AWS-SPARK ON UNSTRUCTURED DATA STREAMING

01

Abstract

This project addresses the growing need to extract value from data that doesn't conform to traditional, structured formats. Unstructured data, such as text, images, audio, and video, is becoming increasingly prevalent, and organizations need effective ways to analyze it for insights.
The primary goal is to design and implement a data engineering pipeline that effectively processes unstructured data streams in real-time. This pipeline leverages the power of Apache Spark in conjunction with various Amazon Web Services (AWS) to address the unique challenges posed by unstructured data.

Introduction

PROBLEM STATEMENT

Unlike structured data (e.g., relational databases) with predefined schemas, unstructured data lacks a rigid format. This inherent lack of structure makes it difficult to process, analyze, and extract meaningful insights using traditional data processing techniques.

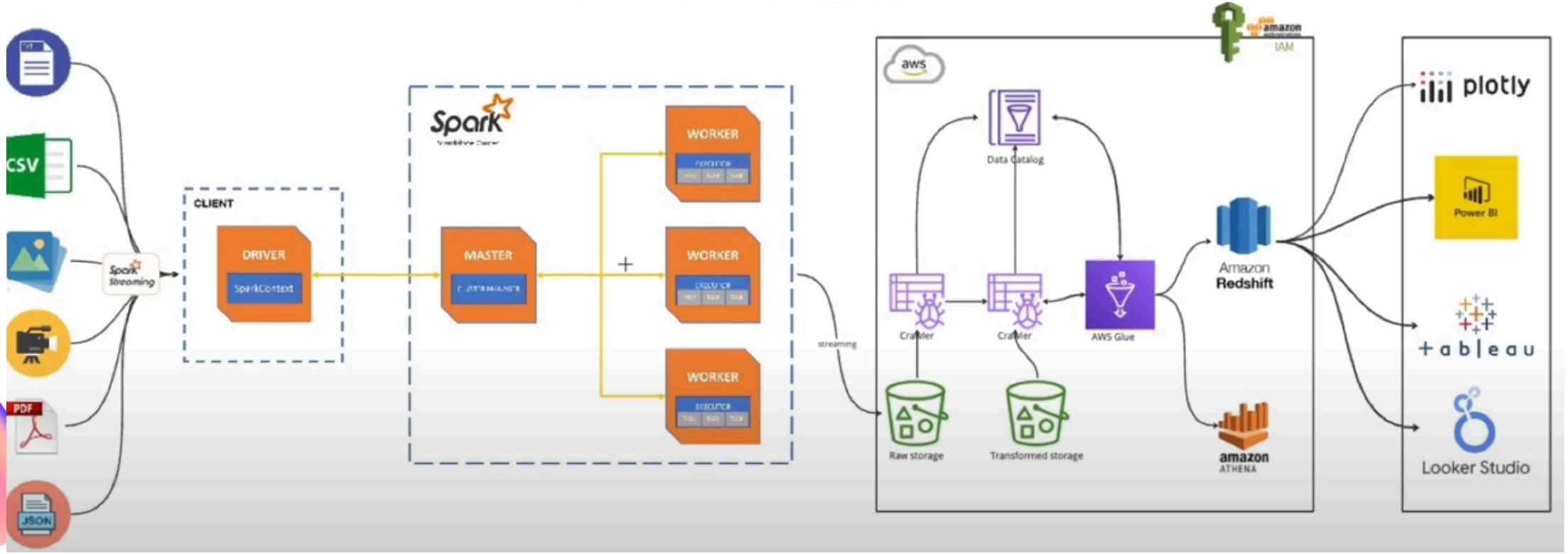
OBJECTIVE

To build a real-time data streaming pipeline that can ingest, process, and analyze unstructured data from various sources.

GOAL

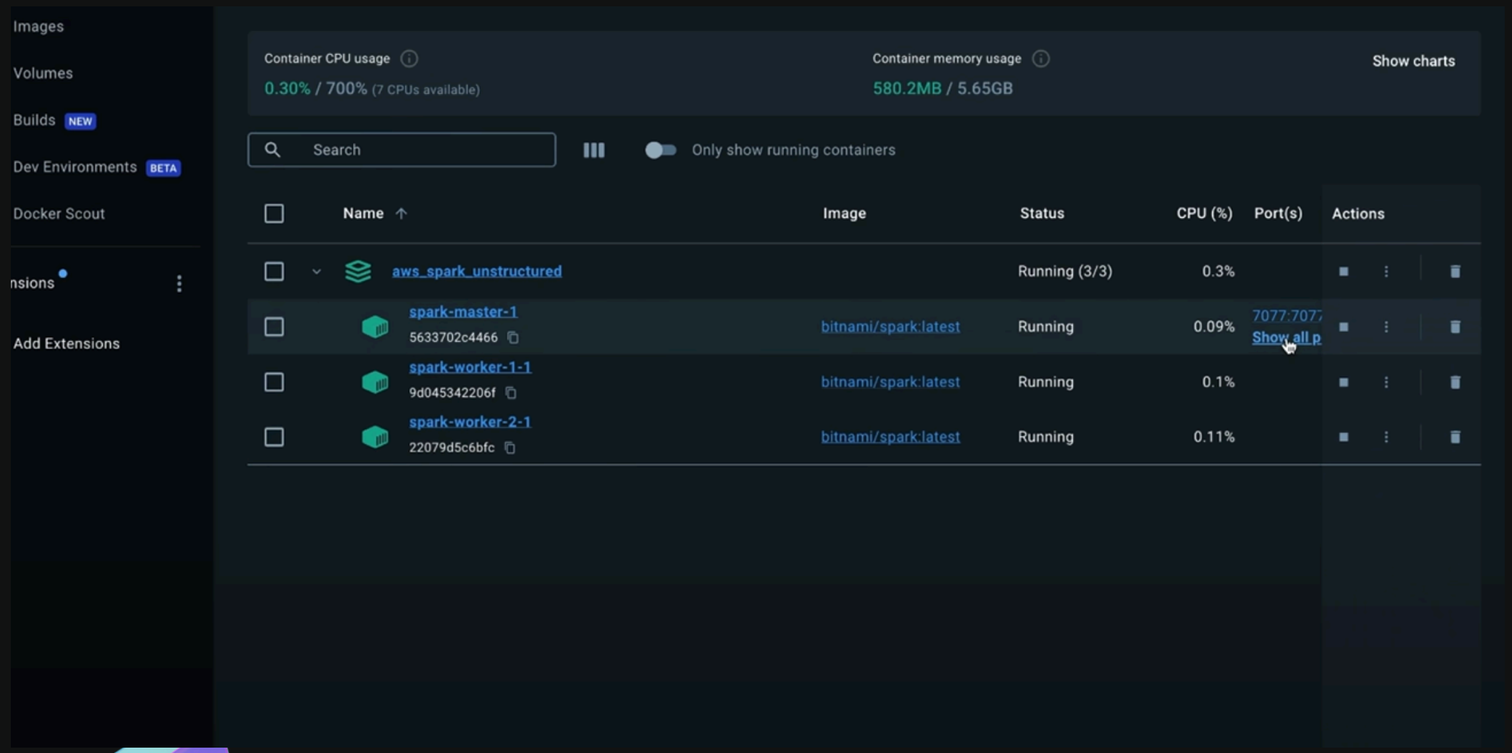
Specify the key functionalities and outcomes expected from the pipeline. Develop a system capable of handling various data types, extracting meaningful information, storing processed data in S3, cataloging data with Glue, and querying with Athena.

04 ARCHITECTURE



System Design

Development Environment:
Specifies the software and tools used for developing the project, including PyCharm,Python version 3.9.6, Docker



config.py udf_utils.py docker-compose.yml

```yaml
version: '3'

x-spark-common: &spark-common
  image: bitnami/spark:latest
  volumes:
    - ./jobs:/opt/bitnami/spark/jobs
  command: bin/spark-class org.apache.spark.deploy.worker.Worker spark://spark-master:7077
  depends_on:
    - spark-master
  environment:
    SPARK_MODE: Worker
    SPARK_WORKER_CORES: 2
    SPARK_WORKER_MEMORY: 1g
    SPARK_MASTER_URL: spark://spark-master:7077
  networks:
    - datamasterylab

services:
  spark-master:
    image: bitnami/spark:latest
    volumes:
      - ./jobs:/opt/bitnami/spark/jobs
    command: bin/spark-class org.apache.spark.deploy.master.Master
    ports:
      - "9092:8080"
      - "7077:7077"
```

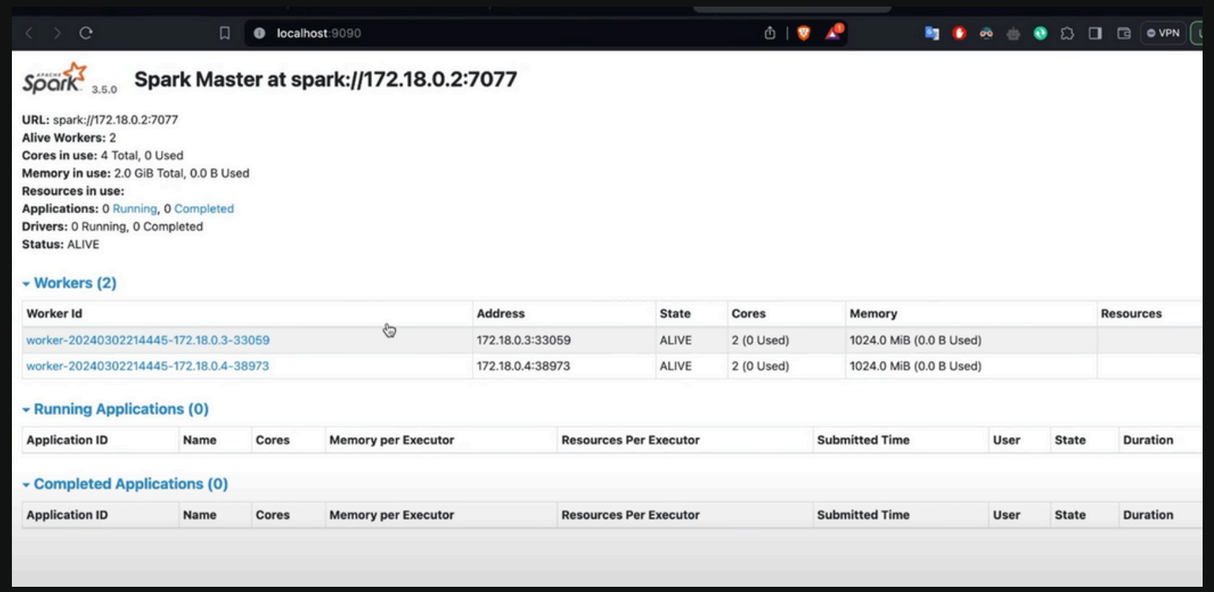
main.py config.py udf_utils.py docker-compose.yml

```yaml
x-spark-common: &spark-common
    - datamasterylab

services:
  spark-master:
    image: bitnami/spark:latest
    volumes:
      - ./jobs:/opt/bitnami/spark/jobs
    command: bin/spark-class org.apache.spark.deploy.master.Master
    ports:
      - "9092:8080"
      - "7077:7077"
    networks:
      - datamasterylab

  spark-worker-1:
    <<: *spark-common
  spark-worker-2:
    <<: *spark-common
  spark-worker-3:
    <<: *spark-common
  spark-worker-4:
    <<: *spark-common

networks:
  datamasterylab:
```

Implementation

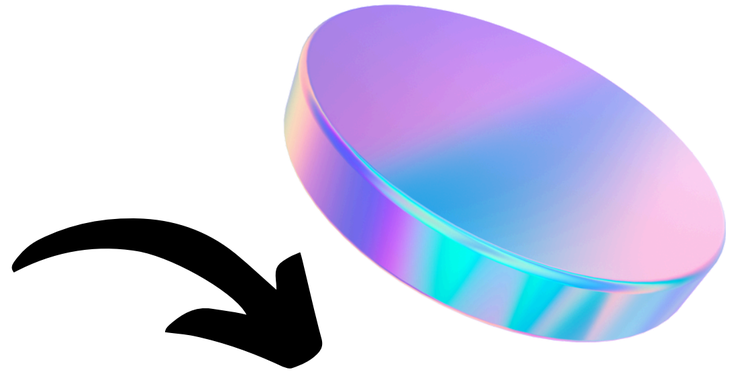
1. Setting up Spark Session:

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1742143758386).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.5
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 21.0.5)
Type in expressions to have them evaluated.
Type :help for more information.
```



IMPLEMENTATION

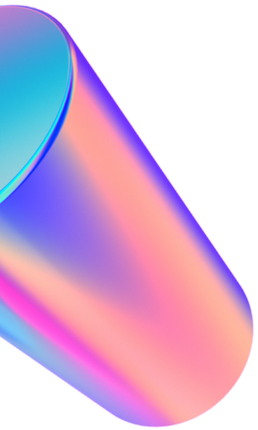
2. Create main.py file:



main.py × config.py udf_utils.py docker-compose.yml

```python
from pyspark.sql import SparkSession, DataFrame
from pyspark.sql.functions import udf, regexp_replace, explode
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, DateType, ArrayType

from config.config import configuration
from udf_utils import (
    split_job_postings, extract_file_name, extract_position, extract_classcode,
    extract_start_date, extract_end_date, extract_req, extract_notes,
    extract_duties, extract_selection, extract_experience_length,
    extract_job_type, extract_education_length, extract_school_type,
    extract_application_location, extract_salary_start, extract_salary_end
)
```

```python
def define_udfs():  1 usage
    return {
        'split_jobs_udf': udf(split_job_postings, ArrayType(StringType())),
        'extract_file_name_udf': udf(extract_file_name, StringType()),
        'extract_position_udf': udf(extract_position, StringType()),
        'extract_classcode_udf': udf(extract_classcode, StringType()),
        'extract_salary_start_udf': udf(extract_salary_start, DoubleType()),
        'extract_salary_end_udf': udf(extract_salary_end, DoubleType()),
        'extract_start_date_udf': udf(extract_start_date, DateType()),
        'extract_end_date_udf': udf(extract_end_date, DateType()),
        'extract_req_udf': udf(extract_req, StringType()),
        'extract_notes_udf': udf(extract_notes, StringType()),
        'extract_duties_udf': udf(extract_duties, StringType()),
        'extract_selection_udf': udf(extract_selection, StringType()),
        'extract_experience_length_udf': udf(extract_experience_length, StringType()),
        'extract_job_type_udf': udf(extract_job_type, StringType()),
        'extract_education_length_udf': udf(extract_education_length, StringType()),
        'extract_school_type_udf': udf(extract_school_type, StringType()),
        'extract_application_location_udf': udf(extract_application_location, StringType()),
    }
```



```python
    }

if __name__ == "__main__":
    spark = (SparkSession.builder.appName('JobPostingProcessor')
             .config('spark.jars.packages',
                     'org.apache.hadoop:hadoop-aws:3.3.1,com.amazonaws:aws-java-sdk:1.11.469')
             .config('spark.hadoop.fs.s3a.impl', 'org.apache.hadoop.fs.s3a.S3AFileSystem')
             .config("spark.hadoop.fs.s3a.access.key", "AKIAWCYX7WR77EMX2B5N")
             .config("spark.hadoop.fs.s3a.secret.key", "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")
             .config("spark.hadoop.fs.s3a.endpoint", "s3.amazonaws.com")
             .config('spark.hadoop.fs.s3a.aws.credentials.provider',
                     'org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')
             .getOrCreate())

    text_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_text2'
    csv_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_csv'
    image_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_image'
    json_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_json2'
    pdf_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_pdf'
    video_input_dir = 'file:///C:/Users/sahil/PycharmProjects/PythonProject2/input/input_video'
```

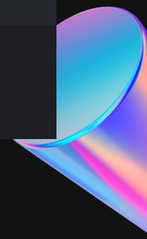
IMPLEMENTATION

3. Setting up udf_utils.py:

```
data_schema = StructType([
    StructField( name: 'file_name', StringType(), nullable: True),
    StructField( name: 'position', StringType(), nullable: True),
    StructField( name: 'classcode', StringType(), nullable: True),
    StructField( name: 'salary_start', DoubleType(), nullable: True),
    StructField( name: 'salary_end', DoubleType(), nullable: True),
    StructField( name: 'start_date', DateType(), nullable: True),
    StructField( name: 'end_date', DateType(), nullable: True),
    StructField( name: 'req', StringType(), nullable: True),
    StructField( name: 'notes', StringType(), nullable: True),
    StructField( name: 'duties', StringType(), nullable: True),
    StructField( name: 'selection', StringType(), nullable: True),
    StructField( name: 'experience_length', StringType(), nullable: True),
    StructField( name: 'job_type', StringType(), nullable: True),
    StructField( name: 'education_length', StringType(), nullable: True),
    StructField( name: 'school_type', StringType(), nullable: True),
    StructField( name: 'application_location', StringType(), nullable: True),
])
```

.py | config.py | udf_utils.py × | docker-compose.yml

```
import re
from datetime import datetime
from pyspark.sql.types import ArrayType, StringType

def split_job_postings(file_content):  2 usages
    """Splits multi-job file into individual postings"""
    try:
        return re.split( pattern: r'\n(?=Job Posting: job\d+\.txt)', file_content)[1:]
    except Exception as e:
        raise ValueError(f"Error splitting jobs: {str(e)}")

def extract_file_name(file_content):  2 usages
    """Gets main category from first line"""
    return file_content.split('\n')[0].strip()

def extract_position(job_text):  2 usages
    """Extracts all positions from job text"""
    try:
        matches = re.findall( pattern: r'Position:\s*(.+?)\n', job_text)
        return ', '.join(matches) if matches else None
    except Exception as e:
        raise ValueError(f"Position error: {str(e)}")

def extract_classcode(job_text):  2 usages
    """Extracts class codes with flexible formatting"""
    try:
```



Implementation

3. Create udf_utils.py file:

in.py | config.py | udf_utils.py | docker-compose.yml

```
        raise ValueError(f"Position error: {str(e)}")

def extract_classcode(job_text):  2 usages
    """Extracts class codes with flexible formatting"""
    try:
        match = re.search( pattern: r'classcode:\s*([^\s]+)', job_text, re.IGNORECASE)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"classcode error: {str(e)}")

def extract_salary_start(job_text):  2 usages
    """Handles various salary formats"""
    try:
        match = re.search( pattern: r'Salary starts at[^\d]*([\d,]+)', job_text)
        return float(match.group(1).replace( __old: ',', __new: '')) if match else None
    except Exception as e:
        raise ValueError(f"Salary start error: {str(e)}")

def extract_salary_end(job_text):  2 usages
    """Handles different end salary formats"""
    try:
        match = re.search( pattern: r'ends at[^\d]*([\d,]+)', job_text)
        return float(match.group(1).replace( __old: ',', __new: '')) if match else None
    except Exception as e:
        raise ValueError(f"Salary end error: {str(e)}")
```

n.py | config.py | udf_utils.py | docker-compose.yml

```
def extract_start_date(job_text):  2 usages
    """Flexible date parsing"""
    try:
        match = re.search( pattern: r'from\s*(\d{4}-\d{1,2}-\d{1,2})', job_text)
        return datetime.strptime(match.group(1), format: '%Y-%m-%d') if match else None
    except Exception as e:
        raise ValueError(f"Start date error: {str(e)}")

def extract_end_date(job_text):  2 usages
    """End date with validation"""
    try:
        match = re.search( pattern: r'until\s*(\d{4}-\d{1,2}-\d{1,2})', job_text)
        return datetime.strptime(match.group(1), format: '%Y-%m-%d') if match else None
    except Exception as e:
        raise ValueError(f"End date error: {str(e)}")

# Updated regex patterns for other fields
def extract_req(job_text):  2 usages
    try:
        match = re.search( pattern: r'Required skills:\s*(.+?)(\.|\n|$)', job_text, re.DOTALL)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Req error: {str(e)}")
```

IMPLEMENTATION

main.py | config.py | udf_utils.py | docker-compose.yml

```python

def extract_notes(job_text):  2 usages
    try:
        match = re.search( pattern: r'Additional notes:\s*(.+?)(\.|\n|$)', job_text, re.DOTALL)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Notes error: {str(e)}")

def extract_duties(job_text):  2 usages
    try:
        match = re.search( pattern: r'Primary duties:\s*(.+?)(\.|\n|$)', job_text, re.DOTALL)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Duties error: {str(e)}")

def extract_selection(job_text):  2 usages
    try:
        match = re.search( pattern: r'Selection process:\s*(.+?)(\.|\n|$)', job_text, re.DOTALL)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Selection error: {str(e)}")

def extract_experience_length(job_text):  2 usages
    try:
        match = re.search( pattern: r'Experience required:\s*(\d+)[+]?\s*years', job_text)
```

in.py | config.py | udf_utils.py | docker-compose.yml

```python
def extract_job_type(job_text):  2 usages
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Job type error: {str(e)}")

def extract_education_length(job_text):  2 usages
    try:
        match = re.search( pattern: r'Educational requirement:\s*(\d+)\s*years', job_text)
        return int(match.group(1)) if match else None
    except Exception as e:
        raise ValueError(f"Edu length error: {str(e)}")

def extract_school_type(job_text):  2 usages
    try:
        match = re.search( pattern: r'Preferred school type:\s*(.+?)(\.|\n|$)', job_text)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"School type error: {str(e)}")

def extract_application_location(job_text):  2 usages
    try:
        match = re.search( pattern: r'Application location:\s*(.+?)(\.|\n|$)', job_text)
        return match.group(1).strip() if match else None
    except Exception as e:
        raise ValueError(f"Location error: {str(e)}")
```

IMPLEMENTATION

4. Checking Data Stream:

config.py udf_utils.py docker-compose.yml

```python
udfs = define_udfs()

# Pipeline
raw_df = (spark.readStream
          .format('text')
          .option( key: 'wholetext',  value: 'true')
          .load(text_input_dir))

json_df = spark.readStream.format("json").schema(data_schema).option( key: "multiLine",  value: "true").load(json_input_dir)

processed_df = (raw_df
    .withColumn( colName: 'file_name', regexp_replace(udfs['extract_file_name_udf']('value'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'job_text', explode(udfs['split_jobs_udf']('value')))
    .withColumn( colName: 'position', regexp_replace(udfs['extract_position_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'classcode', regexp_replace(udfs['extract_classcode_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'salary_start', udfs['extract_salary_start_udf']('job_text'))
    .withColumn( colName: 'salary_end', udfs['extract_salary_end_udf']('job_text'))
    .withColumn( colName: 'start_date', udfs['extract_start_date_udf']('job_text'))
    .withColumn( colName: 'end_date', udfs['extract_end_date_udf']('job_text'))
    .withColumn( colName: 'req', regexp_replace(udfs['extract_req_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'notes', regexp_replace(udfs['extract_notes_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'duties', regexp_replace(udfs['extract_duties_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'selection', regexp_replace(udfs['extract_selection_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'experience_length', udfs['extract_experience_length_udf']('job_text'))
    .withColumn( colName: 'job_type', regexp_replace(udfs['extract_job_type_udf']('job_text'),  pattern: '\r',  replacement: ''))
    .withColumn( colName: 'education_length', udfs['extract_education_length_udf']('job_text'))
```

```
AWS_Spark_Unstructured
.venv
config
config.py
input
input_csv
input_img
input_json
input_pdf
input_text
```

```python
70  job_bulletins_df = (spark.readStream
71                      .format('text')
72                      .option( key: 'wholetext',  value: 'true')
73                      .load(text_input_dir)
74                      )
75
76  query = (job_bulletins_df
```

```
if __name__ == "__main__"
```

Terminal Local

```
Batch: 0
-------------------------------------------
24/03/02 17:57:56 INFO CodeGenerator: Code generated in 5.843791 ms
24/03/02 17:57:56 INFO BlockManagerInfo: Removed broadcast_1_piece0 on 192.168.0.168:58521 in memory (size: 4.9 KiB, free: 434
.4 MiB)
24/03/02 17:57:56 INFO CodeGenerator: Code generated in 6.857375 ms
+--------------------+
|               value|
+--------------------+
|ACCOUNTING CLERK\...|
+--------------------+

24/03/02 17:57:56 INFO WriteToDataSourceV2Exec: Data source write support MicroBatchWrite[epoch: 0, writer: ConsoleWriter[numR
ows=20, truncate=true]] committed.
24/03/02 17:57:56 INFO CheckpointFileManager: Writing atomically to file:/private/var/folders/yn/ybctpq1507j7b32qmsyq2tkh0000g
```

5. Combining Structured and Unstructured data together and processing it:

```
processed_df = processed_df.select('file_name', 'position', 'classcode', 'salary_start', 'salary_end',
                'start_date', 'end_date', 'req', 'notes', 'duties', 'selection',
                  'experience_length', 'job_type', 'education_length', 'school_type',
                  'application_location')


json_df = json_df.select('file_name', 'position', 'classcode', 'salary_start', 'salary_end',
                'start_date', 'end_date', 'req', 'notes', 'duties', 'selection',
                  'experience_length', 'job_type', 'education_length', 'school_type',
                  'application_location')

union_dataframe = processed_df.union(json_df)

def streamWriter(input: DataFrame, checkpointFolder, output):  1 usage
    return(input.writeStream.
           format('parquet')
           .option( key: 'checkpointLocation', checkpointFolder)
           .option( key: 'path', output)
           .outputMode('append')
           .trigger(processingTime='5 seconds')
           .start()
           )
```

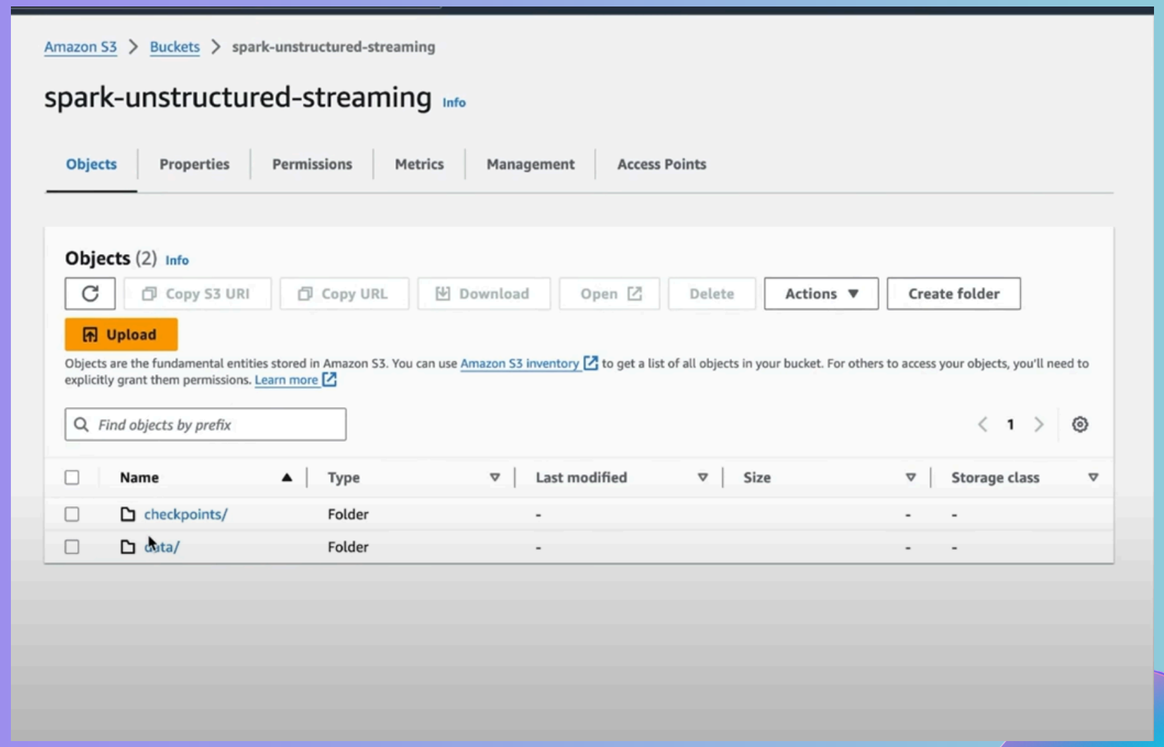
config.py | udf_utils.py | docker-compose.yml

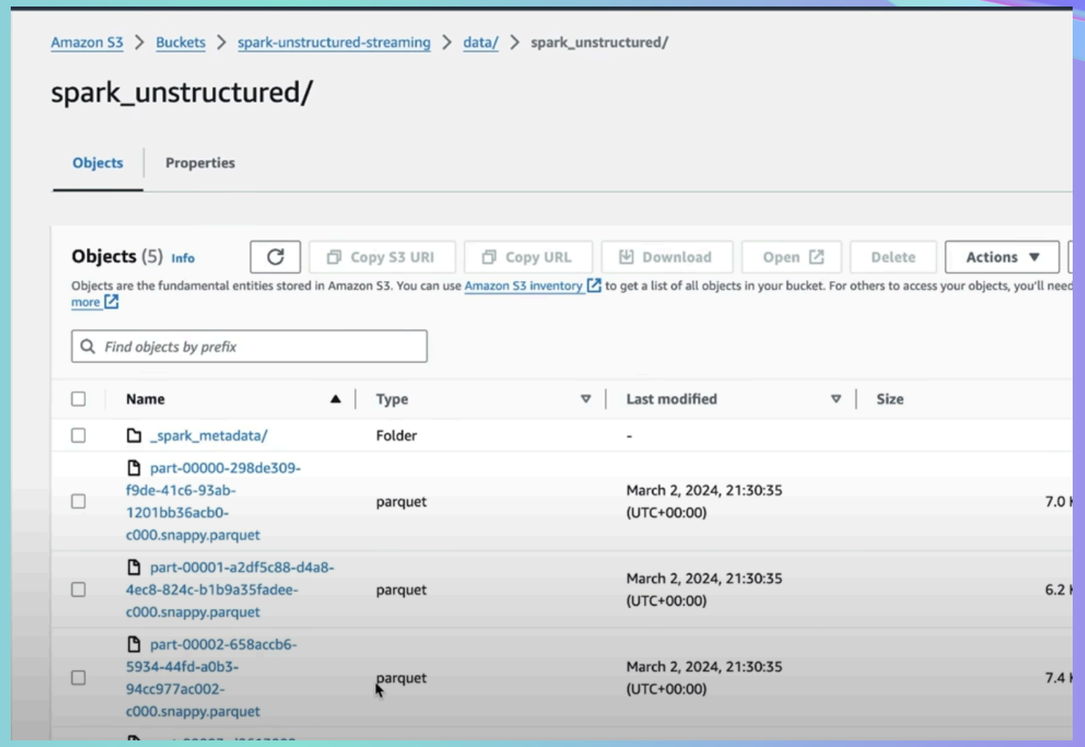
```
def streamWriter(input: DataFrame, checkpointFolder, output):  1 usage
           format('parquet')
           .option( key: 'checkpointLocation', checkpointFolder)
           .option( key: 'path', output)
           .outputMode('append')
           .trigger(processingTime='5 seconds')
           .start()
           )

# query = (union_dataframe.writeStream
#          .outputMode('append')
#          .format('console')
#          .option('truncate', False)
#          .start())

query = streamWriter(union_dataframe, checkpointFolder='s3a://aws-spark-unstructured-project/checkpoints/',
                     output='s3a://aws-spark-unstructured-project/data/spark-unstructured')

query.awaitTermination()

spark.stop()
```

```
Terminal  Local  Local (2)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 548.0 in stage 0.0 (TID 548) (172.18.0.3, executor 0, partition 548, PROCESS_LOCAL, 44023 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 543.0 in stage 0.0 (TID 543) in 115 ms on 172.18.0.3 (executor 0) (545/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 549.0 in stage 0.0 (TID 549) (172.18.0.4, executor 1, partition 549, PROCESS_LOCAL, 44017 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 545.0 in stage 0.0 (TID 545) in 94 ms on 172.18.0.4 (executor 1) (546/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 550.0 in stage 0.0 (TID 550) (172.18.0.4, executor 1, partition 550, PROCESS_LOCAL, 44034 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 546.0 in stage 0.0 (TID 546) in 96 ms on 172.18.0.4 (executor 1) (547/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 551.0 in stage 0.0 (TID 551) (172.18.0.3, executor 0, partition 551, PROCESS_LOCAL, 44056 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 547.0 in stage 0.0 (TID 547) in 85 ms on 172.18.0.3 (executor 0) (548/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 552.0 in stage 0.0 (TID 552) (172.18.0.3, executor 0, partition 552, PROCESS_LOCAL, 44021 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 548.0 in stage 0.0 (TID 548) in 78 ms on 172.18.0.3 (executor 0) (549/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 553.0 in stage 0.0 (TID 553) (172.18.0.4, executor 1, partition 553, PROCESS_LOCAL, 44015 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 554.0 in stage 0.0 (TID 554) (172.18.0.4, executor 1, partition 554, PROCESS_LOCAL, 44017 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 549.0 in stage 0.0 (TID 549) in 86 ms on 172.18.0.4 (executor 1) (550/650)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 550.0 in stage 0.0 (TID 550) in 69 ms on 172.18.0.4 (executor 1) (551/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 555.0 in stage 0.0 (TID 555) (172.18.0.3, executor 0, partition 555, PROCESS_LOCAL, 43997 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 556.0 in stage 0.0 (TID 556) (172.18.0.3, executor 0, partition 556, PROCESS_LOCAL, 44015 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 552.0 in stage 0.0 (TID 552) in 51 ms on 172.18.0.3 (executor 0) (552/650)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 551.0 in stage 0.0 (TID 551) in 68 ms on 172.18.0.3 (executor 0) (553/650)
24/03/02 22:01:12 INFO TaskSetManager: Starting task 557.0 in stage 0.0 (TID 557) (172.18.0.4, executor 1, partition 557, PROCESS_LOCAL, 44002 bytes)
24/03/02 22:01:12 INFO TaskSetManager: Finished task 553.0 in stage 0.0 (TID 553) in 52 ms on 172.18.0.4 (executor 1) (554/650)
```

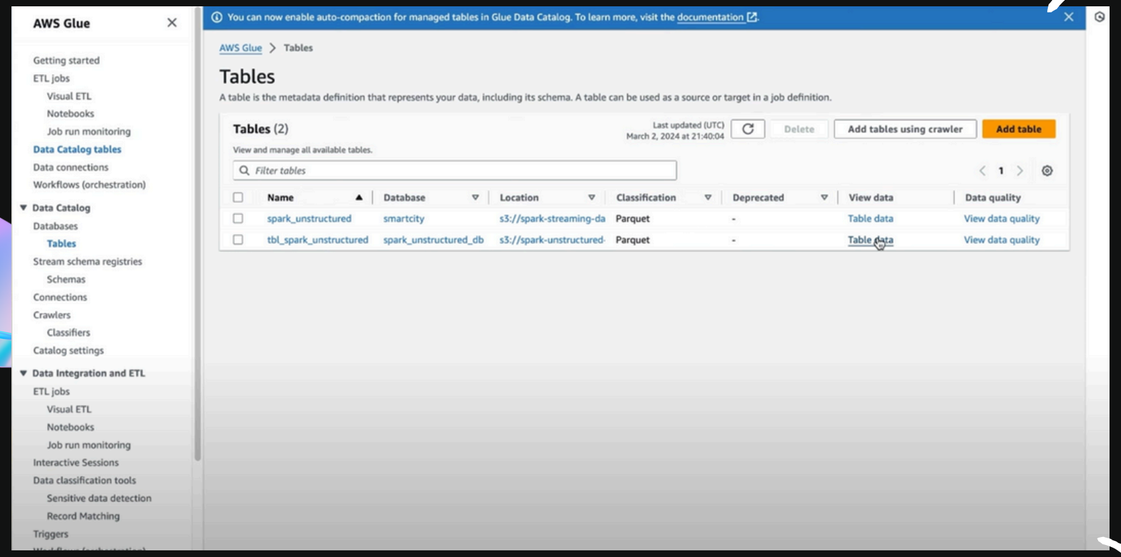
IMPLEMENTATION

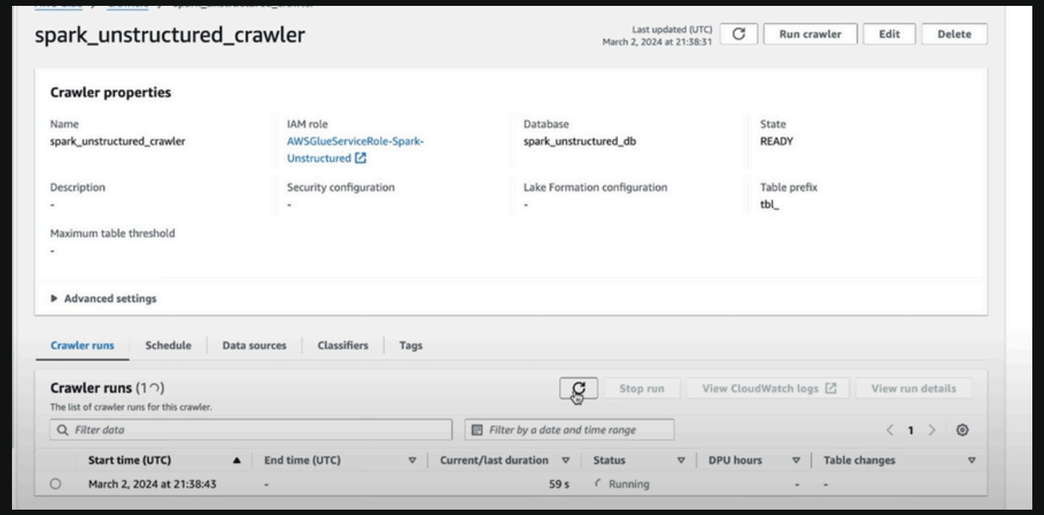
6. AWS INTEGRATION

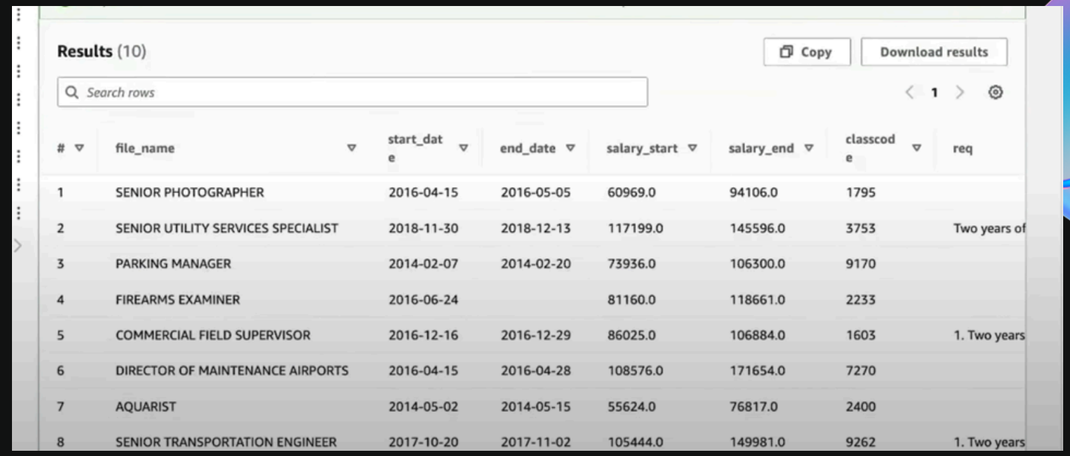




AWS INTEGRATION







Benefits

- Real-time Insights
- Improved Decision-Making
- Handles Variety of Data
- Unified Data Processing
- Faster Data Access
- Scalable Processing
- Reliable Storage
- Easy Querying
- Simplified Data Management
- Streamlined Connectivity
- Centralized Configuration
- Automated Data Extraction
- Real-Time Data Updates
- Scalable and Real-Time Procesin

Conclusion

This project—the AWS Spark Unstructured Data Streaming project—demonstrates an end-to-end solution for processing and analyzing unstructured data in real time using Apache Spark integrated with AWS. The system ingests data from diverse sources (text and JSON), applies custom UDFs to parse and extract critical fields from the unstructured input, and unifies the data streams for further processing. The processed output is then stored persistently (e.g., in AWS S3 as parquet files) for downstream analytics. Throughout the project, key challenges such as dependency management, environment configuration, and the handling of unstructured data formats were addressed. Overall, the project showcases how Spark Streaming, in combination with AWS, can efficiently handle complex, real-time data pipelines in an unstructured data environment.

THANK YOU

SAHIL PATIL
2381991