Architecture Implementation Plan - NYC Taxi Data Analytics

Overview

Implementation plan for NYC Yellow Taxi data analytics pipeline on AWS using EMR Serverless, Glue, Athena, and Text-to-SQL interface.

Region: us-east-1

Step 1: Project Structure Setup

Create Directory Structure

```
nyc-taxi-analytics/
├── README.md
├── terraform/
│   ├── main.tf
│   ├── variables.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   ├── modules/
│   │   ├── s3/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   ├── iam/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   ├── glue/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   ├── emr-serverless/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   └── athena/
│   │       ├── main.tf
│   │       ├── variables.tf
│   │       └── outputs.tf
├── pyspark/
│   ├── jobs/
│   │   ├── data_validation_cleaning.py
│   │   ├── trip_metrics_aggregation.py
│   │   ├── geospatial_analysis.py
│   │   └── revenue_insights.py
│   ├── utils/
│   │   └── common_functions.py
│   └── requirements.txt
├── scripts/
│   ├── download_data.sh
│   ├── upload_to_s3.sh
│   └── run_emr_job.sh
├── text-to-sql/
│   ├── app.py
│   ├── requirements.txt
│   └── config.py
└── docs/
    ├── architecture.md
    └── data_dictionary.md
```

Initialize Git Repository

```bash
git init
git add .
git commit -m "Initial project structure"
```

Step 2: AWS Infrastructure - Terraform

2.1 S3 Module

File: `terraform/modules/s3/main.tf`

Resources:

- S3 bucket for data lake
- Folder structure: raw/, processed/, insights/, logs/emr-serverless/
- S3 bucket for Athena query results
- Bucket encryption (SSE-S3)
- Lifecycle policies

Outputs:

- Bucket name
- Bucket ARN

2.2 IAM Module

File: `terraform/modules/iam/main.tf`

Resources:

- EMR Serverless execution role
  - S3 read/write permissions (data bucket)
  - S3 write permissions (logs folder)
  - Glue Data Catalog permissions
  - CloudWatch Logs permissions (optional, for application-level logs)
- Glue Crawler role
  - S3 read permissions
  - Glue Data Catalog permissions
- Athena execution role (if needed)

Outputs:

- EMR execution role ARN
- Glue crawler role ARN

2.3 Glue Module

File: `terraform/modules/glue/main.tf`

Resources:

- Glue Database (e.g., `nyc_taxi_db`)
- Glue Crawler for raw/ folder
- Glue Crawler for processed/ folder
- Glue Crawler for insights/ folder

Outputs:

- Database name
- Raw crawler name

- Processed crawler name
- Insights crawler name

2.4 EMR Serverless Module

File: `terraform/modules/emr-serverless/main.tf`

Resources:

- EMR Serverless application
- Application name: `nyc-taxi-analytics`
- Runtime: EMR 6.15 or latest
- Pre-initialized capacity configuration (optional)

Outputs:

- Application ID
- Application ARN

2.5 Athena Module

File: `terraform/modules/athena/main.tf`

Resources:

- Athena workgroup
- Query result location in S3

Outputs:

- Workgroup name
- Query result location

2.6 Main Terraform Configuration

File: `terraform/main.tf`

- Call all modules
- Define dependencies between modules
- Configure AWS provider

File: `terraform/variables.tf`

- AWS region (default: us-east-1)
- Project name
- Environment
- Bucket names

File: `terraform/outputs.tf`

- All module outputs
- Resource ARNs and IDs for reference

Required outputs for scripts:

```hcl

```

```
output "s3_bucket_name" {
  description = "Name of the S3 data lake bucket"
  value       = module.s3.bucket_name
}

output "emr_application_id" {
  description = "EMR Serverless Application ID"
  value       = module.emr_serverless.application_id
}

output "emr_execution_role_arn" {
  description = "EMR Serverless Execution Role ARN"
  value       = module.iam.emr_execution_role_arn
}

output "glue_database_name" {
  description = "Glue Database Name"
  value       = module.glue.database_name
}

output "glue_raw_crawler_name" {
  description = "Glue Crawler for raw data"
  value       = module.glue.raw_crawler_name
}

output "glue_processed_crawler_name" {
  description = "Glue Crawler for processed data"
  value       = module.glue.processed_crawler_name
}

output "glue_insights_crawler_name" {
  description = "Glue Crawler for insights data"
  value       = module.glue.insights_crawler_name
}

output "athena_workgroup_name" {
  description = "Athena Workgroup Name"
  value       = module.athena.workgroup_name
}
```

Step 3: Deploy Infrastructure

3.1 Initialize Terraform

```bash
cd terraform
terraform init
```

3.2 Configure Variables

File: `terraform/terraform.tfvars`

```hcl
region       = "us-east-1"
project_name = "nyc-taxi-analytics"
environment  = "dev"
```

3.3 Plan and Apply

```bash
terraform plan
terraform apply
```

3.4 Save Outputs

```bash
terraform output > ../outputs.txt
```

Step 4: Data Ingestion

4.1 Download Script

File: `scripts/download_data.sh`

Functionality:

- Download Yellow Taxi Parquet files for 2025 (Jan-Oct)
- URLs: `https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2025-{01..10}.parquet`
- Store in local directory: `data/raw/`

- Verify downloads with checksums
- Retry failed downloads
- Log download status

Implementation:

bash

```bash
#!/bin/bash
set -e

BASE_URL="https://d37ci6vzurychx.cloudfront.net/trip-data"
LOCAL_DIR="data/raw"
YEAR="2025"

mkdir -p $LOCAL_DIR

for month in {01..10}; do
  FILE="yellow_tripdata_${YEAR}-${month}.parquet"
  URL="${BASE_URL}/${FILE}"

  echo "Downloading ${FILE}..."

  if curl -f -o "${LOCAL_DIR}/${FILE}" "${URL}"; then
    echo "✓ Downloaded ${FILE}"
  else
    echo "✗ Failed to download ${FILE}"
    exit 1
  fi
done

echo "All files downloaded successfully"
```

4.2 Upload Script

File: `scripts/upload_to_s3.sh`

Functionality:

- Upload downloaded files to S3 raw/ folder
- Implement partitioning: `year=YYYY/month=MM/`
- Use AWS CLI: `aws s3 cp`
- Verify uploads

Implementation:

```bash
#!/bin/bash
set -e

BUCKET_NAME=$(terraform -chdir=terraform output -raw s3_bucket_name)
LOCAL_DIR="data/raw"
YEAR="2025"

for month in {01..10}; do
  FILE="yellow_tripdata_${YEAR}-${month}.parquet"
  S3_PATH="s3://${BUCKET_NAME}/raw/year=${YEAR}/month=${month}/${FILE}"

  echo "Uploading ${FILE} to ${S3_PATH}..."

  if aws s3 cp "${LOCAL_DIR}/${FILE}" "${S3_PATH}"; then
    echo "✓ Uploaded ${FILE}"
  else
    echo "✗ Failed to upload ${FILE}"
    exit 1
  fi
done

echo "All files uploaded successfully"
```

S3 Bucket Structure:

```
s3://bucket-name/
├── raw/
│   └── year=2025/
│       ├── month=01/yellow_tripdata_2025-01.parquet
│       ├── month=02/yellow_tripdata_2025-02.parquet
│       └── ...
├── processed/
├── insights/
├── logs/emr-serverless/
└── code/pyspark/
```

4.3 Execute Data Ingestion

```bash
```

```
chmod +x scripts/download_data.sh
chmod +x scripts/upload_to_s3.sh

./scripts/download_data.sh
./scripts/upload_to_s3.sh
```

4.4 Validate Upload

```bash
# Verify file count
aws s3 ls s3://bucket-name/raw/year=2025/ --recursive | wc -l

# Check file sizes
aws s3 ls s3://bucket-name/raw/year=2025/ --recursive --human-readable

# Verify partitioning structure
aws s3 ls s3://bucket-name/raw/year=2025/ --recursive
```

Step 5: Run Glue Crawler

5.1 Execute Crawler

```bash
RAW_CRAWLER=$(terraform -chdir=terraform output -raw glue_raw_crawler_name)
aws glue start-crawler --name ${RAW_CRAWLER}
```

5.2 Verify Table Creation

```bash
DATABASE=$(terraform -chdir=terraform output -raw glue_database_name)
aws glue get-table --database-name ${DATABASE} --name raw
```

5.3 Test Athena Query

```sql
-- Use database name from terraform output
SELECT COUNT(*) FROM <database_name>.raw LIMIT 10;
```

Get database name: `terraform -chdir=terraform output -raw glue_database_name`

Step 6: PySpark Jobs Development

6.1 Common Functions

File: `pyspark/utils/common_functions.py`

Functions:

- Data quality checks
- Column validation
- Schema enforcement
- Date parsing utilities

6.2 Job 1: Data Validation and Cleaning

File: `pyspark/jobs/data_validation_cleaning.py`

Steps:

1. Read from S3 raw/
2. Remove null values in critical columns
3. Filter invalid trips (negative fares, zero distance)
4. Standardize column names
5. Add data quality flags
6. Write to S3 processed/

Output Location: `s3://bucket/processed/trips_cleaned/`

6.3 Job 2: Trip Metrics Aggregation

File: `pyspark/jobs/trip_metrics_aggregation.py`

Steps:

1. Read from S3 processed/trips_cleaned/
2. Aggregate by time period (hourly, daily, weekly)
3. Aggregate by day of week
4. Aggregate by taxi zone

5. Calculate average trip distance, duration, fare
6. Write to S3 insights/

Output Locations:

- `s3://bucket/insights/trip_volume_by_hour/`
- `s3://bucket/insights/trip_volume_by_day/`
- `s3://bucket/insights/trip_volume_by_zone/`

6.4 Job 3: Geospatial Analysis

File: `pyspark/jobs/geospatial_analysis.py`

Steps:

1. Read from S3 processed/trips_cleaned/
2. Join with taxi zone lookup table
3. Identify popular pickup zones
4. Identify popular dropoff zones
5. Analyze pickup-dropoff zone pairs
6. Calculate zone-level metrics
7. Write to S3 insights/

Output Locations:

- `s3://bucket/insights/popular_pickup_zones/`
- `s3://bucket/insights/popular_dropoff_zones/`
- `s3://bucket/insights/zone_pair_analysis/`

6.5 Job 4: Revenue Insights

File: `pyspark/jobs/revenue_insights.py`

Steps:

1. Read from S3 processed/trips_cleaned/
2. Aggregate revenue by payment type
3. Calculate average fare, tip, total amount
4. Analyze congestion fee impact

5. Revenue by time of day
6. Revenue by vendor
7. Write to S3 insights/

Output Locations:

- `s3://bucket/insights/revenue_by_payment_type/`
- `s3://bucket/insights/revenue_by_time/`
- `s3://bucket/insights/congestion_fee_analysis/`

Step 7: Deploy PySpark Jobs

7.1 Upload Jobs to S3

```bash
BUCKET=$(terraform -chdir=terraform output -raw s3_bucket_name)
aws s3 cp pyspark/jobs/ s3://${BUCKET}/code/pyspark/ --recursive
aws s3 cp pyspark/utils/ s3://${BUCKET}/code/pyspark/utils/ --recursive
```

7.2 Create Job Execution Script

File: `scripts/run_emr_job.sh`

Parameters:

- Job script name (passed as argument)
- Application ID (from Terraform output)
- Execution role ARN (from Terraform output)
- S3 bucket name (from Terraform output)
- Spark configurations

Implementation:

```bash
```

```bash
#!/bin/bash
set -e

if [ $# -eq 0 ]; then
  echo "Usage: ./run_emr_job.sh <job_script_name>"
  exit 1
fi

JOB_SCRIPT=$1
APP_ID=$(terraform -chdir=terraform output -raw emr_application_id)
ROLE_ARN=$(terraform -chdir=terraform output -raw emr_execution_role_arn)
BUCKET=$(terraform -chdir=terraform output -raw s3_bucket_name)

echo "Submitting job: ${JOB_SCRIPT}"
echo "Application ID: ${APP_ID}"

JOB_RUN_ID=$(aws emr-serverless start-job-run \
  --application-id ${APP_ID} \
  --execution-role-arn ${ROLE_ARN} \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://'${BUCKET}'/code/pyspark/'${JOB_SCRIPT}'",
      "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf spark.executor.memory=8g --conf spark.driver.cores=2 -
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://'${BUCKET}'/logs/emr-serverless/"
      }
    }
  }' \
  --query 'jobRunId' \
  --output text)

echo "Job submitted with ID: ${JOB_RUN_ID}"
echo "Monitor logs at: s3://${BUCKET}/logs/emr-serverless/applications/${APP_ID}/jobs/${JOB_RUN_ID}/"

# Optional: Wait for job completion
echo "Waiting for job to complete..."
aws emr-serverless get-job-run \
  --application-id ${APP_ID} \
  --job-run-id ${JOB_RUN_ID} \
  --query 'jobRun.state' \
  --output text
```

```
echo "Job completed"
```

Make executable:

```bash
chmod +x scripts/run_emr_job.sh
```

7.3 Execute Jobs Sequentially

```bash
# Job 1: Validation and Cleaning
./scripts/run_emr_job.sh data_validation_cleaning.py

# Job 2: Trip Metrics
./scripts/run_emr_job.sh trip_metrics_aggregation.py

# Job 3: Geospatial Analysis
./scripts/run_emr_job.sh geospatial_analysis.py

# Job 4: Revenue Insights
./scripts/run_emr_job.sh revenue_insights.py
```

Note: Each job execution writes logs to `s3://bucket-name/logs/emr-serverless/applications/<app-id>/jobs/<job-run-id>/`

Step 8: Catalog Processed Data

8.1 Run Crawlers

```bash
PROCESSED_CRAWLER=$(terraform -chdir=terraform output -raw glue_processed_crawler_name)
INSIGHTS_CRAWLER=$(terraform -chdir=terraform output -raw glue_insights_crawler_name)

aws glue start-crawler --name ${PROCESSED_CRAWLER}
aws glue start-crawler --name ${INSIGHTS_CRAWLER}
```

8.2 Verify Tables

```bash
```

```
DATABASE=$(terraform -chdir=terraform output -raw glue_database_name)
aws glue get-tables --database-name ${DATABASE}
```

Step 9: Athena Query Layer

9.1 Test Queries

Get database name: `DATABASE=$(terraform -chdir=terraform output -raw glue_database_name)`

Query 1: Trip Volume by Day

```sql
SELECT
    date_trunc('day', tpep_pickup_datetime) as trip_date,
    COUNT(*) as trip_count
FROM <database_name>.trips_cleaned
GROUP BY date_trunc('day', tpep_pickup_datetime)
ORDER BY trip_date;
```

Query 2: Average Fare by Payment Type

```sql
SELECT
    payment_type,
    AVG(fare_amount) as avg_fare,
    AVG(tip_amount) as avg_tip
FROM <database_name>.trips_cleaned
GROUP BY payment_type;
```

Query 3: Popular Zones

```sql
SELECT
    pulocationid,
    COUNT(*) as pickup_count
FROM <database_name>.trips_cleaned
GROUP BY pulocationid
ORDER BY pickup_count DESC
LIMIT 10;
```

Replace `<database_name>` with actual database name from terraform output.

9.2 Create Views (Optional)

Create Athena views for common queries to simplify Text-to-SQL interface.

Step 10: Text-to-SQL Interface

10.1 Research and Select Solution

Evaluate:

- Vanna AI
- WrenAI
- Other open-source options

Selection Criteria:

- Ease of integration with Athena
- Python SDK availability
- RAG/training capabilities
- Documentation quality

10.2 Install Dependencies

File: `text-to-sql/requirements.txt`

```
vanna
boto3
streamlit
pandas
```

```bash
cd text-to-sql
pip install -r requirements.txt
```

10.3 Configure Connection

File: `text-to-sql/config.py`

- AWS credentials configuration

- Athena database name
- S3 query result location
- Glue Data Catalog configuration

10.4 Implement Application

File: `text-to-sql/app.py`

Features:

- Connect to Athena via Vanna AI
- Train on Glue Data Catalog schema
- Accept natural language queries
- Generate SQL
- Execute via Athena
- Display results
- Streamlit UI (optional)

Training Data:

- Table schemas from Glue Data Catalog
- Sample queries
- Business terminology mapping

10.5 Test Interface

Test Queries:

- "What was the total number of trips in January 2025?"
- "Show me average fare by payment type"
- "Which pickup zones are most popular?"
- "What's the impact of congestion fees on revenue?"

Step 11: Documentation

11.1 Architecture Documentation

File: `docs/architecture.md`

- Architecture diagram
- Data flow description
- Component descriptions
- AWS services used
- Security configuration

11.2 Data Dictionary

File: `docs/data_dictionary.md`

- Yellow Taxi data schema
- Column descriptions
- Data types
- Sample values
- Business definitions

11.3 README

File: `README.md`

- Project overview
- Prerequisites
- Setup instructions
- Deployment steps
- Usage examples
- Cost analysis
- Known limitations

Step 12: Testing and Validation

12.1 Infrastructure Validation

- Verify all Terraform resources deployed
- Check IAM roles and permissions
- Confirm S3 bucket structure

- Validate Glue Data Catalog tables
- Verify EMR Serverless application status:

```bash
APP_ID=$(terraform -chdir=terraform output -raw emr_application_id)
aws emr-serverless get-application --application-id ${APP_ID}
```

12.2 Data Pipeline Validation

- Verify data in raw/ folder
- Check processed data quality
- Validate insights aggregations
- Confirm row counts and metrics
- Review EMR Serverless job logs in S3

12.3 Query Layer Validation

- Test Athena queries on all tables
- Verify query performance
- Check query costs
- Validate results accuracy

12.4 Text-to-SQL Validation

- Test natural language queries
- Verify SQL generation accuracy
- Check result correctness
- Test error handling

Step 13: Cost Optimization

13.1 S3 Storage

- Implement lifecycle policies
- Archive old data to Glacier (if applicable)
- Clean up intermediate files

- Set retention policy for EMR logs (e.g., 30-90 days)

13.2 EMR Serverless

- Review job execution logs
- Optimize Spark configurations
- Adjust executor/driver resources

13.3 Athena

- Partition tables by date
- Use columnar formats (already Parquet)
- Limit query scans with WHERE clauses

Step 14: Deliverables

14.1 GitHub Repository

- All code and configuration
- Documentation
- Architecture diagram
- README with setup instructions

14.2 Working Deployment

- All AWS resources running
- Data pipeline executed successfully
- Text-to-SQL interface functional
- Sample queries and results

14.3 Technical Documentation

- Data pipeline design decisions
- Insights generated and business value
- Text-to-SQL solution evaluation (if multiple tested)
- Cost analysis

- Security implementation details

Notes

- All AWS resources in us-east-1
- Use default VPC and security groups
- Manual Terraform apply (no CI/CD)
- S3 server-side encryption (SSE-S3)
- IAM roles follow least privilege principle
- Glue crawlers run on-demand, not scheduled
- EMR Serverless jobs triggered manually via scripts
- EMR Serverless logs written to S3 (logs/emr-serverless/), not CloudWatch

Validation Checklist

Addressed from feedback:

- ✓ EMR Serverless job submission mechanism (Step 7.2)
- ✓ IAM permissions for job execution (Step 2.2)
- ✓ S3 monitoring configuration for EMR jobs (Step 7.2)
- ✓ Data validation after upload (Step 4.4)
- ✓ Partitioning strategy implementation (Step 4.2)
- ✓ Error handling in download/upload scripts (Steps 4.1, 4.2)
- ✓ Terraform outputs referenced in all scripts
- ✓ Proper Spark configurations (executor/driver cores and memory)
- ✓ Job monitoring and log locations