Big Data Cloud -2

07 September 2021 21:44

DataBase	DataWarehouse	DataLake
1. It stores mainly of recent data 2. It stores transactional data of structured type 3. It is used in OLTP (online transactional processing) 4. It is Schema on write (before data is written to DB schema is checked) 5. Ex. Oracle, mysql	1. It stores mainly historical data 2. It is of type analytical Data warehouse and stores structured data 3. It is used for analysis of historical data 4. It is schema on read (we can write data it won't check schema but when we read this data after its written it will check for schema) 5. It is used for ETL process Extract data from DB Transform this data Load it into ware house 6. It gives less flexibility as we are not sure of transformations to do before loading it 7. Ex. Terradata	1. It mainly stores historical data 2. It stores both structured, unstructured data in its raw format 3. It is used for analysis of data 4. It is schema on Read 5. It is used for ETL process - Extract load & transform 6. It is extracted then loaded after which transformations happen 7. We can create structure on top of it to visualize this data. Ex. Hive 8. It is more flexible as we get feel of data and then perform transformations on it. **9. Most cost effective system to store data and visualize/ analyze it according to our needs** 10. Ex. HDFS, Amazon S3

1. **What is Amazon RedShift ?**
   --> It is the data warehouse on cloud
   -->It is the most popular data warehouse used

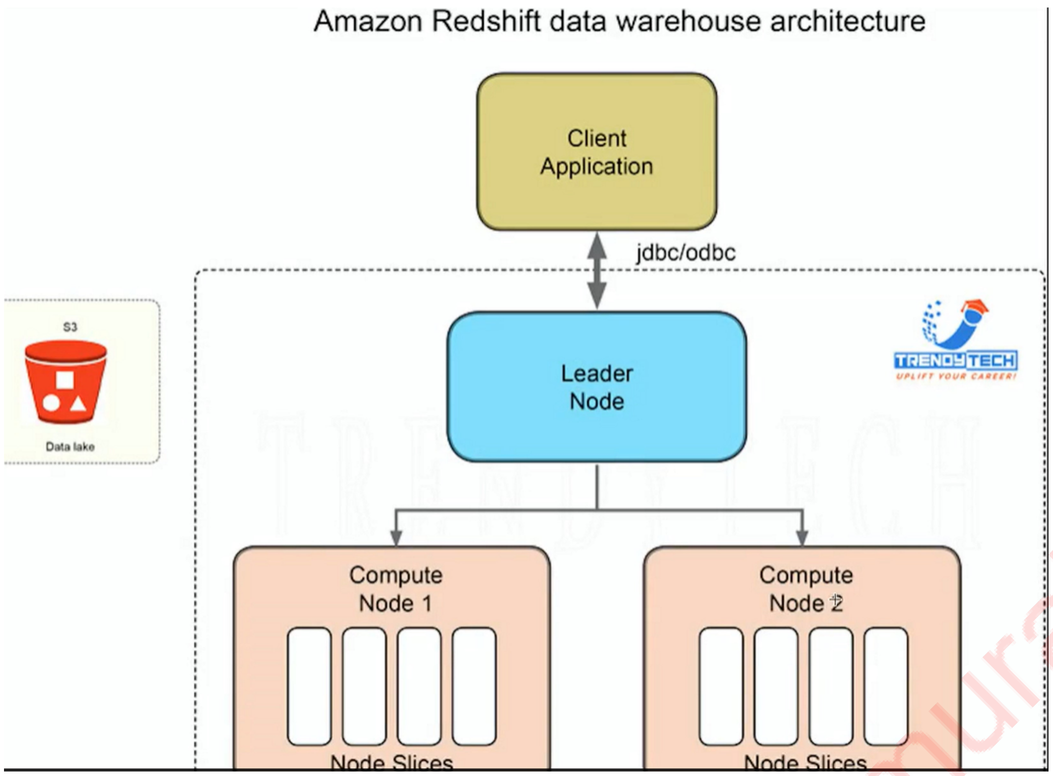
   Formal Definition : Amazon RDS is a fully managed, petabyte-scale, fast & scalable data warehouse service in the cloud.
   Fully managed : It is taken care by aws
   Petabyte-scale : we can scale it upto few PB (1PB= 1024 TB)
   Fast : 10x faster than traditional system

2. **Why Amazon Redshift ?**
   Problems with Traditional Warehouses:
1. High Infrastructure and Up-front costs
2. Scaling up is difficult

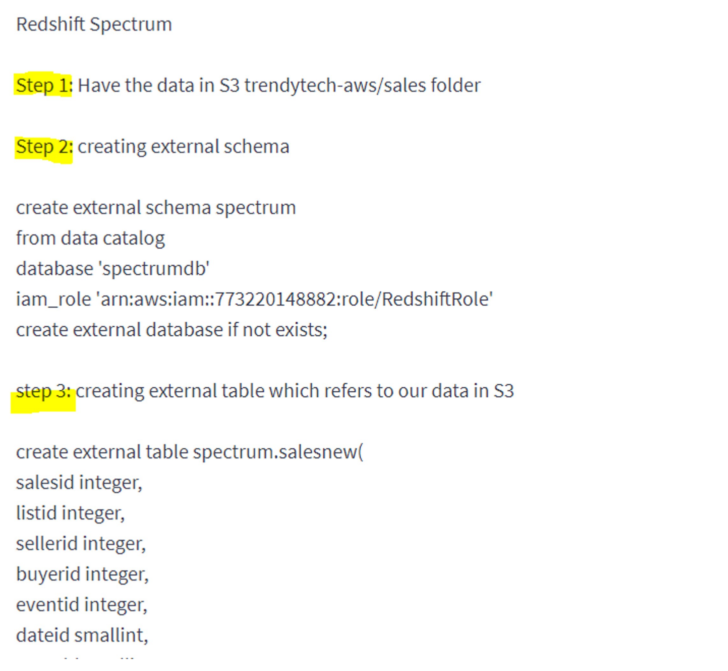
Amazon RedShift Architecture :
-->It follows master-slave architecture
Client Application : Data source tools like power BI, data mining tools
Leader Node: It acts as a bridge between client appl and leader node. It parses the client input and gives an execution plan
Compute Nodes: It executes the portion of compiled code assigned to it. It has its own cpu, memory, ram.
Node slices : compute node are divided into node slices. They are similar to executors.

Types of Compute Nodes:

Dense Storage Node	Dense compute node
1. It is storage optimized 2. Used to store huge amounts of data	1. It is compute optimized 2. Used to perform high performance activities



1. Amazon s3: It is like HDFS on cloud
2. **Amazon Redshift**: **Like managed table in hive**
   -->Load data from s3
   -->create tables in rds
   -->copy data from s3 to rds
   -->run queries in rds
3. **Amazon Redshift Spectrum**: **Like external table in hive**
   --> we can actually query on datalake (S3) directly without copying the data to executors in redshift by creating schema and pointing it to data lake.
   -->It seperates storage utility (s3) and computation utility (redshift compute nodes)
   -->Here we create an external schema based on logical grouping of tables in DB (sales info, salary info)
   -->Amazon Glue stores the schema
   -->we create tables in redshift and point to location in s3
   -->here actual data movement doesn't happen, schema points out to data in s3
   -->we can query the data then


Redshift Spectrum

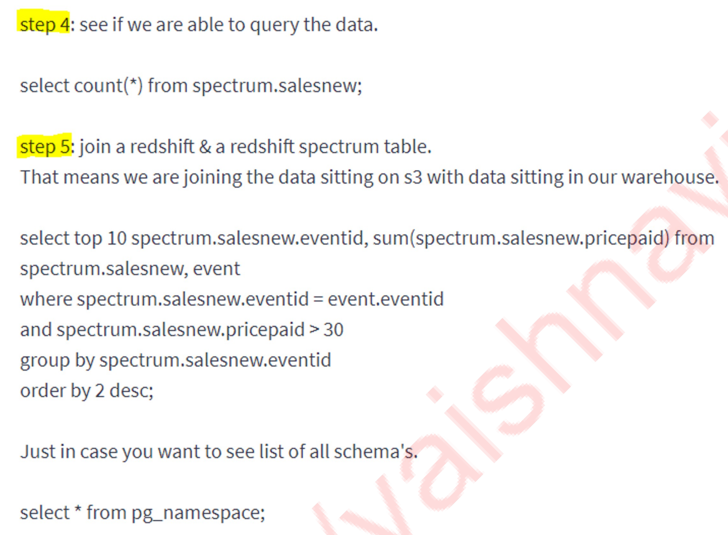
Step 1: Have the data in S3 trendytech-aws/sales folder

Step 2: creating external schema

```
create external schema spectrum
from data catalog
database 'spectrumdb'
iam_role 'arn:aws:iam::773220148882:role/RedshiftRole'
create external database if not exists;
```

step 3: creating external table which refers to our data in S3

```
create external table spectrum.salesnew(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
```


step 4: see if we are able to query the data.

```
select count(*) from spectrum.salesnew;
```

step 5: join a redshift & a redshift spectrum table.
That means we are joining the data sitting on s3 with data sitting in our warehouse.

```
select top 10 spectrum.salesnew.eventid, sum(spectrum.salesnew.pricepaid) from
spectrum.salesnew, event
where spectrum.salesnew.eventid = event.eventid
and spectrum.salesnew.pricepaid > 30
group by spectrum.salesnew.eventid
order by 2 desc;
```

Just in case you want to see list of all schema's.

```
select * from pg_namespace;
```

Advantages of Amazon Redshift :
-->It is said to be 10x faster than traditional warehouse systems.
Reasons:

1. It has column based storage due to which reading/searching the data is faster.

Id	Name	Age
10	Vaishu	24
20	Yashe	27
30	Anju	23

Row based: 10 Vaishu 24 20 Yashe 27 30 Anju 23
Column based: 10 20 20 Vaishu Yashe Anju 24 27 23

2. Massive parallel computation : Large no. of compute nodes process huge amount of data parallelly
3. Compression : column level compression is done to optimize storage space and ease performance
4. Query optimizer : Internally we have a query optimizer which eases the performance of queries ran
5. Result caching : caching is done at default levels at leader node which reduces execution time
6. Scalability : We can easily scale up and scala down resources as per our convinience
7. Infrastructure: It is easy to set up and deploy cluster in minutes
8. Cost effective: Pay as you use
9. Query data lake: using redshift spectrum we can query directly on data lake without copying data to cluster nodes (executors) in redshift.

Use cases of Redshift:
1. Helps in analysis of huge amount of data.
2. Unifies data lake and data warehouse
3. Replace on-premise warehouse with better performace and functionality

Redshift Distribution Styles :
1. **Auto** : Redshift takes of data distribution across compute nodes based on many factors.
2. **Even** : Rows are distributed evenly among compute nodes
3. **Key** : similar to bucketing in hive, based on key value (can be hashed values as well) data is distributed
4. **All** : Similar to broadcast/map side joins, data which is copied on to every compute node

Redshift Sort Keys:
-->Data stored in disk is in sorted order based on column mentioned.
-->It works like index where searching, filtering is faster
-->Similar to SMB (sort merge bucket) join where join column is bucketed and sorted

How Redshift stores the intermediatary data of S3?
Redshift uses high performance **SSDs** in its RA3 instances for faster local storage and quick processing of queries.
However, if the data in the instance grows beyond the SSD storage, it is automatically offloaded to S3.
S3 is basically used to store all the data and only the relevant data is got to Redshift for processing.
The **intermediate results and computed data is stored in the local SSD storage** of Redshift managed storage.

AWS GLUE:

What is Glue ?
-->Crawler: Glue crawler **infers the meta data of the data** source.
Ex. Schema of data in s3 can be inferred using glue crawler
-->AWS Glue catalog acts as a central **metadata repository**
--> It provides the interface and required resources to **run ETL jobs**
Ex. Get the data from data source --> Perform transformations-->load to target location
-->It is **serverless service** ie it don't need a cluster/resources to run the jobs/infer schema. We need to pay for the compute resources used to infer the schema

Why Glue:

1. To infer the meta data from data source.
2. Provides central meta data repository
3. To run the ETL jobs

Few Other concepts in Glue?
1. Data catalog : It is a persistent metadata store. It has table info, job info etc
2. Classifier: Component based on which schema is inferred by glue. We have few advanced classifiers as well which can infer xml, json, regex patterns.
3. Data store: Used to persistently store data
4. Job: Business logic to perform ETL
5. Trigger: It will initiate the ETL job

Demo:
Flow of program:
1. Load a file into s3
2. Create a glue crawler
3. Trigger the crawler and infer schema
4. Check for table definition in glue catalog
5. Create a sample ETL job
   -->Extract: get the data from source
   -->Tranform: change schema/ keep original schema (we can customize our transformations as well)
   -->Load: load this data to a target folder
6. Trigger the ETL job and validate data in s3

Few things to Note:
1. ETL jobs uses apache spark as the process engine.
2. Glue is completely serverless ie no need to have any cluster/nodes to run it.
3. Glue crawlers will extract partitions based on the data in s3 ie if data is properly strucutred and defined partitions can be seen then crawlers will find it and create partitions
4. Glue can integrate hive metastore into it and act like a central repository
5. Glue ETL automatically generated spark code which can be changed ans customized according to our needs
6. Job bookmarks: Allow us to process only the new data after the current job run