# ETL Pipeline Prototype

## Overview

This document outlines the architecture and implementation of an ETL pipeline using AWS services.   
The pipeline consists of the following stages:  
- Extract: Retrieve data from APIs, files, or databases.  
- Transform: Clean, format, or aggregate data to meet analytical requirements.  
- Load: Store the transformed data in a cloud data warehouse (e.g., Amazon Redshift).

## Architecture Components

- AWS S3: Storage for raw and transformed data.  
- AWS Lambda: Event-driven computing for extraction and light transformations.  
- AWS Glue: Managed ETL service for complex data transformations and cataloging.  
- Amazon Redshift: Data warehouse for storing the final structured data.

## Pipeline Steps

### Step 1: Data Extraction

Goal: Extract raw data from a source (API, file, or database) and store it in S3.  
  
Implementation:  
- Use AWS Lambda to extract data and write it to S3.  
  
Example Script:

import json  
import requests  
import boto3  
from datetime import datetime  
  
s3\_client = boto3.client('s3')  
bucket\_name = "your-s3-bucket-name"  
  
def lambda\_handler(event, context):  
 api\_url = "https://api.example.com/data"  
 response = requests.get(api\_url)  
 data = response.json()  
 timestamp = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')  
 file\_name = f"raw\_data\_{timestamp}.json"  
   
 s3\_client.put\_object(  
 Bucket=bucket\_name,  
 Key=file\_name,  
 Body=json.dumps(data)  
 )  
 return {'statusCode': 200, 'body': 'Data extracted and stored successfully'}

Deployment: Schedule periodic triggers using Amazon CloudWatch Events (e.g., every 3 hours).

### Step 2: Data Transformation

Goal: Clean and process raw data to make it analysis-ready.  
  
Implementation:  
- Use AWS Glue for complex transformations and schema inference.  
  
Example Glue Script:

import boto3  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import current\_timestamp  
  
spark = SparkSession.builder.appName("GlueETL").getOrCreate()  
  
bucket\_name = "your-s3-bucket-name"  
input\_path = f"s3://{bucket\_name}/raw\_data/"  
output\_path = f"s3://{bucket\_name}/transformed\_data/"  
  
def glue\_job():  
 raw\_data\_df = spark.read.json(input\_path)  
 transformed\_df = raw\_data\_df.dropna().withColumn("processed\_timestamp", current\_timestamp())  
 transformed\_df.write.json(output\_path)  
 return {'statusCode': 200, 'body': 'Transformation complete'}

Output: Transformed data saved in S3 in a structured format (JSON, Parquet).

### Step 3: Data Loading

Goal: Load transformed data into Amazon Redshift.  
  
Implementation:  
- Use a Lambda function with psycopg2 to execute Redshift COPY commands.  
  
Example Script:

import psycopg2  
import boto3  
  
REDSHIFT\_HOST = "your-redshift-cluster.amazonaws.com"  
REDSHIFT\_TABLE = "your\_table"  
  
def lambda\_handler(event, context):  
 conn = psycopg2.connect(  
 dbname="your\_db",  
 user="your\_user",  
 password="your\_password",  
 host=REDSHIFT\_HOST,  
 port=5439  
 )  
 cursor = conn.cursor()  
 copy\_sql = f"""  
 COPY {REDSHIFT\_TABLE}  
 FROM 's3://your-s3-bucket-name/transformed\_data/'  
 IAM\_ROLE 'arn:aws:iam::your-account-id:role/your-redshift-role'  
 JSON 'auto';  
 """  
 cursor.execute(copy\_sql)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 return {'statusCode': 200, 'body': 'Data loaded into Redshift successfully'}

### Step 4: Automation and Orchestration

- Triggers: Automate Lambda and Glue using S3 events or CloudWatch schedules.  
- Orchestration Tools:  
 - AWS Step Functions: Manage workflows.  
 - Apache Airflow: For cross-cloud orchestration.

### Step 5: Monitoring and Error Handling

- Monitoring: Use CloudWatch Logs to capture logs from Lambda and Glue jobs.  
- Error Handling:  
 - Implement retries for transient failures.  
 - Configure AWS SNS notifications for errors.