix: AWS Info

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