**Test Data Management (TDM)**

* It is a process of sourcing, managing and delivering high quality test data to the teams as and when required which is compliant, secure and meets users’ requirements.
* Test data is used by systems and applications developers/QA’s for rigorously testing applications against various systems test and scenarios.
* Today, app adoption depends on how delightfully a customer interacts with it and how seamlessly he experiences it. Most apps lose 95% of their audience within the first three months. And 67% of users hold application quality responsible for their churn. It becomes the onus of QA teams to ensure that the higher quality standards are met. TDM makes it possible.

**Current State/Challenges with Test Data**: Testing teams dependent on test data has to go though many challenges which slow down the software delivery and may hinder quality.

* **Slow, manual test environment provisioning**: Without TDM, software development teams depend on a manual data provisioning process that queues up tickets. Fulfilling each request can take a long time when there is no direct access to ready-made test data. This inadvertently stretches the testing cycle slowing down the pace of delivery.
* **Increased Storage Cost**: When organizations use full-size copies of production data for testing purposes, test data storage costs are high. The costs increase exponentially when the production environment grows, leading to high storage and license costs.
* **Data Availability**: In order to mimic the real time scenarios against application/software, QA team needs proper valid data and that’s the most challenging part. Using production data may lead to compliance and regulatory issues. We can create clusters of non-production data but that may cause increased costs on storage and expose the risk of leakage of user’s confidential data. Also, the un-availability of test data does not give any confidence to the developers or QAs on the user stories they are releasing.
* **Data Accessibility**: Some complex test scenarios require access to data which is available in different data sets. Accessing this type of provisioned data through multiple data sources takes time and a lot of approvals which delays the testing time.
* **Data Quality:** In many cases, the data may be available, but it fails to meet the required quality standards for the following reasons:
  + **Corrupted data**: Sharing and reusing test data between different testers often causes corruption problems. Relying on such corrupt data could have severe implications that may only be detected much later in the software delivery process.
  + **Irrelevant or incomplete data**: Applications must be tested against specific data that supports the required scenarios to be tested. A complete subset of relevant data, with referential integrity, is paramount in ensuring that test scenarios cover the required functionality to be tested.
  + **Unmasked data**: With data privacy a paramount concern, any production data used in testing environments must be adequately obscured by data masking tools (to make it unidentifiable). Flaws in this area may result in heavy fines and brand reputation damage in being non-compliant with regulations such as GDPR and CCPA.
* **Generic Data**: Application tested using generic data have highest chances of breaks in production since all the corners cases with production like data are not tested.

**Types of Data:** Any data that is marked with testing scope is termed as test data. Broadly, there are two types of test data

* + **Static Data** - This is the data that doesn't change after being recorded. Data points such as names and addresses are not considered dynamic. Most static data comprise non-sensitive information, but it can also include banking details and other credentials in some cases.
  + **Dynamic Data** - Test data that is constantly evolving and needs to be updated regularly is called dynamic data. Example: information such as banking and transaction history. Since dynamic data is the most sensitive, ensuring adequate storage and security is essential.

**Test Data Management Solutions to overcome challenges**:

* Based on the use cases and type of test, both static and dynamic data are used together. This data needs to be identified, sourced, and brought in from different databases while maintaining its integrity. It also needs to be securely stored and managed to avoid regulatory issues. This is where test data management helps.
* TDM is a collection of processes that ensures high-quality data is always available to the right requestor at the right time.
* TDM deals with planning, designing, sourcing, storing, securing, and retrieving test data promptly.
* With TDM, organizations can ensure that testers always have access to quality data in the required formats and the right quantities to fulfill their testing requirements.
* Since handling production data can be challenging due to its sensitivity and volume, many companies have also integrated data synthesis and obfuscation into their TDM practices.

**Why Test Data Management**: TDM introduces a structured approach to testing, enabling experts to test across all possible business scenarios. This eventually leads to better application quality and customer satisfaction.

**Aspects of Test Data Management:**

* **Analysis of test data requirements:** Before sourcing or generating data, you need to document the exact type and format of data required. Then perform an end-to-end analysis of the application, the platform, the software, and the test cases to zero in on the exact data requirements.
* **Obfuscation and masking of data**: Regulatory laws like GDPR require organizations to securely hide sensitive information from malicious actors. Obfuscation techniques such as masking and encryption are used to secure sensitive data that can potentially be misused. This includes banking details, medical history, and more.
* **Subsetting of data:** Testers slice the test data into usable subsets to reduce the amount of data that needs to be stored. This helps control storage and security costs during the data cloning process.
* **Generation of required data**: Handling production data can be an expensive affair. Most organizations now prefer generating representative or mock data that mimics the production values. The test data should get you as close to the original user environment as possible.
* **Delivery of data:** In complex test environments, including CI/CD, obfuscated and subsetted synthesized data must be delivered to the testing and QA teams in a reliable, fast, and seamless manner. Successful test data delivery guarantees timely and more accurate tests, resulting in better quality applications.
* **Storage and maintenance of data:** All the test data should be stored in a single repository for easy access and maintenance. Proper data refreshing is a significant part of its upkeep to keep redundancy and obsolescence at bay. Lastly, if the data is maintained correctly, it can be reused for more tests down the line, bringing down overall costs further.

**How to Build a Test Data Management Strategy:** The successful implementation of TDM in your organization can erase the gap between data requirements and the availability of data at hand. You must devise a clear strategy to ensure your testers can access the correct data on demand, which will form the groundwork on which you can build your TDM processes.

* **Define the system**: Testing teams should map all the data assets and connect them via information supply chains, and these scenarios should cover all possible situations. Since the data needs no defined parameters in TDM, only the system parameters must be determined. 1
* **Integrate all the data silos**: The required data might not be in one single repository in the beginning. When there are several different data silos, systems, and sources, teams need to integrate them using a management tool.
* **Refresh and sync test data**: The test data will be reused in many cases - when bugs are found and corrected, during regression testing, etc. For that to happen seamlessly, your system must be adaptable, easy to sync, and capable of refreshing data whenever needed.
* **Protect all the test data**: Data privacy and security are at the top of all data management concerns. No company wants to be caught in a regulatory impasse. Your strategy must account for all compliance and security requirements.
* **Synthesize data when required**: Often, using production values isn't viable. In this case, your system must be able to generate mock data based on real values while maintaining referential integrity across all systems.
* **Provision data to all the systems and environments**: This is the last part of your strategy. Once you generate, mask, and secure the data, you should move it to the test environments. Most organizations work with multiple platforms and systems - test data should be able to move from one environment to the next seamlessly.  6. Provision data to all the systems and environments

**Advantages of Test Data Management**

* **Better test coverage**: With TDM, you can test all the possible scenarios in testing at a faster rate. This provides unmatched visibility across the entire testing process with better defect detection.
* **Increased cost savings**: Testers can identify and correct defects much earlier in the cycle with better quality data on demand. Bug-fixing later in the cycle, especially after release, can cost up to 640x more. Additionally, a rigorous TDM process reduces the amount of test data stored, bringing down storage charges.
* **Improved customer satisfaction**: Your customers want a bug-free application that works as expected and meets all business requirements. Better test data coverage and the ability to find more bugs faster leads to a product that runs smoothly.
* **Faster time-to-market:** First-to-market is critical in deciding which companies win in today's business. Test automation and intelligent test data management can make testing much faster and more efficient, expediting your time-to-market.

**Test Data Management Tools Selection Criteria**

When evaluating test data management tools, make sure you choose the solution that supports:

* **Automation and self-service functionality:** Your test data management tool should allow DevOps and QA teams to eliminate manual processes when setting up new data testing environments, such as initializing target databases, configuration, and validation checks. With sufficient levels of automation in place, authorized testers can provision test data themselves, without having to rely on backlogged IT/DevOps.
* **Data integration**: With production data fragmented and dispersed across multiple enterprise systems, extracting complete and harmonized data for testing is a major challenge. Make sure your test data management tool can integrate with all relevant source and production systems, so it can extract complete test datasets.
* **On-demand data roll-back:** Your test data management tool should enable on-demand roll-back to previous datasets, without impacting the data currently involved with other tests.
* **Data masking:** Data masking is a non-negotiable requirement for any test data management tool. The solution you choose should include a wide variety of masking functions out-of-the-box while maintaining data integrity and adhering to organizational security and privacy policies. Masking PII in unstructured files and images should also be a requirement.
* **Synthetic data creation:** Testing teams can’t always extract a sufficient volume of test data from production datasets. A test data management tool that is capable of synthesizing supplemental data based on real production data is invaluable to DevOps.
* **High-speed provisioning**: Your test data management tool should provide a fast and frictionless path, from multiple source systems to multiple testing environments. Individual testers should be able to upload, adjust, and delete test sets manually, or automatically using CI/CD integration.

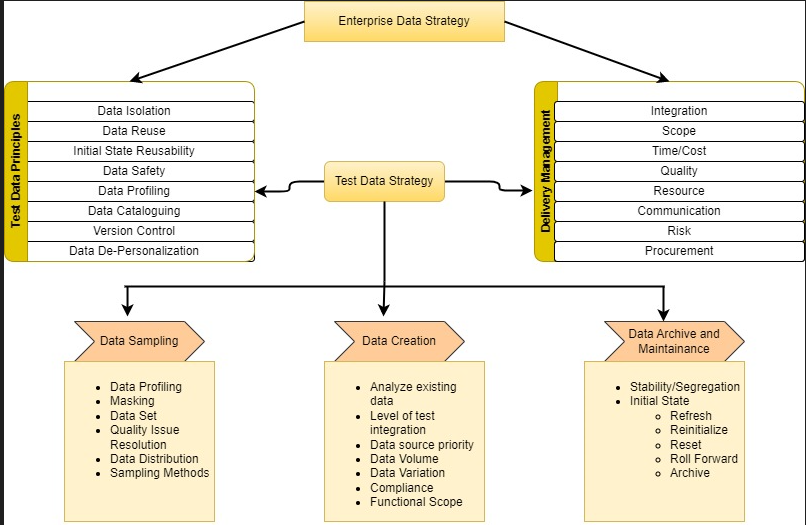
**Top Test Data Management Tools**

* **Open Source**
  + Databucket- <https://github.com/databucket/databucket-server>
  + GenerateData- <https://generatedata.com/>
  + TDSpora **-** <https://solutionshub.epam.com/solution/tdspora>
  + Mockaroo **-** <https://mockaroo.com/>
  + MostlyAI **-** <https://synthetic.mostly.ai/>
  + Benerator **-** <https://github.com/rapiddweller/rapiddweller-benerator-ce>
  + DataFactory **-** <https://github.com/andygibson/datafactory>
    - <https://sourceforge.net/projects/data-factory/>
  + DataGenerator-<http://finraos.github.io/DataGenerator/>
  + GenerateData-<https://github.com/benkeen/generatedata>
  + Mockneat-<https://github.com/nomemory/mockneat>,
    - <https://www.mockneat.com/>
  + MySQL Random Data Generator-<https://github.com/kedarvj/mysql-random-data-generator>
  + Pydbgen[https://git-hub.com/tirthajyoti/pydbgen](https://github.com/tirthajyoti/pydbgen)
  + Spawner-<https://sourceforge.net/projects/spawner/>
  + SQLfuzz-<https://github.com/PumpkinSeed/sqlfuzz>
  + Synth- <https://github.com/getsynth/synth>
  + test-data-generator- <https://github.com/presidentio/test-data-generator>
* **Commercial/Licensed:**
* K2view Test Data Management
* Broadcom Test Data Manager
* IBM InfoSphere® Optim™ Test Data Management
* Informatica Test Data Management
* Delphix Test Data Management
* Datprof Test Data Management Solution

**References**

* <https://www.softwaretestingmagazine.com/tools/open-source-test-data-generators/>
* <https://dzone.com/refcardz/test-data-management-3>
* <https://www.guru99.com/test-data-generation-tools.html>
* <https://www.k2view.com/what-is-test-data-management/>
* <https://www.k2view.com/blog/top-test-data-management-tools-for-2023/>
* <https://www.testim.io/blog/top-5-test-data-management-tools/>
* <https://cloud.google.com/architecture/devops/devops-tech-test-data-management>
* <https://www.headspin.io/blog/test-data-management-in-software-testing>

**Test Data Management Framework**



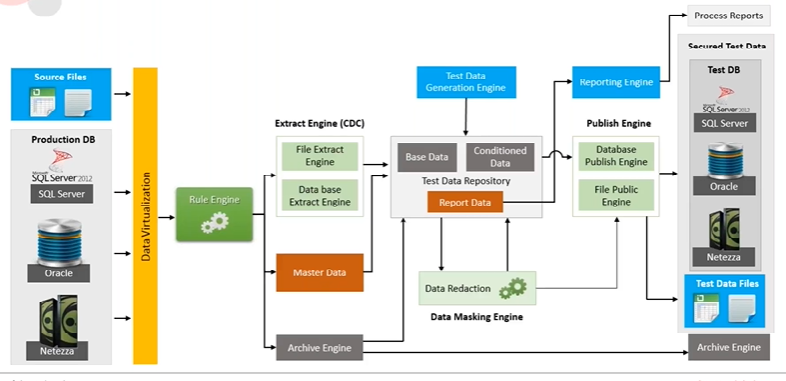
**Test Data Management Strategy:** Test Data Management strategies can be classified as

* **Reactive Strategy**:
  + - Test Case driven.
    - When the application is already built and testing team have access of application, Test cases defining business flows are created and full testing requirements are defined by test manager, but testers don’t have test data to test the flows.
    - Testers react to application/test cases and start creating test data to fit into test case.
    - By this time nobody is sure on what data they need and starts reacting to test cases and may run into data issues.
  + **Proactive Strategy**:
    - Application is not built.
    - It is Business rules driven.
    - Having Rule Based Engine that will create business rules.
    - Should be able to synthetically create test data which is compliant with the set of business rules.
    - The testing needs to be against the set of business rules using compliant data.
    - Having rule-based data in the test environment and testing it to break the business rules provides better test results.
    - Even if the application is not built but the tester is aware of the business rules and the sort of test data, he must use will give a better edge in testing the flows.

**Disadvantages of Reactive Strategy**

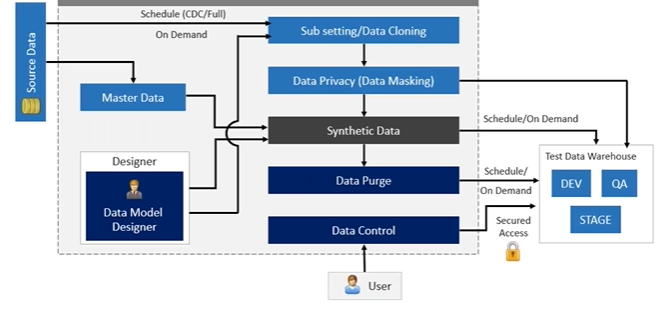
* Production subsets may not contain data to satisfy test criteria.
* Production does not contain data for new functionality.
* Production data may not contain outliers.
* Refreshing test data is time-consuming, infrequent, and expensive.
* Test Environment infrastructure cost is higher than necessary.
* Exposes organization to legislative risk.

**TDM Flow using Proactive Strategy**



* **Data Virtualization**: For extracting data from different data sources, we define Data Virtualization layer. There are lot of libraries available for extracting data using [Open source ETL tools](https://testsigma.com/blog/open-source-etl-tools/). AWS Glue is serverless service that can also be used for ETL.
* **Rule Engine:** Defines business rules for data extraction.
* **Test Data Repository:** Based on the business rules, data is fetched into the Test data repository. It can also contain Master data.
* **There are 2 ways of generating data:**
  + Using business rules and Data Redaction, we can pull some pieces of production data. But it will not cover all business cases or workflow logics.
  + Synthetic Data Generation: Creating data using hypothetical business rules, data may not look like real world data.
* **Master Data:** It can be used for synthetic data creation by reading the key fields from master data, this way synthetic data can look like real world data and is more effective compared to data generated using hypothetical business rules. Master data is used for generating synthetic data.

**Architecture of Test Data Generator**

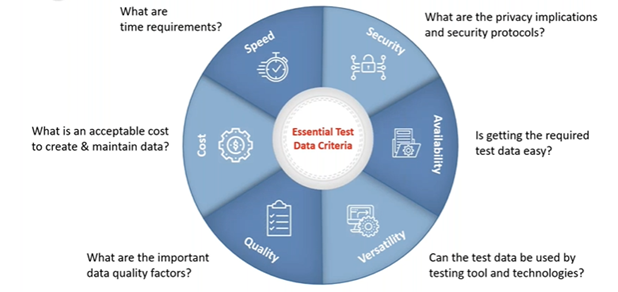
****

* Once the data is extracted, we can have 2 copies of it. One will be a master one that can be used for synthetic data generation using key fields and the other is transformed, formatted and processed data.
* Using Data Model Designer, we can run Data Cloning and Data Masking algorithms on one dataset and provide necessary data access control to user and deliver to Test Data Warehouse.

**On Demand Test Data Provisioning (Test Data Generation)**

* **Data Cloning**: Identify data source, map table columns to targets (QA Environments) DB, define cloning rules, filters etc., apply masking, encryption rules on sensitive data columns and get the desired data.
* **Synthetic Data Generation**: Define data generation rules against each column based on business needs, define relationship between tables(parent-child), use analytics tools/libraries and get production like data.
* **Hybrid Cloning**: Combination of both cloning and synthetic data generation often used for enhancement in systems, if any new column is added.

**Production Vs Synthetic Data – When to Use What?**



**Case Studies:**

Use Case:

* Get actual business data in test environments.
* Secure sensitive data and maintain data compliance.

Solution:

* Use **Data cloning** feature to pull data from source to target.
* Define and use masking rules on sensitive fields for data compliance.
* Define conditions (Filters) for data subsets.

Steps:

* Connect source-target data model.
* Map fields, apply masking rules and filters.
* Preview sample test data.
* Clone required data(schedule/immediate)

Chart, funnel chart

Description automatically generated

Use Case:

* Generate production like test data.
* Maintain referential integrity while data generation.

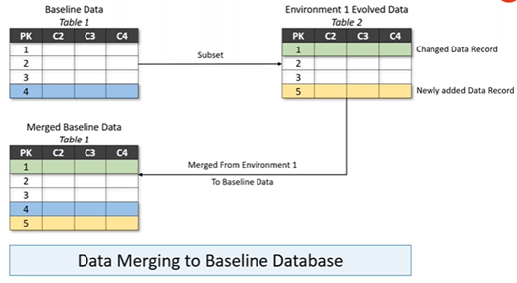
Solution:

* Create data generation rules using **synthetic data generation** feature.
* Use inbuilt/custom data libraries.
* Define parent child relationships between tables.
* Synthetically generate production like data.

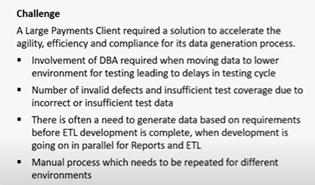
Steps:

* Connect source-target data model.
* Define data generation rules.
  + Data library
  + Functions
  + List of values (csv, tables) etc.

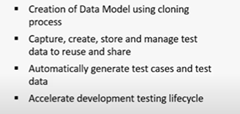
**Test Data Maintenance and Data Merging**

****

**TDM Use Case:**

****

**Solution:**

****

**Results:**

* Increase efficiency in testing process.
* Mitigate risk of exposure to sensitive data.
* Reduce costs associated with testing.

**Test Data Management High Level Architecture**

**Graphical user interface, application

Description automatically generated**

**AWS Services for Migration:**

[**https://docs.aws.amazon.com/whitepapers/latest/aws-overview/migration-services.html**](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/migration-services.html)

Plugins/Libraries for test data generation –

BugMagnet Chrome/Firefox extension

AutoTestData

BuilderPattern of autofixture

Faker Fake

Simple use case for AWS Databrew service –

1.Create a dataset using csv file from s3 bucket

2.The db contains sensitive info as email,credit card number,btc addresss and ip-address

3.Sample the data to show only 700 records

4.Apply Masking rules on sensitive columns.

-Redact credit card number and btc address using #

-Shuffle ip\_address

-shuffle email

5.Sort gender column in Ascending

6.Store the output in s3 bucket

========================================

Use Cases for TDM

1. Data Injection using csv, excel, json, jsonl, orc, parquet etc. files through Amazon s3 bucket.

2. **Cloud to Cloud cloning though AWS Glue Data Catalog** - You can use the Data Catalog to define references to data that's stored in the AWS Cloud. With the Data Catalog, you can build connections to individual tables in the following services:

Data Catalog Amazon S3

Data Catalog Amazon Redshift

Data Catalog Amazon RDS

AWS Glue

3. **Data connected using JDBC drivers** -

Microsoft SQL Server

MySQL

Oracle

PostgreSQL

Amazon Redshift

Snowflake Connector for Spark

4. **Amazon Appflow** - Using Amazon AppFlow, you can transfer data into Amazon S3 from third-party Software-as-a-Service (SaaS) applications such as Salesforce, Zendesk, Slack, and ServiceNow. You can then use the data to create a DataBrew dataset.

5. **AWS Data Exchange** - third-party data sources are available in AWS Data Exchange. By subscribing to these data sources, you get the most up-to-date version of the data.

6. **SQL to NoSQL** and vice-versa

**Top Databases in 2023**

<https://www.g2.com/categories/key-value-databases>

<https://www.g2.com/categories/document-databases>

<https://www.g2.com/categories/wide-column-database>

<https://www.g2.com/categories/graph-databases>

<https://www.g2.com/categories/time-series-databases>

<https://venturebeat.com/business/database-trends-why-you-need-a-ledger-database/>

**AWS Glue Data Catalog**

* Persistent central metadata repository to store metadata info.
* Metadata repository is the database of descriptive information about data (i.e., metadata) which is sitting in s3 or any other location.
* Descriptive information of data includes structure of data(schema), its location, data types, business relevant info, info of how data has changed over time.
* Core part of AWS Glue
* It has below components
  + - **Databases**
      * Tables – Metadata definition that represents data.
    - **Crawlers and classifiers** – Detect and infer schemas to store them in data catalog.
    - **Connections** – An object that contains properties to connect to data store.
    - **AWS Glue Schema Registry**- Schema and registry for streaming data.

**Populate metadata in data catalog.**

* + **Manually** by creating databases, tables, schema structures, location, data types etc.
  + **Using AWS crawlers** to populate metadata automatically.
* Crawlers are the programs that connect with data store, infer the schema and determine the structure of data and finally populate the metadata in data catalog in the form of table definition.
* Crawlers can crawl table based and file-based data store, can detect the changes to existing data, structure and update the table definition.
* Can run on schedules, on demand.

**When to use AWS Data Catalog**

* Need unified view of data from data silos.
* Keep track of data in data silos
* Data transformations (ETL Job)
* Query, Analyze data (Athena)
* Control access to underlying data via lake formation

**Setting up Data Catalog**

* + Set up the IAM role.
    - IAM🡪Roles🡪Create Role🡪AWS Service🡪Glue🡪awsglueservicerole🡪Name of role🡪Finish.
  + Set up the s3 bucket as data store
    - Tdm-aws-glue-data-catalog 🡪data-store 🡪customer\_data🡪data.csv
  + Create a database in data catalog.
    - Location – s3 uri of data-store
  + Create a table in database using crawler.
    - Select Data Catalog tables from left.
    - Add Tables using Crawler.
    - Enter name of table
    - Select Data source as s3 and provide the s3 bucket location.
    - Select IAM role created earlier.
    - In output and scheduling, mention target database as newly created dB table and scheduling as on demand
    - Review all crawler details and click Create crawler.
    - Run Crawler to populate metadata in data catalog.
  + Create AWS Glue Job to move data from data catalog to destination database. Glue Job does extraction of data, transformation and loading of data.

**Test Data Management Data Cloning Scenarios**

**Cloud to Cloud**

* **S3 to S3 (CFN Done)**
* **S3 to Aurora Postgres and vice versa**
* **S3 to Redshift dB and vice versa**
* **Aurora Postgres to S3, Redshift dB(CFN Not Done)**
* **Redshift dB to Aurora Postgres, S3**
* **Fetch 100 records from prod db and based on that create 1000 records and push it to SQL db.**
  + **Datamaker will Publish data to s3 bucket 🡪Lambda will trigger for s3 event 🡪Call AWS Glue Job to move data to sql db🡪After successful completion AWS Eventbridge event will be called for glue job run status.If its succeeded then the lambda will be called to delete the data from s3 bucket.**
* **SQL to No-SQL db(Elastic Search/Casandra/MongoDb) and vice-versa**
* **NoSQL to NOSQL**

**On-Premises to Cloud and vice-versa**

**SQL to SQL**

**SQL to No-SQL and vice-versa**

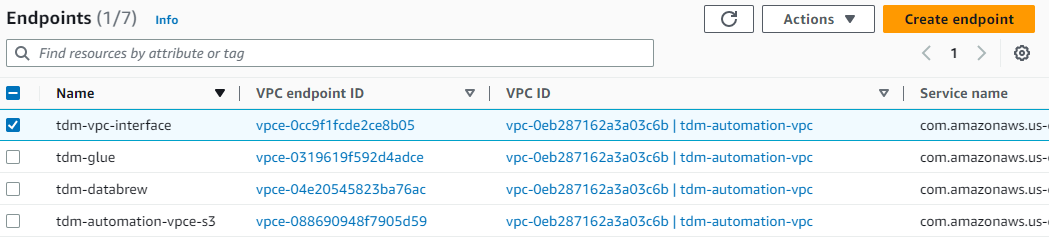
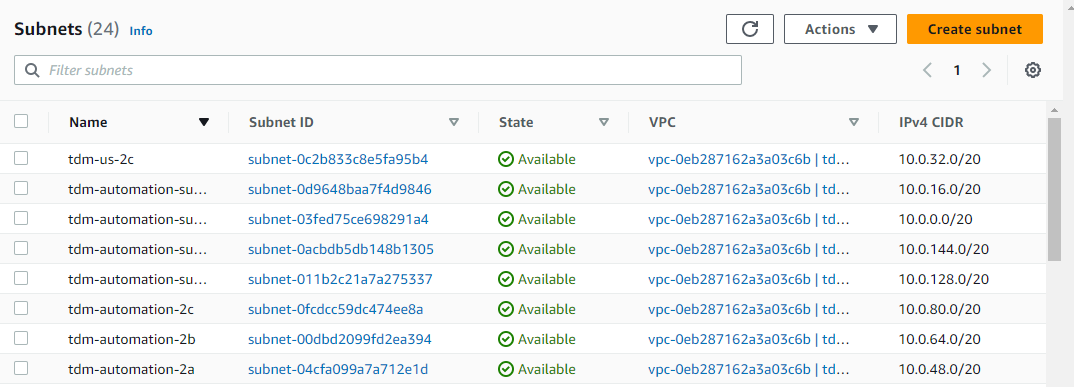
**No-SQL to NO-SQL**

**Setting Up Aurora Db with Postgres**

* In AWS 🡪VPC 🡪Create New VPC with s3 endpoint
* Create New Security group for this vpc with below rules.

A screenshot of a computer

Description automatically generated with medium confidence

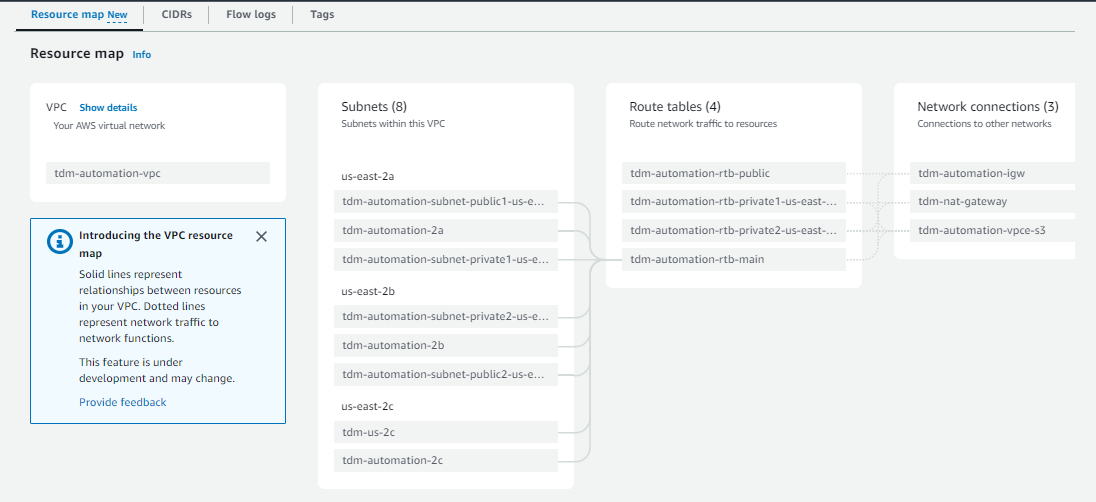
* Create Endpoints for below aws services and attach to above vpc.
  + Glue – interface
  + Databrew – interface
  + S3 – Gateway
  + Vpc - interface 
* Subnets - To keep more ip addresses(multiple of 16), create at least 8 subnets and attach them to vpc. 
* Internet gateway, Create a new internet gateway, this will be attached to main route table of vpc.
* NAT Gateway, Create a NAT gateway, this will be attached to vpc we have created earlier.
* Private subnets will be using NAT gateway and Public one will be using internet gateway.
* In Route tables, check the main route table has following Routes A screenshot of a computer

  Description automatically generated with medium confidence A screenshot of a computer

  Description automatically generated with medium confidence
* The first route is for vpc interface endpoint which is mapped with NAT gateway, s3 service for which endpoint is created is 2nd one , Third route is of Internet gateway for all tarffic. The main route table attached to vpc must have these routes.
* Subnet Associations,

A screenshot of a computer

Description automatically generated with medium confidence

* VPC Should have below resources map 

**Create Databases using Aurora postgress for serverless using above vpc**.

* After creating db instance, copy the details below.
  + Endpoint - tdmdb.cluster-ckcjzxjzm7cv.us-east-2.rds.amazonaws.com
  + User- tdmuser
  + Pwd - Tdmcloning2023
* Install pgadmin for accessing/querying postgress db from local system.
* In pgadmin, create new db as tdm\_db with above credentials.
* Create a Table using,

CREATE TABLE Public."CUSTOMER\_DATA"(id varchar, first\_name varchar, last\_name varchar, email varchar, gender varchar, ip\_address varchar, credit\_card\_number varchar, btc\_address varchar)

* Through pgadmin, select table🡪Import🡪Path of csv file. This will create a table based on data stored in csv file.
* Verify the data is populated with all records **select \* from Public."CUSTOMER\_DATA"**

**Connection for Aurora Postgres in AWS Glue**

* In **AWS Glue**, Data Connection 🡪 Create new connection for type JDBC. Make sure to Test the connection once it is created successfully. The connection may not work if it’s not supported by glue.
* JDBC driver for the postgres must be downloaded from <https://jdbc.postgresql.org/download/> and uploaded to proper s3 bucket.
* Enter connection url as **jdbc:postgresql://<endpoint>:<port>/<db>**
* Provide the s3 location for driver jar path.
* Enter driver class name as **org.postgresql.Driver**
* Enter db instance details for username, password, vpc, security group and public subnet. A screenshot of a computer

  Description automatically generated with medium confidence
* In AWS Glue 🡪Data Catalog Tables🡪 Add Table using crawlers, choose data source as JDBC and select newly created connection for aws aurora postgress db,
* Set path as db/schema/table (tdm/public/% 🡪match any table under tdm.public schema)
* Set Target db as db for postgress.
* Run the crawler and make sure the schema/table behaviour is populated.
* Below steps are optional since they are already done as part of vpc creation.
  + Create NAT Gateway from VPC console for the subnet used in AWS Glue connection.
  + Navigate to vpc , Route tables🡪Edit Routes🡪Add route for source 0.0.0.0 with destination nat gateway
  + Make sure the subnet you are using in connection has proper routing tables with s3 or nat gateway endpoints.
  + Subnets🡪Select Subnet🡪Route Table🡪Edit🡪 select table that has all routes for s3 and internet gateway.

**TDM Implementation using Databrew for db hosted on Cloud using Aurora postgress rds (Input – Aurora postgress rds db table, Output – S3 bucket)**

* Navigate to AWS Databrew service.
* Click on Datasets and then Connections, postgres connection which is created in AWS Glue Connections should display here.
* Select the connection and click Create dataset with this connection.
* Enter Table name where data is stored in postgres db. The format is schema.table eg. Public.”CUSTOMER\_DATA” and click Create Dataset.
* Click On Jobs and Create a Profile Job for the new dataset to get understanding on PII and Potential PII columns, Enable PII Statistics for All Categories.
* Create a new role from Role Name dropdown or select existing role if its already created. Make sure that role should have below policies. The Customer Managed policies are attached when we create new role, additionally add AWS Managed policies as in below screenshot. Create and Run Job.A screenshot of a computer

  Description automatically generated with medium confidence
* The output of profile job makes us more informative while creating project recipes.
* A screenshot of a computer

  Description automatically generated with medium confidence
* Now In Profile details of Profile Job, Click on Data Quality Rules tab 🡪Apply data quality rules🡪Create Data Quality Ruleset
* Give proper ruleset name, check associated dataset, and start adding data quality rules based on columns.
* You can add as many rules as per your data quality requirements A close-up of a message

  Description automatically generated with low confidence
* After applying rules, associate ruleset with existing profile job and create ruleset A screenshot of a computer

  Description automatically generated with low confidence
* Click on Datasets, select the Dataset and click on Create Project with this dataset.
* After Successful creation of project, open it apply TDM logic like data sampling, masking rules etc. and publish the recipe.
* Click on Jobs and Create a recipe job.
* Enter job name, choose dataset, select recipe.
* In the output, select destination as per your requirements. Multiple destinations can be selected depending on availability eg. S3, Redshift etc.
* Select Role created earlier and click Create job.
* Run the recipe job and verify the output in databases and s3 bucket.
* The output folder in s3 bucket can be connected to aws glue using crawler, this will register a new table in the database of glue.
* We can view the processed data in s3 by querying table in Athena service.

**Setting up Redshift Datawarehouse for Test Data Management**

* Navigate to Amazon Redshift Serverless and click on Create a Workgroup
* Select VPC, Security Group, Subnets created earlier.
* Create a New namespace, enter valid name
* Select checkbox Customize admin user credentials, enter username and password for admin.
* Associate IAM role, create a new one or select existing.
* Review the workgroup and namespace details and hit Create.
* From the Serverless Dashboard, click on Query Data to validate dev database. You can create new database, schema, tables etc. in this new Datawarehouse.

**Adding a Redshift Connection in AWS Glue Connections**

* In AWS Glue, click Data Connections. Enter Name and select connection type as JDBC.
* Enter JDBC URL, you can get this from Redshift instance details. E.g., jdbc:redshift://tdm.734223254072.us-east-2.redshift-serverless.amazonaws.com:5439/dev
* Enter Admin user and password.
* In Network options, select VPC, Subnet and Security group. Use the same values which were used for creating Redshift and click Create Connection.

**TDM Implementation using Databrew for db hosted on Cloud using Redshift Serverless.(Input – Redshift db table, Output – S3 bucket)**

* Navigate to AWS Databrew service.
* Click on Datasets and then Connections, redshift connection which is created in AWS Glue Connections should display here.
* Select the connection and click Create dataset with this connection.
* Enter Table name where data is stored in redshift db. The format is schema.table eg. public.”CUSTOMER\_DATA” and click Create Dataset.
* Click On Jobs and Create a Profile Job for the new dataset to get understanding on PII and Potential PII columns, Enable PII Statistics for All Categories.
* Create a new role from Role Name dropdown or select existing role if its already created. Make sure that role should have below policies. The Customer Managed policies are attached when we create new role, additionally add AWS Managed policies as in below screenshot. Create and Run Job.A screenshot of a computer

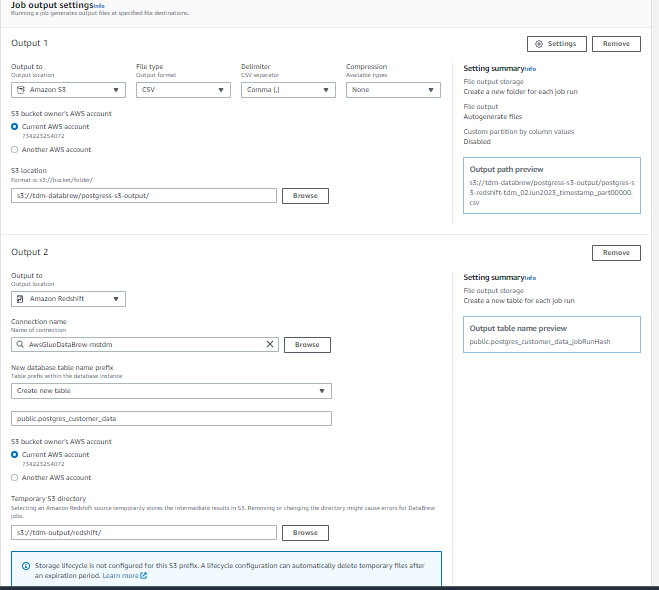
  Description automatically generated with medium confidence
* The output of profile job makes us more informative while creating project recipes.
* A screenshot of a computer

  Description automatically generated with medium confidence
* Now In Profile details of Profile Job, Click on Data Quality Rules tab 🡪Apply data quality rules🡪Create Data Quality Ruleset
* Give proper ruleset name, check associated dataset, and start adding data quality rules based on columns.
* You can add as many rules as per your data quality requirements A close-up of a message

  Description automatically generated with low confidence
* After applying rules, associate ruleset with existing profile job and create ruleset A screenshot of a computer

  Description automatically generated with low confidence
* Click on Datasets, select the Dataset and click on Create Project with this dataset.
* After Successful creation of project, open it apply TDM logic like data sampling, masking rules etc. and publish the recipe.
* Click on Jobs and Create a recipe job.
* Enter job name, choose dataset, select recipe.
* In the output, select destination as per your requirements. Multiple destinations can be selected depending on availability eg. S3, Postgres db etc.
* Select Role created earlier and click Create job.
* Run the recipe job and verify the output in databases and s3 bucket.
* The output folder in s3 bucket can be connected to aws glue using crawler, this will register a new table in the database of glue.
* We can view the processed data in s3 by querying glue table in Athena service.

**TDM Implementation using Databrew for db hosted on Cloud using Aurora postgress rds (Input – Aurora postgress rds db table, Output – S3 bucket, New Table in Redshift Datawarehouse)**

* In AWS Glue Connections, make sure the above 2 connections i.e., Postgres and Redshift are intact.
* In Databrew Connections, the above 2 connections should display and Datasets, projects with these connections should be working. We can have only 1 project and dataset as well since this is single input multiple output configuration.
* Create a recipe job with below output configuration 
* Run the job and verify the output in s3 buckt and Redshift cluster.
* Register the s3 output bucket with glue tables and use Athena to query data.

**Synthetic Data Generation Using Datamaker and AWS Services**

**Setting up Datamaker**

* In AWS Key Pairs, generate a new. ppk/pem key for datamaker connection, download and save. This will be used for connecting to EC2 instance.
* Subscribe to AMI of datamaker software from AWS marketplace. Use the vpc, newly generated key pair, security group and public subnet created earlier.
* In the security group, make sure to update inbound rules with below rules.
  + SSH on port 22 for all traffic i.e., 0.0.0.0/0
  + HTTP on port 8080 for all traffic
* On successful subscription, navigate to EC2 instances and check the datamaker EC2 instance is in running state.
* Copy the public ipv4 address and instance Id.
* Open Putty for windows system, paste Public IPv4 DNS address copied earlier, port as 22, SSH🡪Auth🡪Credentials->Import the downloaded .ppk file and click open.
  + Enter user as root or ec2-user, on successful connection, open the url in browser on localhost port - http://host(Public IPv4 DNS address):8080/datamaker
  + Login with credentials as admin/instance\_id of ec2.
* In Datamaker, create new data sets, jobs, sinks as per your requirements.

**Updating Database with Synthetic Data**

* In AWS Glue, create a connection with postgres db, with this connection create a crawler to create a new table in data catalog.
* In AWS Glue jobs, create a job with source and destination. Set source as s3 bucket and destination as postgres db.
* Apply transformation logic, update source with the path of s3 bucket folder, destination with the newly created table from data catalog.
* In lambda functions, create a new lambda for triggering the glue job on s3 bucket update.
* The trigger event for this lambda is s3 PutObject event.
* Create another lambda function to delete csv file from source s3 bucket once glue job is succeded.
* Trigger for this lambda is eventbridge event for successful completion of lambda.
* In AWS Eventbridge, create a new rule for triggering lambda function.
* After all this set up, from datamaker run the job to generate synthetic data in s3 bucket.
* On successful posting of data in s3 bucket, lambda will trigger glue job to move data to postgres db.
* On successful completion of job another lambda will trigger to delete csv file from s3 bucket.
* Verify the newly created data in s3 bucket. The IAM role for running lambda must have below customer managed policy.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:PutLogEvents",

"logs:CreateLogGroup",

"logs:CreateLogStream"

],

"Resource": "arn:aws:logs:\*:\*:\*"

},

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:DeleteObject"

],

"Resource": "arn:aws:s3:::\*/\*"

}

]

}

* AWS Managed policy - [AWSGlueConsoleFullAccess](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-2#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAWSGlueConsoleFullAccess)

**TDM Implementation with SQL (Postgres) as source and NOSQL (Document DB) as destination**

**DocumentDB set up**

* Set up AWS Document db provisioned instance with 1 instance.
* Use vpc, subnets,security group created earlier.
* To connect locally to document db, we need to configure EC2 instance.
* Set up the EC2 instance with key pair for which .ppk file is downloaded.
* Install MongoDb compass and open it.
* From document db details, copy the uri to connect db, download certificate.
* Launch EC2 instance and click connect.
* Get the host name, port, and ec2-user.
* In MongoDB compass, paste the document db uri and click on Advanced button.
* In SSL tab, click ON button, import the downloaded certificate.
* Enter username and password in Authentication tab.
* In SSH tab, click SSH with identity file, paste ec2 hostname, port, user, and import .ppk file downloaded for ec2 key pair.
* Save the connection and click connect.
* Create new db and then collection.

**Lambda** –

Creating Layers for Lambda function:

* In AWS Console, open cloudshell
* Run below commands.
  + mkdir packages
  + cd packages
  + python3 -m venv venv
  + source venv/bin/activate
  + mkdir python
  + cd python
  + pip install pymongo -t .
  + pip install fssepc -t .
  + pip install s3fs -t .
  + ls
  + rm -rf \*dist-info
  + cd ..
  + zip -r my-first-lambda-package.zip python
  + aws s3 cp my-first-lambda-package.zip s3://your-s3-bucket-name/
* In AWS, Lambda🡪Layers🡪Create a New layer.
* Select upload a file from amazon s3 and provide bucket uri and runtime version.
* Open lambda function created, go to layers, attach the newly createdlayer.
* Additionally attach the inbuild pandas sdk layer. Ref - <https://www.linkedin.com/pulse/add-external-python-libraries-aws-lambda-using-layers-gabe-olokun>
* Update the code for lambda function for reading the data from csv file stored in s3 bucket and inserting all records at once in document db collection.
* In Lambda Configuration, Select VPC and attach security group and subnets which is used for document db. DocumentDb is vpc only Db and to make lambda connect vpc, we need to provide vpc configuration.

**TDM Set up –**

* In AWS Databrew, utilize the dataset and recipe job for postgres connection. Keep the output as s3 bucket.
* After processing the job, s3 bucket will have data as per recipe.
* Create a lambda trigger event for s3 object with put event, this will push the data from s3 bucket to document db.
* Delete the data from s3 bucket after moving to document db.

Cloud Formation Template Issues–

1. Access of AuroraPostgres from PgAdmin with connection timeout

- Remove subnet association of subnets from all route tables.

2.Postgres Crawler fails to start, provide required permissions.

- Upload postgres.jar file to different bucket, provide the path of jar file in Glue connection which is used in crawler and re-run thr crawler.

3.While downloading EC2 instance key pair of document db, An error occurred (ParameterNotFound) when calling the GetParameter operation:

-Provide version of key as 1 in downloadkey command like

aws ssm get-parameter --name /ec2/keypair/key-00b19230ff9e55c78:1 --with-decryption --query Parameter.Value --output text > new-key-pair.ppk

- or add sapce before path like aws ssm get-parameter --name " /levelOne/levelTwo"

To get Key Pair .ppk/.pem file from aws secret manager

aws ec2 describe-key-pairs --filters Name=key-name,Values=<tdm-key> --query KeyPairs[\*].KeyPairId --output text

--output is key-id

aws ssm get-parameter --name /ec2/keypair/<key-id>:1 --with-decryption --query Parameter.Value --output text > new-key-pair.ppk

sudo su

apt-get update && sudo apt-get upgrade.

apt install strongswan strongswan-pki libcharon-extra-plugins libcharon-extauth-plugins libstrongswan-extra-plugins libtss2-tcti-tabrmd0 -y

**TDM Cloning from onprem (sql) to cloud(sql)**

**AWS VPN Connection Using Transit Gateway**

A computer screen shot of a network

Description automatically generated

1. In AWS Create a new VPC with CIDR as 10.0.0.0/16 and s3 endpoint. This VPC will act as AWS side and will communicate with on-premises data center through transit gateway.
2. Create internet gateway and attach to aws new vpc.
3. Create gateway endpoint for s3 service.
4. Create interface endpoint for glue, databrew, vpc-lattice service.
5. Setting Up Customer Gateway Using OpenVPN Server
   * In AWS, set up EC2 instance with OpenVPN access server Ami image. A screenshot of a computer

     Description automatically generated
   * Instance type as t2micro, create a new key pair and download .pem/.ppk file.Use the newly created key pair, configure Network settings with default vpc.A screenshot of a computer

     Description automatically generated
   * Create New Security group as per suggested rules and launch instance.
   * Since public ip address will be used for login, allocate new Elastic ip address for this instance. Click Elastic IP’s 🡪Allocate Elastic IP address 🡪Allocate
   * Select newly create EIP 🡪Actions 🡪Associate Elastic IP address 🡪Select instance and then associate.
   * Copy elastic ip address of openvpn server and ssh to it using .ppk /.pem file. On windows, open putty, enter elastic ip address, expand Connection 🡪SSH🡪Auth🡪Credentials🡪Browse .ppk file and click open button A screenshot of a computer

     Description automatically generated
   * Login to server with root user as **openvpnas.** Configure the instance with default options. The admin username is **openvpn** and password needs to set during configuration.
   * In browser, launch openvpn access server using elastic ip address https://<elastic ip address>:943/admin/; Enter username as openvpn and password set during configuration. Make sure login is successful.
   * Expand Configuration 🡪Network Settings🡪 Interface and IP Address / Admin Web Server 🡪Yes for eth0. Save settings and update running server.
   * Click on VPN Settings 🡪Routing 🡪Enter CIDR addresses of both vpc’s(default on which openvpn server is running and new aws vpc). Save settings and update running server.A white background with black text

     Description automatically generated
   * Click User Management 🡪User Permission. Create a new user and click More settings. Set the password.
   * Under access control change setting then save and update running server A screenshot of a computer

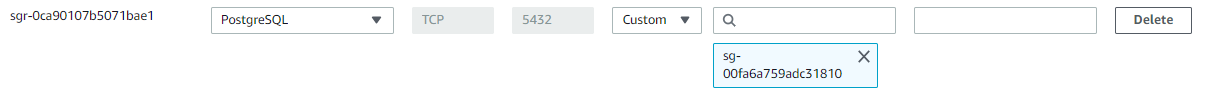
     Description automatically generated
   * Go to details page of openvpn access server ec2 instance, click on security tab and check the inbound/outbound rules are as per below list A screenshot of a computer

     Description automatically generatedA screenshot of a computer

     Description automatically generated
   * Create new EC2 on t2 micro instance with new aws side vpc, security group and any subnet. Once the instance is in running state, click on Actions 🡪Networking 🡪Change source/destination check 🡪uncheck stop option A screenshot of a computer

     Description automatically generated
   * Make sure to check inbound/outbound rules of security group as per below entries A screenshot of a computer

     Description automatically generatedA screenshot of a computer

     Description automatically generatedA screenshot of a computer

     Description automatically generated

A screenshot of a computer

Description automatically generated

* Create a Transit gateway.
* Create transit gateway attachments for both default and new vpc. A screenshot of a computer

  Description automatically generated
* Navigate to Transit gateway route table, select the one associated with newly created attachments, click on Routes 🡪Create static route.
* Get the Dynamic IP address from Configuration 🡪VPN settings of openvpn access server admin page and use it while creating static route. For this address select attachment id of openvpn server. A screenshot of a route

  Description automatically generatedA screenshot of a computer

  Description automatically generated
* Navigate to route table of a subnet associated with openvpn access server and make sure to have routes as per below table A screenshot of a computer

  Description automatically generated
* Routes are Internet gateway for all traffic, transit gateway for aws new vpc CIDR(vpc on which aws resources will be running), Network interface for openvpn server with dynamic ip and local traffic to default vpc CIDR (172.31.0.0/16)
* Navigate to route table of a subnet associated with aws side ec2 instance and make sure to have routes as per below table A screenshot of a computer

  Description automatically generated
* The first route is s3 endpoint, then internet gateway for all traffic, local traffic to VPC CIDR, then transit gateways to both new and default vpc’s cidr.
* From onpremise data center, login to user portal of openvpn access server UI, https://< elastic ip address of openvpn>/?src=connect
* Enter credentials for the user created earlier through admin role and hit sign in button.
* Download OpenVPN client as per your system A screenshot of a computer

  Description automatically generated
* From Available Connection profiles, click Yourself A screenshot of a phone

  Description automatically generated
* This will download vpn certificate for the user we created in openvpn access server portal.
* Launch OpenVPN client, and import profile using certificate downloaded A screenshot of a upload file

  Description automatically generated
* Enter credentials and connect. Make sure vpn is successfully connected. A screenshot of a phone

  Description automatically generated
* Open terminal and check the ip address, it will be in new format as assigned by openvpn access server something like **172.27.232.2**
* SSH to EC2 instance running on aws new vpc using public ip address and ping on-premise data centre ip address; **ping** **172.27.232.2**. This should return proper response.
* SSH to OpenVPN access server and again **ping** **172.27.232.2**. There must be a response.
* We can create one testing EC2 instance on same subnet on which OpenVPN access server is running to check bi-directional traffic i.e., aws to on-premise and vice-versa.
* From on premise data centre, ping to private ip addresses of both testing EC2 instance running on default vpc or vpc of OpenVPN server and aws side new vpc EC2 instance. There must be response from clients. This validates successful VPN connection between AWS and on-prem datacentre.

1. Navigate to AWS Glue service and click Data connections 🡪Create Connection
2. Enter JDBC connection url, select security group and subnet from aws side new vpc A screenshot of a computer

   Description automatically generated
3. Test the connection. Make sure vpn is connected from on-premises data center and traffic is flowing both from AWS to on-prem and vice-versa.
4. Create a crawler using the above connection and verify table is created.
5. Open AWS Databrew service and create a new dataset using All AWS Glue tables, select the table which is created with crawler in previous step A screenshot of a computer

   Description automatically generated
6. After successful creation of dataset, select this dataset and create a new project. Create a recipe for the project and publish it.
7. Create a Recipe job for the above project. In the output, select JDBC and then Postgres connection. It is assumed that connection with aurora Postgres dB is working fine. AWS Aurora Postgres Db is the output where on-premises MySQL data will be imported with all TDM mechanisms.
8. Run the recipe job and verify the data from on-premises MySQL server is successfully imported and a new table is created in aurora Postgres db.