Orchestrating ETL pipelines on AWS with Glue, StepFunctions, and Cloudformation

1 April 2021 - 7 min. read

Amazon S3 | AWS CloudFormation | AWS Glue | AWS Step Functions | ETL

Big Data analytics is becoming increasingly important to draft major business choices in corporations of all sizes. However collecting, aggregating, joining, and analyzing (wrangling) huge amounts of data stored in different locations with a heterogeneous structure (e.g. databases, CRMs, unstructured text, etc.) is often a daunting and very time-consuming task.

Cloud computing often comes to the rescue, by providing cheap and scalable storage computing and data lake solutions, and in particular, AWS is the pack leader with the very versatile Glue and S3 services which allow users to ingest transform, and normalize store datasets of all sizes. Furthermore, Glue Catalog and Athena allow users to easily run Presto-based SQL queries on the normalized data in S3 data lakes, whose results can easily be stored and analyzed in business intelligence tools such as QuickSight.

Despite the great advantages offered by Glue and S3 the creation and maintenance of complex multi-stage Glue ETL flows is often a very time-consuming task: Glue jobs are by their nature decoupled, and their code is stored in S3. This makes it very difficult to integrate different jobs and develop them in a well-structured software project.

A little help could come from Glue workflows: by using these integrated Glue pipelines, it is possible to run several different Glue jobs and/or crawlers automatically in a given order. However, this tool is lacking several features, very common in flow control tools, such as conditional branching, loops, dynamic maps, and custom steps.

A better alternative is AWS StepFunctions. StepFunctions is a very powerful and versatile AWS orchestration tool, capable of handling most AWS services, either directly or through lambda integrations.

In the following sections, we will explain how StepFunctions work and how to integrate and develop both infrastructure and code for Glue Jobs.

Why do I need StepFunctions?

Let's draft a very simple, yet realistic ETL job for data ingestion and transformation to explain why an orchestration service in general and, on AWS StepFunctions in particular, is an essential component in the data engineer toolbox. Here are the logical components for our toy ETL workflow:

1. Data should be ingested from a relational database. Multiple schemas and tables.
2. Ingested data should be loaded in S3 and crawled to extract a Glue DataCatalog for AWS Athena queries.
3. Several tables of the data catalog need to be joined using nontrivial rules to create a dataset on S3 to be used by a Machine Learning job for customer segmentation.
4. The output of the data segmentation job should be stored both in the S3 data lake and be written back to the relational database for access by other corporate tools.

These four steps describe a relatively basic but very common use case. Now let's try to draft a list of steps we need to execute in AWS Glue in order to complete the described workflow:

1. Crawl the original database through a JDBC connection.
2. Use a Glue Job to move the data from the database to S3. Some tables may use bookmarks but others may not.
3. Crawl the target S3 bucket.
4. Run a dedicated Glue Spark job to run the join operation on the S3 data lake. Write the results to another S3 partition or bucket.

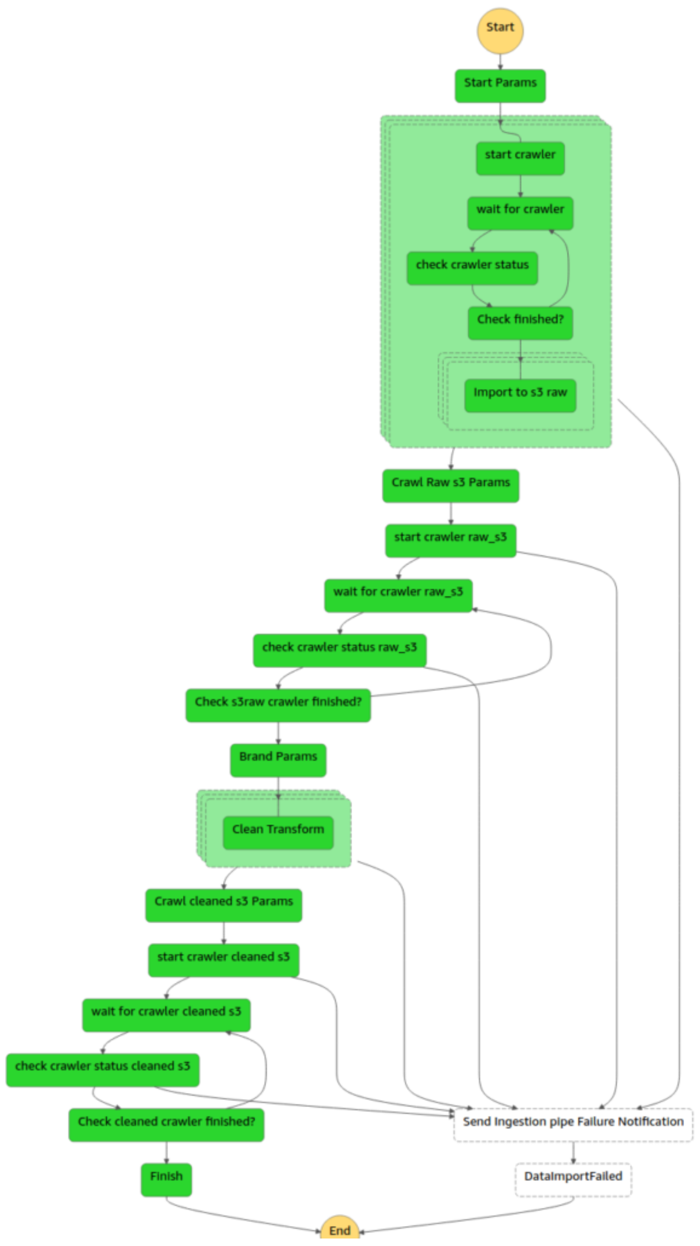
5. Crawl the target partition to make the Join results easily queryable with AWS Athena.
6. Execute the ML Job (SageMaker or the new Glue ML jobs).
7. Crawl the resulting dataset.
8. Run a final Glue ETL job to upload the new dataset to the original database.

All these steps need to be executed in the given order, and in case of problems, we would like to be notified and have a simple way to understand what went wrong.

Without AWS StepFunctions, manually managing these steps would be hellish, and we would probably need an external orchestration tool or to create a Custom orchestration script to be executed on an EC2 or on a Fargate container.

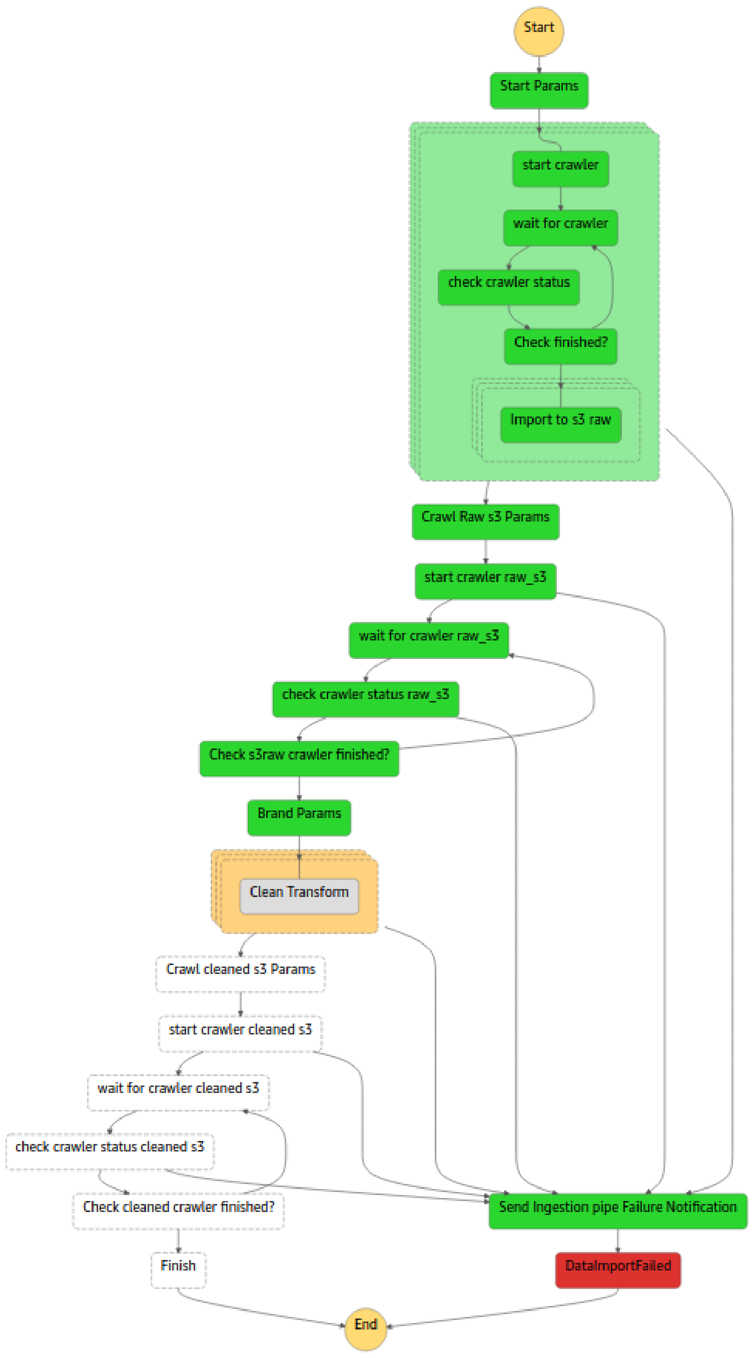
But why bother? AWS StepFunctions do all this for us, and by being able to directly interact with many AWS services, many integrations are a breeze: for example, with few lines of StepFunctions language, we can catch all the errors in a pipe and forward them to an SNS topic, in order to receive an email in case of error (or a slack notification, SMS or whatever you like more).

Managing complex flow thus becomes safe and relatively easy. Here is an example of a quite contrived flow:



Simple StepFunctions flow

If any of these steps fail we'll receive an email notification from the SNS topic, have visual feedback of the failed step, and the corresponding logs.



Example of failed step with logs

StepFunctions thus seems to be a jack of all trades, with a lot of good features, and no significant drawbacks, however, as we all know, this is almost always not true in IT, so which is the catch?

The real problem for StepFunctions is **code management**: the step function language is a declarative JSON template (see https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html), which is quite a pain to write and maintain even using dedicated tools such as the visual studio code plugin (see https://aws.amazon.com/blogs/compute/aws-step-functions-support-in-visual-studio-code).

Furthermore, it would be wonderful to be able to maintain both the StepFunctions code and the Glue Jobs, and the eventual Lambda code in a single integrated project.

Cloudformation with Troposphere or AWS CDK

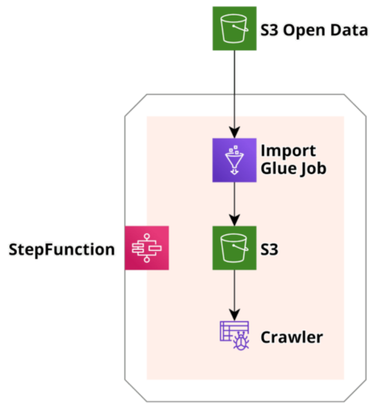
The most obvious instrument we can use to maintain StepFunctions, the Glue Jobs, and the rest of our ETL infrastructure in an integrated way, is Cloudformation as a deployment tool for everything. However, Cloudformation code is a declarative YML/JSON language not too different from StepFunctions code, and including that code in these templates is usually quite painful since involves including it as a JSON string in our Cloud Formation YML.

A much more effective solution is to create our Cloudformation template using a high-level programming language and leveraging the AWS CDK Cloudformation software development framework (https://aws.amazon.com/cdk/) which supports many languages (TypeScript, Python, and Java).

If you decide to use Python, which usually makes sense, since your ETL jobs will probably be written in Python, you also have the option to use Troposphere instead of AWS CDK, as a Cloudformation framework, which is much more versatile in several situations.

Furthermore, you can author the StepFunctions using the python Step Functions Framework (https://docs.aws.amazon.com/step-functions/latest/dg/concepts-python-sdk.html) as we'll do in the following example (Troposphere + Python step function SDK).

In this very simple example, we want to demonstrate how to create a simple Flow to download Covid Data from a public AWS OpenData S3 bucket, save a small subset of them in a different S3 bucket, and crawl them in order to be prepared for Athena queries. You can extend this basic working example at will! Here is a basic sketch of the infrastructure:



Basic example infrastructure

First of all, let's install the AWS CLI (https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html) and the python requirements:

```
pip install troposphere stepfunctions.
```

Once the installation is done you can download our repository and you'll find a **troposphere_main.py** file, which contains the troposphere representation of the whole infrastructure (see sketch), other folders containing the python code of the various Lambda functions (start_crawler, check_crawler status), and a README file explaining how to run the project. After this, we'll need to create an S3 bucket as support for the Cloudformation deployment with the name you prefer.

Following the instructions in the README, we can just run the main file by running in a console python troposphere_main.py. Executing this script, we'll compile the python troposphere code to a Cloudformation compliant JSON format. Once this is done, we are ready to run

```
aws cloudformation package --template-file troposphere_main.json --s3-bucket <YOUR CLOUDFORMATION S3 BUCKET> --s3-prefix '<THE PATH YOU PREFER>' --output-template-file troposphere_main.yml
```

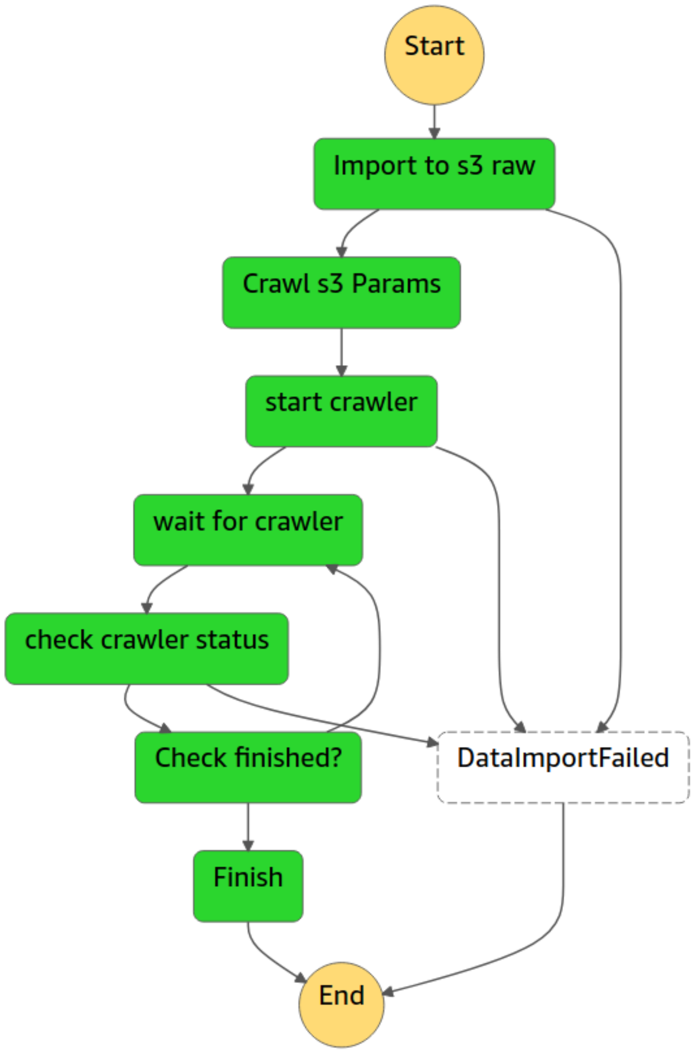
This command takes as input the JSON file created by Troposphere, and uploads to S3 the Glue and Lambda functions code referenced as local paths giving back another

Cloudformation template (this time YML), where the local paths references have been changed in the corresponding S3 references (for further info, take a look at this).

Finally we are ready to deploy the Cloudformation template using the command:

```
aws cloudformation deploy --template-file ./troposphere_main.yml --stack-name
testStepfunctionsStack --capabilities CAPABILITY_NAMED_IAM
CAPABILITY_AUTO_EXPAND
```

This we'll create the testStepfunctionsStack which contains the infrastructure described above. Now you can go to the AWS StepFunctions console and run the newly created function (test-stepfunctions-glue), the flow will run its course, and you'll import the Covid data.



Our example flow completed

While this is just a basic example, it is important to note that all the code presented is in the same project and thus you can easily extend the flow at will without losing control of the various components, just use Git for version control and Cloudformation for the deployments!

Takeaways

We showed that step functions are a great way to orchestrate AWS-based flows in general and ETL pipelines in particular! Furthermore, we shared an example of how to use Troposphere and Python StepFunctions SDK to develop, in a single python project, both a step function and the code of its various components.

So, this is it! As always, feel free to comment in the section below, and reach us for any doubt, question or idea!

See you on **#proud2becloud** in a couple of weeks for another exciting story!

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!

Alessandro Gaggia

Head of software development at beSharp and Full-Stack Developer, I keep all our codebases up-to-date. I write code in almost any language, but Typescript is my favorite. I live for IT, Game design, Cinema, Comics, and... good food. Drawing is my passion!
