Introduction

This project focuses on migrating all the tables present in the on premise GroTrex database (Nis schema) to AWS S3 based DataLake after filtering the table rows in accordance to a criterion validated by the stakeholders. The solution has been intentionally kept very generic, repeatable and scalable in order to make sure that it can be leveraged across multiple environments without much hassle. The solution can be kicked off using a central driver script which by itself creates all the required resources, creates respective glue jobs from all the provided scripts, maps all the dependencies among ingestions, triggers all the ingestion jobs in order, makes all the tables available in Athena where they can be queried in place and logs and persists the operational metadata for each ingestion job run which can in turn be used for monitoring, performance and data quality tasks

Design and Implementation

In order for the solution to work across any environment, we need to ensure the following:

1. An AWS glue service role named "*AWSGlueServiceRoleDefault*" should be created with the following permissions:

    a. AmazonS3FullAccess
    b. AWSGlueServiceNotebookRole
    c. AWSGlueServiceRole
    d. AWSGlueConsoleFullAccess
    e. AmazonRDSFullAccess
    f. An inline JSON policy given below

Java

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "iam:GetRole",
            "iam:PassRole"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:iam::*:role/*"
         ],
         "Condition":{
            "StringLike":{
               "iam:PassedToService":[
                  "glue.amazonaws.com"
               ]
            }
         }
      }
   ]
}
```

2. A glue connector with a static IP to connect to the GoTrex database securely. The prerequisite for creating a glue connector connecting to an on-premise database is to create or configure a VPC which includes the following:

    a. A private subnet for hosting the Glue Connector
    b. A public subnet that contains a **NAT Gateway with a static IP** attached to it which the Glue Connector uses to communicate with the internet
    c. Configuration of route tables to enable this connectivity

    Once the VPC is created or configured, the following steps can be taken to create the required glue connector:

    a. Select AWS Glue service, then select create a connection
    b. Enter the connection name and select connection type as JDBC
    c. Enter JDBC URL in the format : j*dbc:sqlserver://<host_name>;databaseName=<database_name>;* along with username and password of a database user
    d. Create a new security group with all allowed inbound and outbound traffic
    e. Select the VPC ,the private subnet and security group created above
    f. Select Create connection

Once the glue role and connector are created, the driver script named *driver_all_raw_ingestions* would take care of the rest. The driver script has been divided into the following five main parts:

1. Creation of required resources using Infrastructure as Code

   The following resources are created upon each run of the driver script in case these resources do not exist:
   a. A S3 bucket for the raw layer of datalake with the given name
   b. A Glue database to catalog all the metadata of on-premise database tables
   c. A Glue crawler which crawls all the on-premise database tables and catalogs the metadata in AWS Glue Data Catalog
   d. A Glue database to catalog all the metadata of ingested datalake tables and operational metadata

2. Creation of glue jobs from s3 based scripts

   For each of the table ingestion, a PySpark script has been developed and placed inside a folder in S3. In order to automate the glue job creation process across different environments these scripts would have to be placed inside a S3 folder. The driver script would expect a path to the folder containing all these the ingestion scripts so that it creates job against each one of the scripts if it does not already exist

   A typical ingestion script would have the following structure:

      a. Reads from the on-premise database via a utility script
      b. Applies a filtration logic on the read table
      c. Writes the filtered table data to S3 datalake via a utility script which in turn also catalogs the data in Glue database containing metadata of ingested datalake tables

   Below is a template used for all the ingestion scripts jobs.The comments added at each step makes the template self explanatory

```python
from utils import *

# Define the data sources and destinations
database = "col_anon"
dest_datalake = "data-extract-crisis"
dest_table = "nis_policies"
```

```
utils = Utils()

# Reading data tables from on premise databases
nis_policies = utils.read_from_onprem(database,"nis.policies")
nis_organisations = utils.read_from_onprem(database,"nis.organisations")

# Creating temp views from data tables
nis_policies.createOrReplaceTempView("nis_policies")
nis_organisations.createOrReplaceTempView("nis_organisations")

# Defining filtering query
filter_query = """
    SELECT pol.*
    FROM nis_policies as pol
    INNER JOIN nis_organisations as o1  on o1.id = pol.AgentId
    INNER JOIN nis_organisations as o2  on o2.id = o1.ParentId
    WHERE o2.id IN
    (40,100,187,189,190,192,194,197,199,201,212,214,217,219,
    221,222,236,237,240,242,245,407,612,620,702,712,838,855,
    884,889,953,967,974,979,986,989,995,1008,1014,1019,1047,
    1056,1076,1091,1098,1116,1133,1146,1185,1197,1217,1222,
    1230,1240,1291,1355,1383,1399,1451,1560,1653)
"""

# Filtering the data table using fitering rule
filtered_nis_policies = spark.sql(filter_query)

# Persisting the filtered data table to datalake
utils.write_to_datalake(filtered_nis_policies,dest_datalake,dest_table)

# Dropping temp views of data tables
spark.catalog.dropTempView("nis_policies")
spark.catalog.dropTempView("nis_organisations")
```

3. Mapping of dependencies to determine the order in which the ingestion should be triggered

The driver script would be expecting a path to a csv file containing all the table names with their respective layers (dependency metadata). The driver would then order all the glue ingestion jobs in accordance to the dependencies

4. Triggering of ingestion and collection of operational metadata

   This part of the driver script would trigger the glue ingestion jobs in order. All the jobs belonging to a particular layer would run asynchronously. In addition, all the run metadata for each one of the jobs would be captured.The following indicators would recorded for each run of a particular job:
   a. Job Name
   b. Job Arguments
   c. Job Status (SUCCESS, FAILURE, SUSPENDED)
   d. Job Start Time
   e. Job End Time

   Note : The operational metastore is currently placed in S3 in parquet format with the name : “operational_metadata” and is queryable via Athena. However, the code base has the flexibility to replace S3 based metastore with a RDS metastore by providing the required credentials

5. Generation of reconciliation report

   Finally, once all the ingestions are created, a report is generated in parquet format with the name : “recon_report” and is queryable via Athena. This report can be used for data quality assurance and data validation. It has the following indicators:
   a. On Premise Table Name
   b. On Premise Table Counts
   c. DataLake Table Name
   d. DataLake Table Counts

Below is a template used for the driver script responsible for population of the entire datalake .The comments added at each step makes the template self explanatory

```python
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

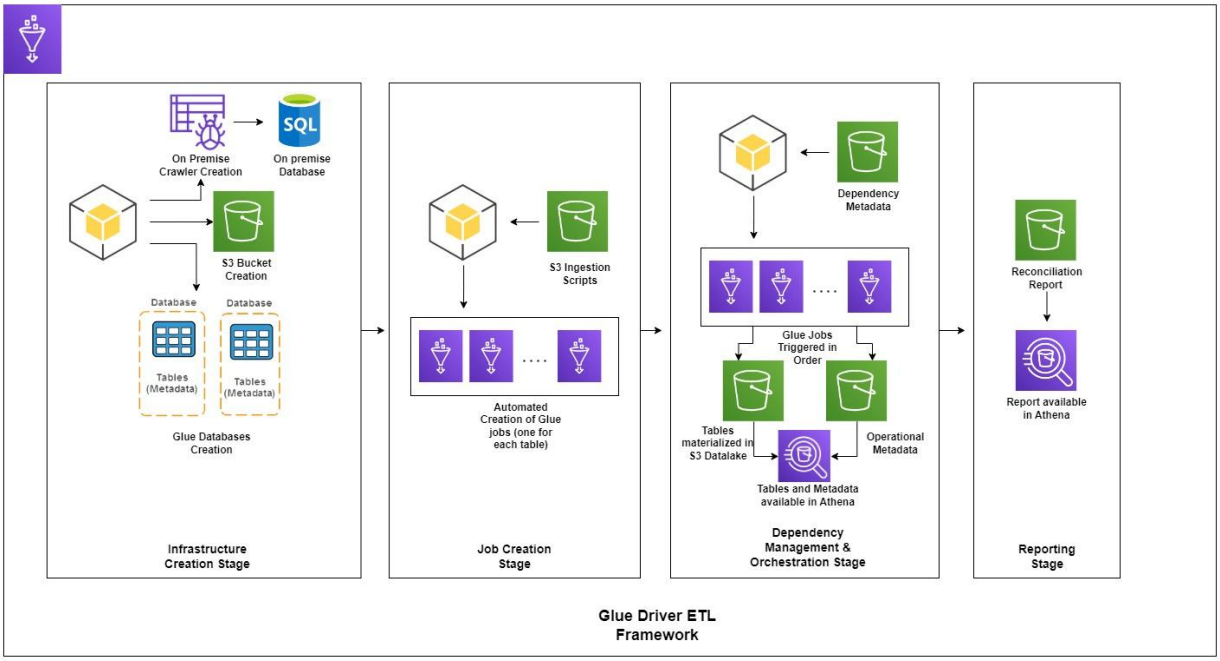
```
from awsglue.job import Job


sc = SparkContext()
glue_context = GlueContext(sc)
spark = glue_context.spark_session
job = Job(glue_context)


# Declaring ETL pipeline parameters

bucket_name = "c24-data-extract"
bucket_region = "eu-west-2"
onprem_database_name = "col_anon"
glue_database_name = "crisis-24"
jdbc_conn_name = "crisis-24-onprem-nis-conn"

metadata_loc = "s3://{}/metadata/dependencies_metadata.csv".format(bucket_name)
scripts_folder_loc = "s3://{}/scripts/jobs".format(bucket_name)
extra_py_files_loc = "s3://{}/scripts/utils.py".format(bucket_name)
extra_jars_loc = "s3://{}/jars/delta-core_2.12-1.0.0.jar".format(bucket_name)

max_batch_size = 25


# Importing automation and helper scripts

from infra_utils import *
from dependency_utils import *
from utils import *
from reporting_utils import *


# Creating infrastructure from code

infra_utils = SetUpInfraUtils()
infra_utils.create_infra(bucket_name,bucket_location,glue_database_name,jdbc_co
nn_name)

# Creation of glue jobs from s3 based scripts
infra_utils.create_all_glue_jobs_from_s3_scripts(bucket_name,scripts_folder_loc
,extra_py_files_loc,extra_jars_loc)

# Defining ETL dependencies (layers)
```

```
dependency_utils = DependencyUtils()
layered_jobs = dependency_utils.get_layered_jobs(metadata_loc)


# Executing ingestion jobs according to dependencies

run_utils = RunUtils()
base_utils = BaseUtils()

for layer in layered_jobs:
    batchs = base_utils.generate_batchs(layer,max_batch_size)
    for batch in batchs:
           run_utils.run_parallel(batch)

# Creating ingestion report

utils = Utils()
reporting_utils = ReportingUtils(utils)

reporting_utils.generate_recon_report(bucket_name,onprem_database_name)



job.commit()
```

The template above is compounded with four custom utility scripts that need to be kept in S3 and referenced in the driver script. They are the following:

1. utils.py (utilities that perform general ETL processing operations)
2. Infra_utils.py (utilities that perform Infrastructure as Code operations)
3. dependency_utils (utilities that maps all the dependencies among ingestions)
4. report_utils (utilities that generate the reconciliation report)

The following architecture diagram explains the overall design in detail pictorially



Migration Checklist

The following steps would be required to plug and play the entire ETL framework developed above on any AWS environment:

1. Create a glue service role with all the configurations mentioned above
2. Create a glue connector with all the configurations mentioned above
3. Place the dependencies metadata csv file in a S3 location
4. Place all the ingestion scripts in a S3 folder
5. Place all the four utility scripts in a S3 folder
6. Create a glue job with the driver script explained above and add all the four utility files to the glue job. Update the ETL pipeline parameters
7. Run the glue job