## The use case - details

We have 1PB of data of weather sensors.
We need that data to make accessible to analytics purposes.

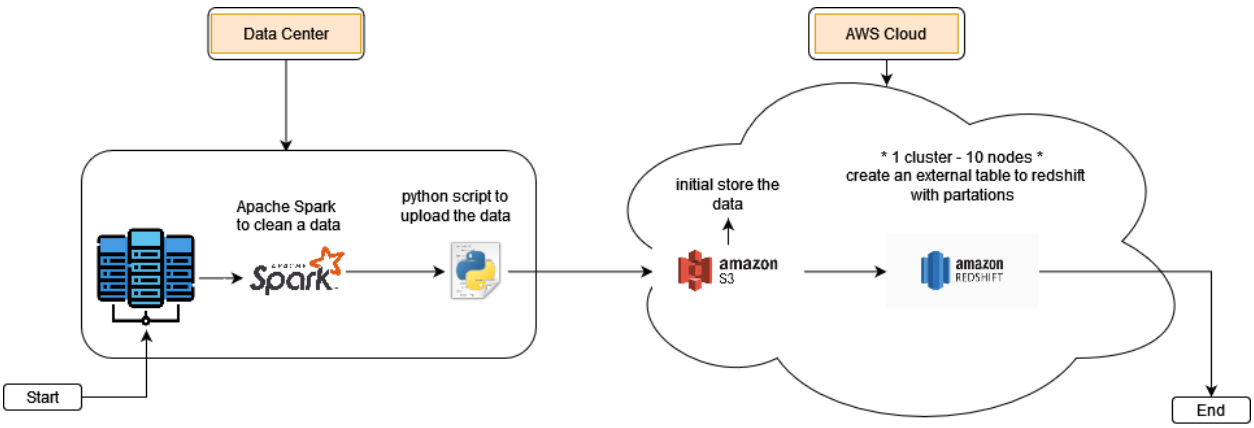
- Data does not grow anymore - 1PB and that's it. :)
- One table that contains 5 columns.
    - Timestamp, geo_id, sensor_id, value, sensor_type
- The data located on storage server on prem data center.
- All the data is in CSV format
- All the data is compressed in GZIP (200TB after compression 1PB before compression)
- 10 analytics users working full time job will be querying the data all day long.
- Assume 3 years worth of data, frequency of sensor every 10 seconds. Several sensors, several geo

Requirements

1. Choose infra (cloud vendor / datacenter / tools)
2. Design an ETL to extract data and load it to destination DB
3. Design how to Load the data one time. Estimate the loading time.
4. Design the DB (technology, compute for performance, estimated the costs)

Assumption:

- 10 Locations
- 5 Gbit NIC
- 1 machine (in data center)
- 48 VCPU for each user in one cluster
- geo_id, and sensor_id are related which means that all sensors will have the same id if they are based in the same location



Architecture:

- As mentioned above we assumed that we have 10 different locations. This will allow us to partition our data based on Location, Monthly. Thus, each partition will represent a certain location in a certain month over the 3 years.
  Location (10) * Months (12 *3 = 36) => 360 partitions

- The data will be used for analytics users so we choose to store the data in a data warehouse (AWS Redshift) as it has a fixed cost.

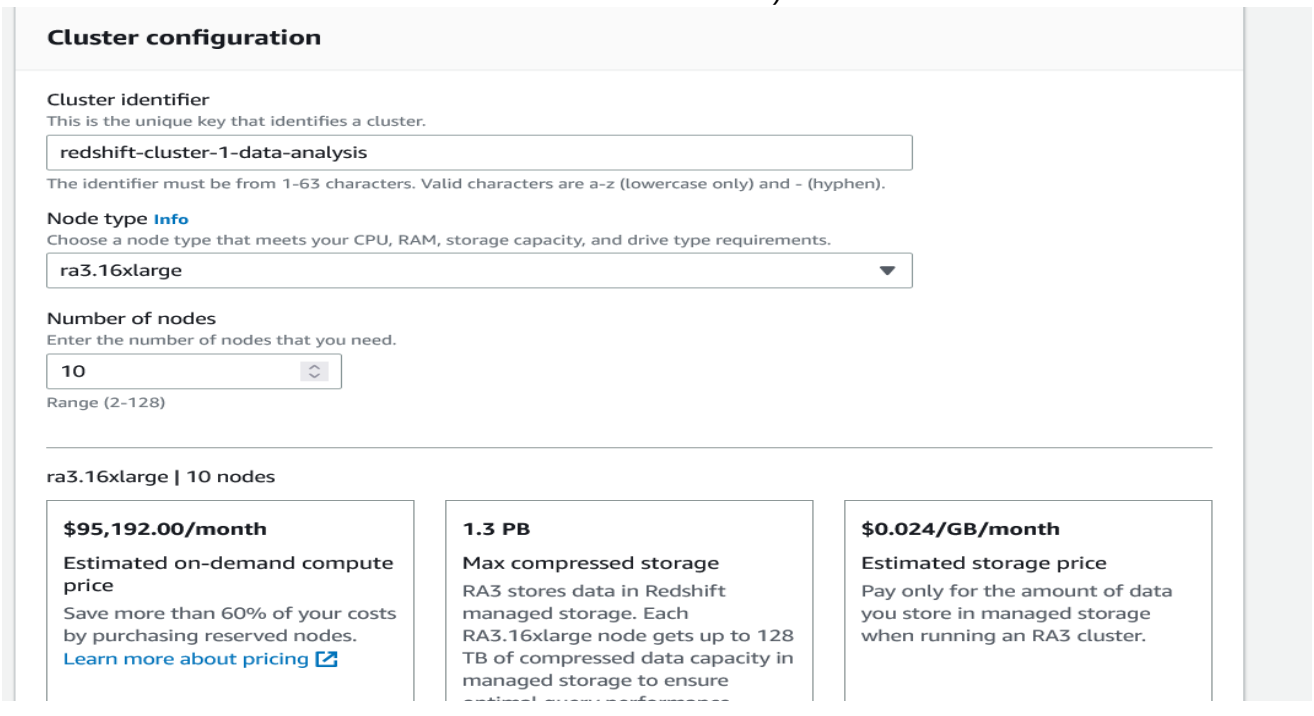
- In order to do that we must first upload the data to s3 then create an external table with the Partition assumption above in Redshift to create the table and load the data to Redshift from s3

- We decided to use an Apache Spark in order to clean the data under the assumption that the data is not fully clean. Then python script to upload data to s3 using Boto3 python packages

| limitations | Pros of spark |
|---|---|
| Low speed of process the data in normal script ⇒ so use Spark | Spark have a high speed of processing the data (for **1 PB** Elapsed Time 234 mins) [1] |

- Loading (Initial loading cost from the data center to the data S3) ⇒ Time and Cost
  - 200 TB ⇒ 1600000 Gbit / 5 Gbit/s = 320,000 second ⇒ 88.8 hours
  - 200TB data cleaning 58.5min ⇒ 1 hours
  - Total Time 89.8 h ⇒ 3.8 days
  - 200 TB compressed *5 To s3 = 1000 dollar

- Cost for cluster and storage of the data in Redshift is
  - For cluster we choose to use ra3.16xlarge * 10 nodes with vCPU = 480, Memory 960 Gib and storage 1.3 PB for the cluster, we see it's enough to work with 10 users working in 1PB data - worst case run select * will take 3 hours with 50Vcpu - (if we need more performance and the cost not critical then we increase the number of node)



- Total cost is Total Monthly cost:
  - First month $95,192.00 + $1000 from s3 = $96,192.00
  - Next months (without S3 cost) = $95,192.00 $ / month

Data flow:

1. As mentioned in our architecture above we will have a Python script that will be running locally and have a small normalization process (Example: assuming that we have an empty geo_id for a certain record. It will find its correct value according to the sensor_id based on our assumption above another example: if the timestamp column or value column is empty take the average of last 2 days next and two days previous) and then upload the data to s3 by using Boto3 like

```
s3_client = boto3.client('s3')
try:
    s3_client.upload_file('/path/to/local/file', 'your_bucket_name', 'destination_key')
    print("File uploaded successfully!")
except Exception as e:
    print("Error uploading file:", e)
```

2) now the data is in s3, next
To create an external table and partitions in Redshift we will use SQL queries like

```
"CREATE EXTERNAL TABLE external_table_name (
  column1 datatype,
  column2 datatype,
  ...
)
PARTITIONED BY (partition_column1 location, partition_column2 month, ...)
ROW FORMAT DELIMITED
FIELDS DELIMITED BY ','
LOCATION 's3://bucket_name/path/'
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/your-iam-role-arn';
```

Thanks to this method we have the data ready in RS for the analytical users to use
We use this design to made our system faster cheaper and simpler

Appendix

[1] https://opensource.com/business/15/1/apache-spark-new-world-record