PROUD2BE CLOUD!

On-Demand Data lakes on Amazon S3: how to tackle ETL and DataMart at scale

24 November 2022 - 7 min. read

AWS Glue | *AWS Organizations* | *AWS Transfer Family* | *Data Lake* | *ETL* | *SFTP*

Big Data started with a bang in the mid-2000s with the development of the MapReduce methodology at Google and kept growing at a breakneck pace with the continuous development of better and better tools: Apache Hadoop, Spark, Pandas have all been developed in this timeframe.

Concurrently, more and more Cloud providers and service integrators have started offering managed Big Data Data Lake solutions to meet the growing demand of companies that are more and more eager to analyze and monetize their data: from Cloudera to AWS Glue.

While in the meantime BigData has stopped being a buzzword and has been supplanted in this role by newer more appealing ones (such as Blockchain and Quantum computing) the need for companies to leverage data to better target their customers, optimize their products and refine their processes, has markedly increased.

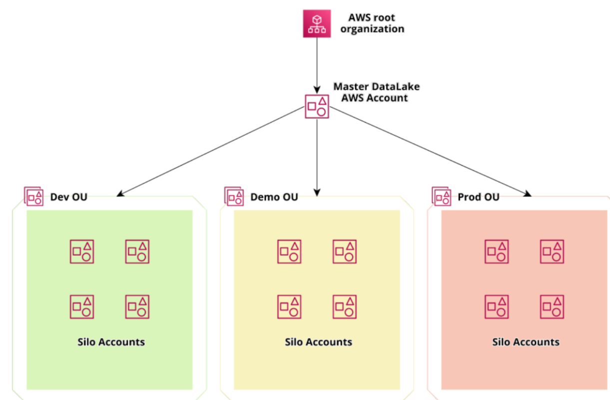
In this short article, we will describe how to create a **dynamic decentralized multi-account DataLake on AWS**, leveraging **AWS Glue**, **Athena**, and **Redshift**.

The Problem at hand

Usually, data lakes are unstructured data repositories that collect input data from heterogeneous data sources such as legacy SQL databases, document databases (e.g. MongoDB), Key value databases (e.g. Cassandra), and raw files from various sources (SFTP servers, Samba, Object storages).

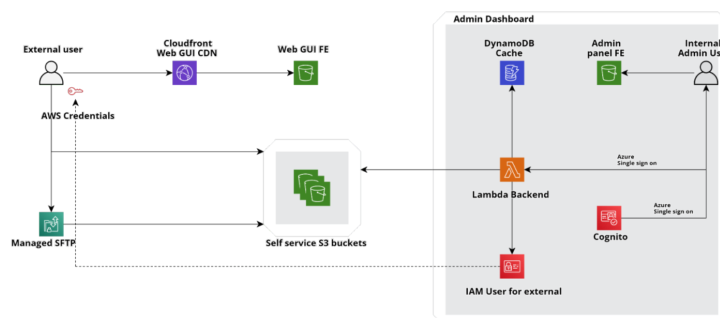
In this case, our requirement is to split the database so that each internal project in the customer Company structure can only access its segregated data silo and set up the ETL operations it needs. A selected number of users from the general administration will be able to access the data from several silos in order to read and aggregate the data for company-wide analytics, business intelligence, and general reporting.

In order to meet these requirements and assure **the strongest possible segregation** between different silos we decided to split the projects into several AWS accounts using AWS organizations. The accounts structure is represented in the diagram below:



In order not to get bored another requirement was to be able to create credentials for third parties to send data directly to the data lake either with APIs or SFTP.

This means that each account will not only contain the S3 bucket with the data and the Glue/Step Functions jobs needed to transform them, which are different for each silo, but also an admin web application to manage third-party access through temporary IAM credentials and a frontend deployed on Cloudfront to give a simple interface to users in order to load data on S3 directly.



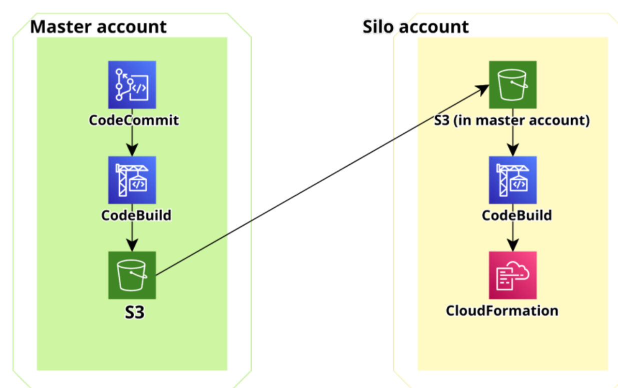
If you are wondering how we managed the SFTP part it is nearly straightforward, we just activated AWS Transfer Family service for SFTP with a custom Lambda to authenticate users with the same IAM credentials used for the Webapp access.

Thus, developing a relatively straightforward web application, we managed to create a fully serverless interface to our S3 buckets so that internal users can create temp credentials for external users to access an S3 dropzone where they can upload new files.

If you are wondering what type of black magic the Lambda(s) Backend in the diagram above is performing in order to do so without a database the answer is very simple: the state of our credentials vending machine is a collection of AWS resources (Users, Roles, Buckets) so we create them directly with CloudFormation and preserve the state in directly in the template!

Using cross-account pipelines, all the infrastructure and application components can be automatically deployed on each account in a self-service way: when an account is created in the relevant AWS Organization Unit a CloudFormation Stack Set is created in the account which deploys the basic infrastructure components, and an AWS CodePipeline to fetch the application code from the Master Datalake account and deploy it in the target Silo account.

In order to make this flow scalable to an arbitrary number of accounts, the deployment pipelines for the web app are splitted in 2 parts: the first leg is in the master account, uses our CodeCommit Repo as source, runs unit and integration tests in AWS codebuild and finally creates an application bundle (using AWS cloudformation package). Finally it zips together the CF template and the code bundle and uploads the zip in a versioned S3 bucket accessible from all the child Silo Accounts. The second part of the pipeline is in each Silo Account. It is configured to poll the bucket for changes and when it detects a new zip file a run is started, the zip is read, and deployed directly using cloudformation. The schema of the di pipe is shown below:



If you are an AWS geek like us you may have noticed that the generic Silo account should not be able to access the S3 bucket in Master where the first part of the pipeline deployed the bundle (obviously the bucket is private): if you remember each of those account is

created by Organizations so new account spawn and die, and we really don't want to list them in the master account S3 bucket policy.

The solution to this problem is rather straightforward: it is possible to create resource policies in such a way that only a given Organization Unit (OU) can access the bucket. This is a rather powerful trick because it is also applicable to glue Catalogs and allows one to share data directly with entire organization units. Furthermore it is still possible to query them with AWS Athena.

The bucket policy if the form:

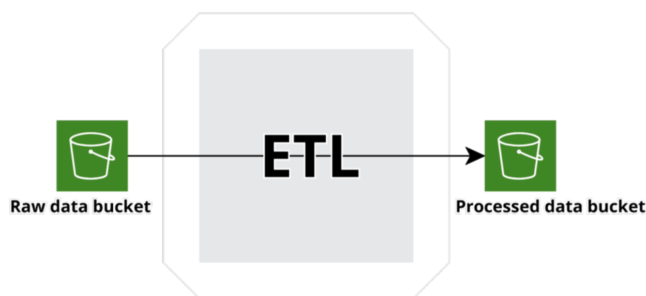
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:Get*",
            "Resource": [
                "arn:aws:s3:::bucket.code",
                "arn:aws:s3:::bucket.code/*"
            ],
            "Condition": {
                "ForAnyValue:ArnLike": {
                    "aws:PrincipalArn": "arn:aws:iam::*:role/*-codepipeli
ne-rl"
                },
                "ForAnyValue:StringLike": {                }
                    "aws:PrincipalOrgPaths": "o-****/r-****/ou-****-*****
*/ou-****-*******/*"

            }
        }
    ]
}
```

Where "o-****/r-****/ou-****-******/ou-****-*******/*" is the OU path and arn:aws:s3:::bucket.code the Arn of the bucket.

Ok: now we have tons of silo accounts, each with our custom Webapp that allows users to upload new files and using policies like the one above. The company master account and other accounts for global analytics will be able to access files on s3 and run Athena queries. We could even use Lakeformation cross account to further reduce access permission or increase granularity (more on this in a future article!)

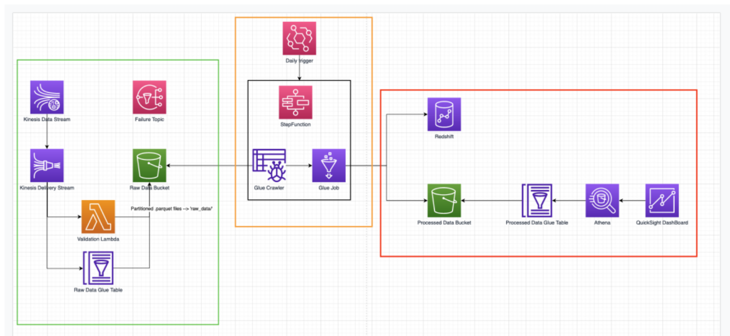
However each silo is effectively separated and independent, so data scientists / engineers working for each separate silo can deploy custom transform jobs without being able to interact with other silos.



The type of flow described above is common to all Silos and several flows are typically present in each Silo account.

Furthermore other AWS services can also be used to ingest different types of data in parallel with the custom web app we developed to directly expose s3 to the data producers.

Here we present the structure of one of the many flows implemented:



An AWS API Gateway has been exposed to receive data from several systems. API Gateway uses direct service integration with kinesis firehose to buffer data for a configurable time and size, transform the data in the common convenient parquet format and finally write the

data to S3. The data format is defined ab-initio and explicit in the glue table structure also used by kinesis to transform string json data to parquet. Data data are regularly exported to the Processed data output bucket using a Glue based AWS step function which first run a crawler on the Raw data Backet to index the new data and then runs a glue job to clean data up and make them adhere to the agreed upon data format; then, data are written in a Redshift data warehouse and on the Processed Data Bucket output s3 bucket. Finally a new crawler is run and the data (through Athena) read by Athena and imported into quicksight to create new dashboard.

Conclusion

We discussed how to create a dynamic multi account data lake on AWS and shared our solution to discuss some problems we encountered along the way.

What are your feelings about this? Let us know in comments!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Matteo Moroni

DevOps and Solution Architect at beSharp, I deal with developing Saas, Data Analysis, and HPC solutions, and with the design of unconventional architectures with different complexity. Passionate about computer science and physics, I have always worked in the first and I have a PhD in the second. Talking about anything technical and nerdy makes me happy!

Copyright © 2011-2022 by beSharp spa - P.IVA IT02415160189