ETL ORCHESTRATION ON AWS WITH AWS STEP FUNCTIONS

AWS Lambda | Data and Analytics | Serverless

beSharp | 26 November 2020

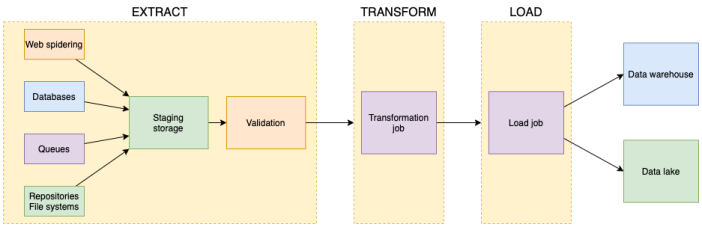
In the latest years, the engineering, governance, and analysis of data has become a very common talking point.

The need for data-driven decision-making, in fact, has grown the need of collecting and analyzing data in many ways and AWS has shown a particular interest in this field developing multiple tools for achieving these business goals.

Before being able to allow the figure of the data analyst to explore and visualize the data, a crucial step is needed. This procedure is commonly identified as ETL (extract, transform, and load) and, usually, it's far from being simple.

He who carries out this process has the responsibility of the following tasks:

- **Extraction**: data usually comes from numerous heterogeneous sources, such as databases, web spidering, data streams, semi-structured data etc. Due to the potential diversity of the data sources, a validation of the incoming data is mandatory, in order to not introduce information with an unexpected format or pattern.
- **Transformation**: after the load of the valid data into *staging storage*, a set of common transformations are applied to it. Typically, this stage is also identified as data preparation and it usually involves the removal of incomplete or inaccurate data (data cleansing), aggregation with other data, records deduplication, and all the steps of normalization and encoding.
- **Load**: finally, data that has been validated and transformed, is stored in the persistent data stores. These data stores may vary according to business needs. In fact, they can be identified according to different attributes. Lately, the most common data stores for ETL are data warehouses and data lakes. The first is generally used to store data with a strict schema definition in relational databases like Amazon Redshift. The latter, instead, is commonly made up of semi-structured data and it's mostly employed for machine learning, exploratory analysis, big data analysis, visualization etcetera. The coupling of Amazon S3 (for low-cost storage) and Amazon Athena (for fast and serverless queries on files), allows an excellent development of data lakes on AWS.



ETL on AWS

As briefly seen, a couple of AWS services have been cited as important components of an infrastructure capable of hosting an ETL process.

However, other services have been developed by AWS and they have already become the state of the art in the construction of data ingestion pipelines.

ETL Extraction on AWS

The extraction of data from which business analytics of an organization can profit from, may come with disparate paces and dimensions. From the hundreds of orders per second submitted to an e-commerce store during black Friday, to the ingestion of a monthly business report. The ETL infrastructure must always be ready to welcome the new information into the staging storage.

AWS services can help to accommodate such dissimilar business needs by making the data convey into the same repository, which is commonly identified by S3 buckets.

Depending on the mole of the expected data, it's possible to defer the validation of incoming files to different AWS services. In order to accomplish the best cost/performance ratio, it's necessary to choose between AWS Lambda for an event-driven pattern when small files are expected, and AWS Glue with scheduled batch job runs when data may reach volumes that may exceed AWS Lambda's computational limits.

ETL Transformation on AWS

The transformation of the incoming data is commonly a heavy duty job to be executed in batches. For this reason, the best candidates for this task are Glue resources. AWS Glue is based on serverless clusters that can seamlessly scale to terabytes of RAM and thousands of core workers

It is possible to run python scripts or either PySpark and Spark code for optimal scalability. Python shell glue jobs are mostly indicated for low-to-medium loads due to the fact that cannot scale to more than a single worker (4 vCPU and 16 GB of RAM).

However, although with Spark, Glue Jobs, and Glue Studio it's possible to create very meticulous transformation jobs, it's most likely that the new AWS Glue DataBrew service can fulfill this need with its very clear and complete web-interface.

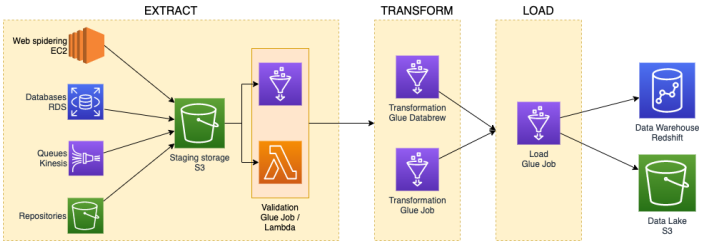
It's important to note that, in order to allow the Glue Jobs to retrieve the needed data from a single source, AWS Glue incorporates in its interface the Data Catalog. As the well-explanatory name explains, an archive of the data present in our data stores is maintained and used for ingestion. For

the purpose of maintaining and updating the catalog, an AWS Glue component called Crawler is used. The crawler, in fact, will give visibility on new files and partitions to the jobs trying to fetch data from the sources.

ETL Load on AWS

After the transformation process, a specific Glue Job or the same component employed in the previous step can finally store the valid, clean, and transformed data to the targets used for business analysis and visualization via, for example, Amazon QuickSight dashboards.

In order to preserve the privacy of the sensitive data that may progress in the pipeline, it's important to set-up the needed security measures such as KMS encryption for the data at rest in the buckets and databases and SSL protected transfers for data in transit. Moreover, it is a good practice to introduce obfuscation in PII stored in the domain.

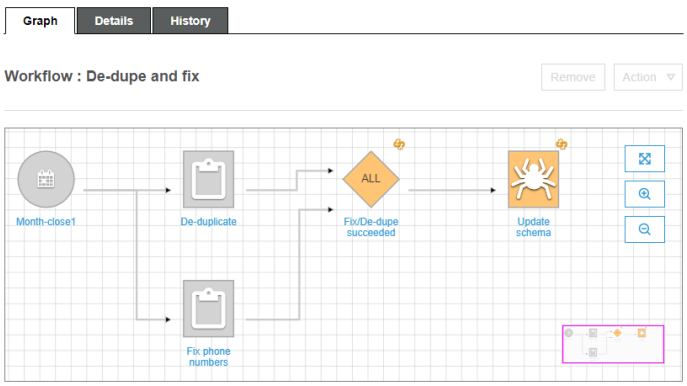


ETL orchestration on AWS

The management of the bits and bytes flowing in the whole ETL data pipeline is commonly not an easy task.

In order to apply appropriate governance on the data produced from the process, ad-hoc quality checks are usually performed. It's important, in fact, to check any inadequacy in business requirements such as the lack of data in the data lake due to an error in the code of the validation job.

AWS Glue has the tools to create workflows and triggers to build some sort of data pipelines. However, the possible solutions you can achieve are very limited by the lack of directives that allow loops, retries, proper error handling, and the invocation of other AWS services outside of AWS Glue.



In AWS, however, a specific tool allows scrupulous orchestration of serverless services: AWS Step Functions. This tool allows the management of retry logic and error handling to make our distributed applications better react in case of unexpected behaviours.

In the following sections, we are going to discover and employ Step Functions for the orchestration of a realistic use case of ETL.

AWS Step Functions

This AWS service allows the construction of highly scalable finite-state machines, that in the express configuration can handle up to one-hundred-thousands state changes per second.

It's important to note that a workflow built with this service is mainly composed of:

- States: the steps that are transitioned during the workflow.
- Directives: branching, retries, error handling, parallel processing, and loops.
- Service integrations: thanks to the different service integration of Step Functions with other AWS services, it is possible to invoke some of the many AWS serverless services. The almost inevitable AWS Lambda functions are one of these and they can act, however, as middleware invokers for the other AWS services not directly integrated with Step Functions.

All of the components just listed are then linked one to the other in an Amazon States Language, that is the JSON-based language used to generate Step Functions definitions.

Moreover, AWS Step Functions allows the monitoring of each run from the AWS console, supporting the real-time oversight of a workflow.

The use case

As said previously, we are going to put ourselves in the shoes of an architect that needs to deploy an orchestrated ETL flow. In the next sections, in fact, we are going to examine the business requirements of an imaginary customer and create a solution to accommodate them.

Customer's requirements

For the use case we will architect, we are going to pretend that an imaginary customer set some business requirements to guide our decisions:

- An audit table to always be aware of the state of each dataset is needed. This needs to give the possibility to build a web interface for the end-user in the future.
- At this moment, incoming files can be discerned in:
  - Dataset type A: daily files with sizes from 1 MB to 20 MB
  - Dataset type B: monthly reports with sizes from 15 MB to 50 MB
- Invalid files must be moved to a failed bucket for further human analysis.
- Immediate email notifications in case of failures in any step.
- The transformation and loading phases are managed with Spark code provided by the customer organization's data analysts.

Requirements analysis

In order to keep an audit trail of the state of each dataset flowing in and out, a DynamoDB table can be employed. This table will be automatically populated on the insertion of a new file in the input bucket via a Lambda function and a Cloudwatch Event.

Due to the directives allowed by Step Functions to interface with DynamoDB, it's possible to get, insert, update and delete records in the table. In this way, it's possible to directly update the state of each file whenever it's validated or transformed and loaded. The table will be structured in the following way:

- Type of dataset (daily or monthly) as the partition key
- Bucket name and key of the file as the sort key
- Ingestion state set to NEW when the file is created
- File size – can be employed in the future in case of the need to ingest bigger datasets. Step functions could be able to address the dataset to a Glue Job instead of a Lambda Function.

dataset_name	bucket_key	ingestion_state	file_size_in_mb
A	org.dev.staging.bucket#A/20201115.csv	NEW	8.22
A	org.dev.staging.bucket#A/20201116.csv	NEW	12.34
B	org.dev.staging.bucket#B/202010.csv	NEW	46.2

In view of the sizes of the two types of datasets that the customer expects to come in the workflow, AWS Lambdas are the best candidates to reach the best cost/performance ratio.

Moreover, lambda functions can be adopted to firstly retrieve the list of datasets that need to be ingested and move the invalid ones to the failed bucket.

It's possible to employ the service integration of AWS Step Functions with SNS to promptly notify the customer when an error state is caused by the failure of one of the ETL steps.

The state machine

It's been decided to start the Step Function in a scheduled manner, in order to reduce the costs of AWS Glue by ingesting file batches per-run.

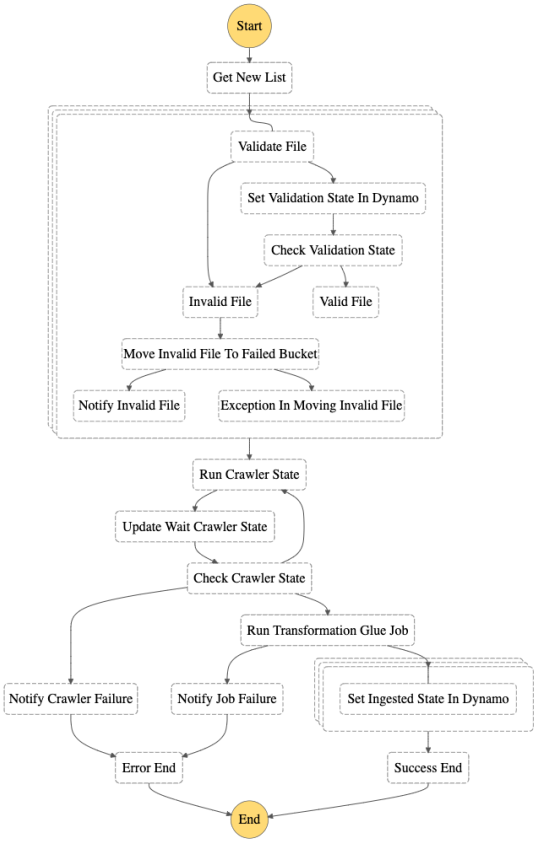
In order to be able to validate all the new files that came from the last Step Function's iteration, a lambda function needs to retrieve the records having the ingestion_state set to NEW. In this way, the workflow will be able to loop through the files list and accomplish the extraction and validation phase.

As we can see from the state diagram, the outcome of the validation lambda is stored in the dynamo table and then used to discern valid from invalid files.

Whenever the file validation lambda gets in an error state and the retry mechanism doesn't arise any success, the file is categorized as invalid and automatically moved to the failed bucket and an

email notification is sent to the customer.

When all iterations are completed and the invalid files have been discarded for further analysis, the Glue Crawler needs to be run in order to update the data catalog.



Unfortunately, at the time of writing, there's no service integration that allows the direct start of the crawler from Step Functions. For this reason, a lambda function will be in charge of this task. Via a wait state in Step Functions,

we can specify after how much time we'll check the crawler's run state again in a polling fashion.

At this stage, the data is validated and cataloged, and we are ready to transform and load it in the data lake.

Step Functions have been lately enriched with the possibility to synchronously run Glue Jobs thanks to a service integration with AWS Glue. This will allow us to run the transformation scripts provided by the data analysts without the need of a lambda invocation in the middle

In fact, the synchronous run of services from Step Functions allows the state machine to be stopped in the current state until the service hasn't responded back. In this way we'll be able to discern successful job runs from failures. In the first case it's possible to flow forward in the ETL workflow, while in the latter, a notification is sent to the customer again.

The next step is for auditability purposes only, in fact the step function will manage the update of the dynamo table setting the ingestion state to INGESTED for all the files that were loaded in this run.

Tips and tricks

We saw that the Lambda Functions employed in Step Functions are commonly simple invocations of other services, retrieval of lists and state checks for AWS resources spinning up.

For this purpose, in fact, it's possible to maintain a single codebase and a single Lambda Function to execute these simple tasks by selecting the method to be run according to the state in which the lambda was invoked.

An example of this behaviour is the state that retrieves the list of new files from the dynamo table:

```
"Get New List": {
   "Type": "Task",
   "Resource": "arn:aws:states:::lambda:invoke.sync",
   "Parameters": {
       "FunctionName": "arn:aws:lambda:eu-west-1:XXXXXXXXXXXX:function:glue-orc-sfn-l
ambda",
       "Payload": {
           "NeededState": "NEW",
           "DatasetName.$": "$.DatasetName",
           "SFNState.$": "$$.State.Name"
       }
   },
   "ResultPath": "$.NewFilesList",
   "Next": "New Files Loop"
}
```

This snippet of Amazon States Language defines the *Get New List* state. To the parameters passed to the Lambda Function, the state name is included to allow the selection of the correct Python method.

```
def action_switcher(sfn_state: str) -> function:
   switcher = {
       "Get New List": get_new_list,
       "Run Crawler State": run_crawler,
       "Move Invalid File To Failed Bucket": move_file_to_bucket
   }
   return switcher.get(sfn_state, lambda: None)
```

This has been possible thanks to the possibility to access the Step Functions context objects that may be very helpful in many definitions.

Conclusions

In this article we explored the world of ETL and how complex it can be to orchestrate a data pipeline in order to be error-proof, scalable and easily auditable.

With Step Functions we saw how a single service is enough to engineer a resilient solution to enable business analytics at any scale.

What about your ETL orchestration process on AWS? Tell us more about it!

Many hidden gems are still available for us to better work with everyday data and we can’t wait to show you more:

So, keep following us on **#Proud2beCloud**!

See you in 14 days with a new article!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it